

# **CONTEXT-SENSITIVE ASYNCHRONOUS MEMORY**

A General Experience-Based Method for  
Managing Information Access in Cognitive Agents

A Thesis  
Presented to  
the Academic Faculty

by

Anthony G. Francis, Jr.

In partial fulfillment  
of the requirements for the Degree  
Doctor of Philosophy in Computer Science

Georgia Institute of Technology

Atlanta, Georgia  
August 2000

Copyright © 1997-2000 Anthony G. Francis, Jr.

## **Context Sensitive Asynchronous Memory**

---

by  
Anthony G. Francis, Jr.

Permission is granted to create and distribute copies of this dissertation for academic research, as long as it is distributed without changes and this copyright notice and permission notice are included on all copies.

For additional permissions, please contact the author.

# **Context-Sensitive Asynchronous Memory**

---

Approved:

---

Ashwin Ram, Chairman  
College of Computing

---

Kurt Eiselt  
College of Computing

---

Ashok Goel  
College of Computing

---

Janet L. Kolodner  
College of Computing

---

Nancy Nersessian  
School of Public Policy

Date Approved: \_\_\_\_\_

# DEDICATION

---

*To Tony and Susan Francis.*

# ACKNOWLEDGEMENTS

---

*To those who made it possible.*

First and foremost, I thank my parents, who gave me my life, every opportunity within their power, and the encouragement and support I needed to succeed. Mom, Dad: without you none of this would have been possible.

Many thanks to Ashwin Ram, who, long before he became my advisor, often lent his ear in his office across the hall to many a crazy idea I had forming in my head. Later, as my advisor, he gave me guidance and wisdom, along with the support and encouragement I needed to pursue this very large thesis topic on its own terms.

Somewhere between my parents and advisor, formative influences put me on the path to science, computing, and artificial intelligence. I never met Douglas Hofstadter or Isaac Asimov, but through their books *Gödel, Escher, Bach: an Eternal Golden Braid* and *The Bicentennial Man* they sparked my interest in artificial intelligence at a formative age. This interest flourished under my high school mentor Doctor Latimer, whose inspiration gave me the confidence to meet any challenge and whose loss still saddens me deeply. I must also thank Mr. O’Leary and Mike Farmer for encouraging my interests in science and computing.

When I first came to Georgia Tech (oh so many years ago!) as an undergraduate, Kurt Eiselt helped me survive my senior year and has served as my mentor ever since. Janet

Kolodner helped me make the transition to graduate school, first by prodding me to apply and then by supporting and guiding my first year of research. In addition I also want to thank all the others who supported or aided me along my long and arduous PhD journey — beyond my advisor, these include the Air Force Office of Scientific Research; Manuela Veloso and Mike Cox at CMU; Marie desJardins, Mike Wolverton, and all my other friends at SRI; and of course Takashi Mizokawa and Naoki Sadakuni and all my friends at Yamaha.

Making the path bearable was the camaraderie: I miss the days of the AI lab at the College of Computing, sharing cubicles, thoughts, ideas, program code and comic books with Kenneth Moorman, Gordon Shippey, Paul Rowlands, Marin Simina, Mark Devaney, Juan Carlos Santamaria, and all the rest.

Later, Ashwin, Mark and I got together and founded Enkia, where we worked together to build things that none of us could have done alone. Supporting us were Ray Liuzzi at Air Force Rome Labs, Lloyd Solomon and others at the ATDC, Barry Rosenberg at GTRC, and many others.

Technical thanks are due to many. First, thanks to the talented authors of the Open Source and Free Software platforms upon which much of this research was conducted, including the Linux operating system, the Xemacs text editor, and the WWW::Search library for Perl. Second, thanks are also due to Franz Incorporated for making its Allegro Common Lisp language, in which the Nicole, Nicole-MPA and core Nicole-IRIA

systems were written, freely available on Linux for research. Thanks are also due to Sun Microsystems for making its Java development kit, in which the CAPABLE interface was written, similarly freely available. Thanks to Kenneth Moorman for providing a portion of the ISAAC knowledge base, which was used to test and exercise the CRYSTAL and MOORE systems during development. Finally, thanks are due to Mark Devaney for building the Perl data harvesting tools used in Nicole-IRIA, and for many hours spent categorizing the data sets used in the experimental evaluations; thanks are also due to Ashwin Ram for hours spent at the Nicole-IRIA interface collecting hundreds of data points by hand during the experimental tests.

Thanks are also due to my thesis committee for helping me write the best thesis that I could: to Ashwin again for reading many, many drafts; to Kurt for tolerating my frequent visits to his office to bend his ear; to Janet for helping me make my scientific contributions explicit; to Nancy Nersessian for helping me make place my work in philosophical perspective; and to Ashok Goel for helping me make sure I crossed all the *t*'s and dotted all the *i*'s necessary to ensure a quality dissertation.

And for this Centaur no list of acknowledgements would be complete without the Edge — David, Fred & Sophia, Derek, Cater, and Mike; William, Stuart, Eric, Gordon & Thavy, Yvette, Mark, and Eric & Cholly; and Shannon, Erin, Patsy. Thanks for being there for me when I needed you.

And this one goes out to Big G, who put that dinosaur book in the Greenville County Library where I would find it. You know who you are.

—*the Centaur*, 2000.07.18AD



# TABLE OF CONTENTS

---



---

<i>Dedication</i>	<i>iv</i>
<i>Acknowledgements</i>	<i>v</i>
<i>Table of Contents</i>	<i>ix</i>
<i>Summary</i>	<i>xvi</i>
<b>PART I. Foundations</b>	<b>1</b>
<b>CHAPTER I. Introduction</b>	<b>2</b>
1.1. The Problem	6
1.1.1. What are Cognitive Agents?	7
1.1.2. What is Memory Retrieval?	8
1.1.3. Examples of Memory Retrieval in Cognitive Agents	11
1.1.4. Unpacking The Problem	13
1.1.5. Why the Problem is Hard	27
1.1.6. Desiderata for a Solution	30
1.1.7. Possibilities Raised By Interacting Constraints	32
1.1.8. Keys to A Solution	33
1.2. The Solution	35
1.2.1. Distinguishing Features of Context-Sensitive Asynchronous Memory	36
1.2.2. The Context-Sensitive Asynchronous Memory Approach	40
1.3. Applying The Solution	46
1.4. Scope of the Approach	50
1.5. The Claims	57
1.6. The Evidence	59
1.6.1. Claim 1: Interleaving Memory with Reasoning and Action is Useful	59
1.6.2. Claim 2: Context-Sensitive Asynchronous Memory Enables Interleaving	62
1.6.3. Claim 3: A Rich Representation Makes Context-Sensitive Asynchronous Memory Domain-Independent	66
1.6.4. Claim 4: Communication, Integration and Control are Required	67
1.7. Benefits and Contributions	70
1.7.1. Benefits to Users and Tasks	70
1.7.2. Contributions to Artificial Intelligence	71
1.8. Outline of the Thesis	74
<b>CHAPTER II. Related Work</b>	<b>77</b>
2.1. Existing Approaches and their Limits	79
2.2. Evaluating Memory Approaches	80
2.2.1. Functional Capabilities of Memory Systems	82
2.2.2. Performance Characteristics of Memory Systems	83
2.2.3. Technologies for Memory Retrieval	85
2.3. Apparent Tradeoffs and Opportunities	88
2.4. Meeting the Desiderata	91
2.4.1. Desideratum 1: Able to store a wide variety of information	92

2.4.2. Desideratum 2: Provides a general method for accessing information	99
2.4.3. Desideratum 3: Scales to large multifunction knowledge bases	108
2.4.4. Desideratum 4: Manages cost of retrieval	118
2.4.5. Desideratum 5: Preserves accuracy of retrieval (in the face of resource limits)	122
2.4.6. Desideratum 6: Exploits task/environmental information	126
2.4.7. Desideratum 7: Exploits extra resources if available	137
2.4.8. Desideratum 8: Potentially interleavable with reasoning	139
2.4.9. Desideratum 9: Provides guidelines for reasoning integration	140
2.5. Conclusion	142
<b>Part II. Approach</b>	<b>144</b>
<b>CHAPTER III. Context-Sensitive Asynchronous Memory</b>	<b>146</b>
3.1. Possibilities Inherent in Memories for Cognitive Agents	150
3.2. The Context-sensitive Asynchronous Memory Approach	152
3.3. Asynchronous Retrieval	156
3.3.1. Reified Retrieval Requests	156
3.3.2. Features of a Retrieval Monitor	160
3.3.3. How the Retrieval Monitor Processes Retrieval Requests	164
3.4. Context Sensitivity	167
3.4.1. Activation, Perturbance, Fanout, and Decay	168
3.4.2. Sources of Perturbance	171
3.4.3. Types of Context-Directed Spreading Activation	171
3.4.4. The Context-Directed Spreading Activation Algorithm	175
3.4.5. Semantic Distance and Context-Directed Spreading Activation	181
3.5. Cost Control Policy	189
3.5.1. Applying Additional Cost Control to CDSA	193
3.5.2. Order Analysis of Context-sensitive Asynchronous Memory	194
3.6. The Experience Store	206
3.6.1. Constraints Imposed by Context-Sensitive Asynchronous Memory	208
3.6.2. Expressive Power	211
3.6.3. Creating a Unified Store of Knowledge	216
3.7. Getting Answers with Context-Sensitive Asynchronous Memory	220
3.8. Applicability of the Approach	226
3.8.1. Traditional Spreading Activation and its Limits	228
3.8.2. How Context-Directed Spreading Activation Uses Context	229
3.8.3. Types of Contextual Information	229
3.8.4. Available Sources of Context	230
3.8.5. Profile of Environments In Which CDSA Will Provide Benefits	232
3.8.6. Some Limitations	236
3.9. Conclusion	240
<b>CHAPTER IV. Integration Mechanisms</b>	<b>242</b>
4.1. Demands of Context-Sensitive Asynchronous Memory	244
4.2. Demands of Dynamic, Unpredictable Worlds	247
4.3. Efficient Handling of Spontaneous Retrievals	251
4.4. Constructing Reasoning Systems That Satisfy the Demands	253
4.4.1. Working with an Asynchronous Memory	256
4.4.2. Providing Feedback to Context-sensitive Memory	259
4.4.3. Responding to Spontaneous Retrievals	260
4.5. Integration Mechanisms	262
4.5.1. Retrieval Handlers	266
4.5.2. The Retrieval Evaluator	267

4.5.3. The Retrieval Dispatcher	268
4.5.4. Relevance Determination	273
4.5.5. Integration Processor	277
4.6. Putting the Components Together	279
<b>CHAPTER V. Experience-Based Agency</b>	<b>281</b>
5.1. Principles of Experience-Based Agency	283
5.2. Architecture of an Experience-Based Agent	285
5.3. Distinctive Features of Experience-Based Agency	291
5.3.1. Working Memory	294
5.3.2. Task Processing	299
5.4. Further Support for Memory and Reasoning Integration	311
5.4.1. Impasse-Driven Retrieval Request Generation	311
5.4.2. Impasse-Driven Integration Mechanism Invocation	313
5.4.3. Significance of Impasse-Driven Memory Processes	315
5.5. General Architectural Issues	317
5.6. Benefits of the Experience-Based Agent Approach	319
5.6.1. What Experience-Based Agency Provides	319
5.6.2. What Experience-Based Agency Enables	320
5.6.3. Exploiting Experience-Based Agency	321
5.6.4. Living up to Experience-Based Agency's Potential	323
5.7. Summary	324
<b>Part III. Evaluation</b>	<b>326</b>
<b>CHAPTER VI. Methodology</b>	<b>328</b>
6.1. Overview	328
6.2. Goals of the Evaluation	329
6.3. Methodology of the Study	330
6.4. Applying the Model	331
6.5. Conducting the Evaluations	333
6.6. Fidelity of the Study	334
6.7. Outline and Expected Results of the Study	335
<b>CHAPTER VII. Implementation</b>	<b>338</b>
7.1. Overview of Nicole	340
7.1.1. Nicole Core Modules	341
7.1.2. Nicole Development Tools	343
7.1.3. Nicole-Based Applications	344
7.1.4. Nicole Support Tools	346
7.2. Knowledge Representation in Nicole	348
7.3. Memory Retrieval in Nicole	352
7.3.1. Memory Retrieval Requests	352
7.3.2. Memory API in MOORE	354
7.3.3. Matching in MOORE	356
7.3.4. Implementing CDSA in MOORE	357
7.3.5. The MOORE Algorithm	360
7.4. Working Memory in Nicole	361
7.5. Task Control in Nicole	363
7.6. Sample Reasoning Tasks in Nicole	371
7.7. Additional Features	373
<b>CHAPTER VIII. Case Study: Planning</b>	<b>376</b>
8.1. Overview	376

8.2. Goals of the Case Study	376
8.3. Methodology of the Case Study.	377
8.4. Execution of the Study	378
8.5. Target Task: Planning	379
8.5.1. Least Commitment Planning	379
8.5.2. Case-Based Planning	380
8.5.3. Prior Work in Least-Commitment Case Based Planning	380
8.6. The Problem: Solving Hard Problems with Past Experience	382
8.7. The Solution: Combining Multiple Past Experiences	383
8.7.1. Principles Behind an Experience-Based Agent Solution	383
8.7.2. Multi-Plan Adaptation	385
8.7.3. Controlling Multi-Plan Adaptation	389
8.7.4. Expected Benefits of Multi-Plan Adaptation	392
8.7.5. Prior Work in Plan Reuse and Multi-Plan Adaptation	393
8.8. Executing the Study	396
8.8.1. Planning Frameworks	396
8.8.2. Planning Domains	400
8.8.3. Case Libraries	404
8.8.4. Supporting Work	405
8.9. Fidelity of the Study	405
8.10. Hypotheses Tested	406
8.11. Results	409
8.11.1. Evaluation 8.1: Validity and Usefulness of the Multi-Plan Adaptation Algorithm	409
8.11.2. Evaluation 8.2: Experience-Based Search-Space Multi-Plan Adaptation	411
8.11.3. Evaluation 8.3: Detailed Analysis of Multi-Plan Adaptation	413
8.11.4. Evaluation 8.4: Controlling Ordering And Binding Problems In Integrative Reasoning	416
8.11.5. Evaluation 8.5: Experience-Based Metaplanning Multi-Plan Adaptation	418
8.12. Explanation and Analysis	419
8.12.1. Usefulness of Multi-Plan Adaptation	420
8.12.2. Partially-Ordered Planning as a Platform for Integrative Reasoning	421
8.12.3. Integrative Reasoning Using Cases as Extended Operators in State Space	424
8.12.4. Integrative Reasoning Can Be Driven By Spontaneous Retrieval	427
8.12.5. Controlling Ordering And Binding Problems In Integrative Reasoning	429
8.12.6. Integrative Reasoning Can Be Achieved Using Metaplanning	430
8.12.7. Interactive Multi-Plan Adaptation Is Useful	431
8.13. Lessons Learned	431
8.14. Overcoming the Caveats	435
<b>CHAPTER IX. Case Study: Information Retrieval</b>	<b>438</b>
9.1. Overview	438
9.2. Goals of the Case Study.	438
9.3. Methodology of Study	439
9.4. Execution of the Study.	440
9.5. The Task: Information Retrieval	441
9.6. The Problem: Getting Good Answers to Bad Questions	442
9.6.1. Modern Information Retrieval	442
9.6.2. Existing Systems and their Limits	445
9.6.3. Building on Existing Work	447
9.7. The Solution: Experience-Based Information Retrieval	452
9.7.1. Relevance as Reminding in Information Agriculture	452
9.7.2. The Information Retrieval Intelligent Assistant	454
9.7.3. Architecture of Nicole-IRIA	457

9.7.4. Details of the Approach	461
9.7.5. Structure of an IRIA-Based Application.	464
9.8. Executing the Study	466
9.8.1. Applications Constructed	466
9.8.2. Datasets Constructed	468
9.9. Fidelity of the Study	470
9.10. Hypotheses Tested	471
9.11. Results	472
9.11.1. Evaluation 9.1. Providing Useful Answers to Individual Users	473
9.11.2. Evaluation 9.2. Providing Useful Answers to Workgroups.	477
9.11.3. Evaluation 9.3. Support for Educational Applications.	480
9.11.4. Evaluation 9.4. Support for Other Applications.	482
9.11.5. Evaluation 9.5. Analysis of Serendipitous Results.	483
9.12. Explanation and Analysis	484
9.12.1. Context-sensitive asynchronous memory can recommend useful information	484
9.12.2. Feedback can improve context-sensitive asynchronous retrieval	485
9.12.3. Asynchronous retrieval is useful	486
9.12.4. Generality of the context-sensitive asynchronous memory approach:	487
9.12.5. Summary of Results	488
9.13. Lessons Learned	490
9.14. Conclusions	495
<b>CHAPTER X. Feasibility of the Approach</b>	<b>497</b>
10.1. Overview	497
10.2. Goals of the Feasibility Study	498
10.3. Methodology of the Study	498
10.4. Applying the Model	499
10.5. Conducting the Study	499
10.5.1. Memory Systems Constructed	500
10.5.2. Retrieval Problem Sets	502
10.5.3. Knowledge Structure Strategy	504
10.5.4. Knowledge Bases Constructed	507
10.5.5. Context Sets Constructed	509
10.5.6. Other Efforts	510
10.6. Fidelity of the Study	511
10.7. Hypotheses Tested	512
10.8. Results	515
10.8.1. Evaluation 10.1: Representational and Retrieval Generality	516
10.8.2. Evaluation 10.2: Evaluation of Scaleup I: Retrieval Performance Across Knowledge Bases	520
10.8.3. Evaluation 10.3: Evaluation of Scaleup II: Interactions between Platform and Scaleup.	530
10.8.4. Evaluation 10.4: Evaluation of Scaleup III: Simulated Accumulation Tests	532
10.8.5. Evaluation 10.5: Evaluation of Scaleup IV: Followup Accumulation Tests	536
10.8.6. Evaluation 10.6: Evaluation of Scaleup V: Effect of Seed Knowledge Bases	542
10.8.7. Evaluation 10.7: Comparative Performance Analyses	545
10.8.8. Evaluation 10.8: Contribution of Context I	549
10.8.9. Evaluation 10.9: Contribution of Context II	552
10.8.10. Evaluation 10.10: Tests of Cost Control	554
10.8.11. Evaluation 10.11: Analysis of Asynchronous and Incremental Search	555
10.9. Explanation and Analysis	557
10.9.1. Desideratum 1: Able to store a wide variety of information	558
10.9.2. Desideratum 2: Provides a general method for accessing information	559
10.9.3. Desideratum 3: Scales to large multifunction knowledge bases	560

10.9.4. Desideratum 4: Manages cost of retrieval	566
10.9.5. Desideratum 5: Preserves accuracy of retrieval	569
10.9.6. Desideratum 6: Exploits task/environmental information	571
10.9.7. Desideratum 7: Exploits extra resources if available	572
10.9.8. Desideratum 8: Potentially interleavable with reasoning	573
10.9.9. Desideratum 9: Provides guidelines for reasoning integration	574
10.10. Analysis of Sources of Power	575
10.10.1. Structural/functional constraints on the knowledge base	578
10.10.2. Content of the knowledge base	581
10.10.3. Content of the context	583
10.10.4. Context-directed spreading activation algorithms	584
10.10.5. Asynchrony	584
10.10.6. Interleavability	586
10.10.7. Integration mechanisms	587
10.11. Lessons Learned	588
10.12. Conclusion	590
<b>Part IV Impact</b>	<b>592</b>
<b>CHAPTER XI. Conclusions</b>	<b>593</b>
11.1. Scope of the Approach	595
11.2. The Model	597
11.3. Evaluations and Results	599
11.4. Benefits	602
11.5. Lessons Learned	604
11.5.1. Lessons Learned in Memory Retrieval	605
11.5.2. Lessons Learned in Reasoning and Task Architecture	607
11.5.3. Lessons Learned in Planning	608
11.5.4. Lessons Learned in Information Retrieval	609
11.6. Contributions	611
11.6.1. Contributions to Memory	612
11.6.2. Contributions to Agent Systems	616
11.6.3. Contributions to Task Control	617
11.6.4. Contributions to Planning	618
11.6.5. Contributions to Information Retrieval	619
11.7. Future work	620
11.7.1. Open Research Questions	621
11.7.2. Extending the Implementations	622
11.7.3. Future Work	624
11.8. Conclusion	628
<b>Appendices</b>	<b>631</b>
<b>Appendix A Planning Domains</b>	<b>632</b>
A.1 The Simple Travel Domain	632
A.2 The Stinger Missile Domain	634
A.3 The Abstract-3 Domain	639
<b>Appendix B Problem Sets</b>	<b>646</b>
B.1 The X2 Problem Set	646
B.2 The Information Retrieval Problem Set	651
B.3 The MetaSpy Problem Set	652
<b>Bibliography</b>	<b>654</b>



# SUMMARY

---

*The dissertation distilled.*

Retrieving useful answers from large knowledge bases given under-specified questions is an important problem in the construction of general intelligent agents. The core of this problem is how to get the information an agent needs when it doesn't know how to ask the right question and doesn't have the time to exhaustively search all available information.

*Context-sensitive asynchronous memory* is a model of memory retrieval that solves this problem. The context-sensitive asynchronous memory approach exploits feedback from the task and environment to guide and constrain memory search by interleaving memory retrieval and problem solving. To achieve this behavior, a context-sensitive asynchronous memory uses an asynchronous retrieval system to manage a context-sensitive search process operating over a content-addressable knowledge base. Solutions based on this approach provide useful answers to vague questions efficiently, based on information naturally available during the performance of a task.

The core claims of this approach are:

- **Claim 1:** An efficient, domain-independent solution to the problem of retrieving useful answers from large knowledge bases given under-specified queries is to



interleave memory retrieval with task performance and use feedback from the task or environment to guide the search of memory.

- **Claim 2:** Interleaving memory retrieval with and exploiting feedback from task performance can be achieved in a domain-independent way using a context-sensitive, asynchronous memory retrieval process.
- **Claim 3:** A rich, reified, grounded semantic network representation enables context-sensitive memory retrieval processes to retrieve useful information in a domain-independent way for a wide variety of tasks.
- **Claim 4:** To effectively use a context-sensitive asynchronous memory to retrieve useful answers, a task must be able to work in parallel with a memory process, communicate with it, provide feedback to it, and must possess integration mechanisms to incorporate asynchronous retrievals provided by the memory.

The context-sensitive asynchronous memory approach is applicable to tasks and domains which exhibit the following criteria: problems are difficult to solve, questions are difficult to formulate, a large knowledge base is available yet contains only a small selection of relevant information, and, most importantly, the environment is regular, in that solutions in the knowledge base occur in patterns and relationships similar to those found in situations in which the solutions are likely to be applicable in the future. This approach is domain independent: it is applicable to a wide variety of tasks and problems from simple search applications to complex cognitive agents.

To exploit context-sensitive asynchronous memory, reasoners need certain properties. *Experience-based agency* is an agent architecture which provides an outline of how to construct complete intelligent agents which use a context-sensitive asynchronous memory to support a reasoning system performing a real task. The experience-based agent architecture combines a context-sensitive asynchronous memory retrieval process with a global store of experience used by all agent processes, a global working memory to provide a uniform way to collect feedback, and a global task controller which orchestrates reasoning and memory. The experience-based agent architecture also provides principles for constructing integration mechanisms that enable reasoning tasks to work with the context-sensitive asynchronous memory.

Furthermore, to help determine when these approaches should be used, this research also contributes theoretical analyses that predict the classes of tasks and situations in which the context-sensitive asynchronous memory and experience-based agent approaches will provide the greatest benefit.

To evaluate the approach, the experience-based agent architecture has been implemented in the Nicole system. Nicole is a large Common Lisp program providing global long-term and working memory stores represented as a rich, reified, grounded semantic network, a context-sensitive asynchronous memory process based on a novel model of context-directed spreading activation, a control system for orchestrating reasoning and memory, and a task language to implement reasoning tasks. Nicole enables the context-sensitive asynchronous memory approach to be applied to real

problems, including information retrieval in Nicole-IRIA, a information management application that uses context to recommend useful information (Francis et al. 2000), planning in Nicole-MPA, a case-based least-commitment planner that adapts multiple plans (Ram & Francis 1995) and language understanding in ISAAC (Moorman 1997), a story understanding system which uses Nicole’s retrieval system as part of its creative understanding process. Nicole and her children thus provide a testbed to evaluate the context-sensitive asynchronous memory approach.

Experiments with Nicole support the claims of the approach. Experiments with Nicole-IRIA demonstrate that a context-sensitive asynchronous memory can use feedback from browsing to improve the quality of memory retrieval, while experiments with Nicole-MPA demonstrate how information derived from reasoning can improve the quantity of retrieval. The use of Nicole’s memory in the ISAAC system demonstrates the generality of the context-sensitive asynchronous memory approach. Other experiments with Nicole-MPA demonstrate the importance of representation as a source of power for context-sensitive asynchronous memory, and further demonstrate that the core features of the experience-based agent architecture are crucial sources of power necessary to enable a reasoning task to work with and exploit a context-sensitive asynchronous memory.

In sum, these evaluations demonstrate that the context-sensitive asynchronous memory approach is a general approach to memory retrieval which can provide concrete benefits to real problems.

# PART I.

## FOUNDATIONS

---

*The problem, existing approaches to the problem, and why a new approach is needed.*

This dissertation investigates the challenges in memory retrieval faced by intelligent agents operating in complex, dynamic, resource poor environments. Cognitive agents need to use the limited information and resources available to them to find useful information from the large and complex knowledge bases they have built up through experience. The solution I propose is to make memory retrieval an active, autonomous process that can not only work in parallel with reasoning but also exploit ambient information available from the task and environment. Chapter 1: Introduction discusses this context-sensitive asynchronous memory approach, its major claims, and its major contributions.

The problem of memory retrieval is not new: it has received much attention in artificial intelligence, cognitive psychology, philosophy and general computer science. However, no existing approach fully meets all of the desiderata for a memory retrieval system for a general cognitive agent. Chapter 2: Related Work surveys prior research in memory retrieval, identifying the strengths of prior approaches, their limitations, and how context-sensitive asynchronous memory overcomes these limitations to provide a more comprehensive solution.

# CHAPTER I.

## INTRODUCTION

---

*Solving the problem of finding good answers to bad questions with limited resources*

Consider the following example:

A doctor treating a patient with a mysterious ailment turns to his Medical Advisor program for help. In the Advisor he creates a profile for the patient, and it automatically searches for case histories, journal articles, and Web sites with relevant information. The Advisor automatically attempts to diagnose the problem using an expert system, but because this class of ailment is rare the Advisor does not have rules that apply.

The initial search returns thousands of documents, most of which are irrelevant: information on other ailments the doctor has already dismissed, informational Web sites, and travelogues from a popular science magazine. One article catches the doctors eye, and when he clicks on it the Advisor automatically suggests dozens of other articles on the same ailment. Within a few moments of browsing, the doctor has narrowed his search from thousands of documents to hundreds. Many contain useful information, but not everything he needs; and there are still a few documents which don't seem to fit, like the travelogue.

His computer beeps with new mail: the patient's blood work is now complete. The doctor drags the attached file into the patient portfolio of the Advisor, and in response it automatically rearranges the documents that he had focused on. Now the travelogue is the highest rated item, and several keywords relevant to the disease are displayed next to its entry. Curious, the doctor clicks on the travelogue, then gasps. The article is by a traveling physician discussing a strange ailment he encountered on his travels, and the doctor immediately recognizes the tell-tale symptoms found in his own patient.

The key to the solution in hand, the doctor renews his search, instructing the Advisor to find more information on the rare ailment using his existing search and profile as its starting point.

In this scenario, the information necessary to solve the doctor's problem is available, but the doctor does not have the information necessary to ask the right question, nor does he have the time to exhaustively search the knowledge base. His medical advisory program serves two functions: it guides his search of existing information based on his needs and interests, and it automatically refines the search as new information arrives.

*Context-sensitive asynchronous memory* is a model of memory retrieval that makes this scenario — and many others — possible. The core problem that context-sensitive asynchronous memory addresses is how to get the information an agent needs when it

doesn't know how to ask the right question and doesn't have the time to exhaustively search all information available to it. The key to this solution is to interleave remembering with thinking and doing, thus making the context of thought and action available to guide remembering.

To fully exploit the benefits of context-sensitive asynchronous memory, reasoning tasks must satisfy certain constraints on how to interact with the memory system. *Experience-based agency* is a memory-focused agent architecture that provides an outline of how to construct complete intelligent systems based on context-sensitive asynchronous memory. Experience-based agency provides support for knowledge representation, memory/task communication, and memory/task parallel processing suitable for exploiting context-sensitive asynchronous memory, and provides guidelines on how to create task-specific integration mechanisms that accept asynchronous retrievals and integrate them into current processing.

To evaluate the context-sensitive asynchronous memory approach, its claims must be tested in an implemented system. *The Nicole system* is an agent architecture which consists of a context-sensitive asynchronous memory embedded within an experience-based agent construction toolkit. Two systems constructed using Nicole are the Nicole-IRIA information retrieval system, an information management application that uses context to recommend useful information, and the Nicole-MPA planning system, a case-based least-commitment planner that adapts multiple plans. Experiments with Nicole-IRIA and Nicole-MPA demonstrate that a context-sensitive asynchronous memory can

use contextual information and asynchronous retrieval to improve the quality of retrieval for end-user tasks and demonstrate the importance of the elements of the experience-based agent architecture for providing these capabilities. In addition to these experiments, Nicole's memory system was used in the construction of the ISAAC story understanding system (Moorman 1997) as further evidence of the generality of the context-sensitive asynchronous memory approach.

*In toto*, these evaluations show that the context-sensitive asynchronous memory approach is a general approach to memory retrieval which can provide concrete benefits to real problems. These evaluations further show that this research has successfully contributed a viable instantiation of the context-sensitive asynchronous memory approach which can exploit feedback to improve memory retrieval, along with an experience-based agent toolkit which provides mechanisms for interleaving memory retrieval and problem solving that are applicable to a wide variety of tasks.

To help determine when the context-sensitive asynchronous memory approach should be used, this research also contributes a theoretical analysis of the approach to help predict the classes of tasks and situations in which the approach will provide the greatest benefit. This analysis begins with an understanding of the problem that context-sensitive asynchronous memory is designed to solve.



## 1.1. The Problem

The context-sensitive asynchronous memory approach was motivated by problems in memory retrieval faced by general intelligent systems, such as the hypothetical doctor in our example. The solution context-sensitive asynchronous memory provides is a general one that can be applied to a wide variety of memory retrieval tasks. However, while it can be applied to almost any task, context-sensitive asynchronous memory provides the greatest benefit over competing approaches when applied to the kinds of challenging memory retrieval problems that arise in general intelligent systems performing multiple tasks under resource constraints — most particularly, the challenge of answering poorly-specified questions using large knowledge bases. Problems that share this profile present challenges to memory retrieval as well as opportunities for a solution; context-sensitive asynchronous memory exploits these opportunities to overcome these challenges.

The significance of these problems and the value of the context-sensitive asynchronous memory solution can be illustrated clearly by examining the nature of general cognitive agents, examining how cognitive agents pose challenges for memory retrieval, unpacking these challenges to understand why they are significant for a wider range of agents, and then turning to the properties of these problems and how they provide the keys to their own solution.

### 1.1.1. What are Cognitive Agents?

A cognitive agent is a general-purpose intelligent system operating within an external environment. By *agent*, I simply mean a system with inputs and outputs trying to achieve some task in an environment, along the lines of Norvig's (1995) definition. A *cognitive agent* is a particular class of agent: a system trying to achieve tasks within an environment by accumulating knowledge about the environment and acting according to that knowledge (Aquinas 1273). Cognitive agents may be contrasted with reactive agents, which possess a fixed amount of knowledge about their environment embedded within the design of processes which enable fast, successful performance.

Whether human or machine, the distinguishing feature of cognitive agents is that they contain within themselves knowledge of things outside themselves (Aquinas 1273, Newell 1990). This store of knowledge itself is alternatively called the agent's knowledge, its knowledge base, its long-term memory, or simply its *memory*. A perfect intelligence would apply all of this knowledge in its store in service of its goals at each point at which it was faced with a choice, but physically realized cognitive agents have limits on the knowledge they can process at any one moment.

This limit on how knowledge can be processed appears to be fundamental, arising in general philosophical arguments (again, Aquinas 1273), analyses of computational systems (e.g., Knuth 1973, Freeman 1975, Korth 1986, Newell 1990, Rosenbloom 1993), and studies of the characteristics of human memory (Miller 1956, Atkinson & Shiffrin

1968, Baddeley & Hitch 1974, Baddeley 1981, Anderson 1983, Newell 1990, Ericsson & Kintsch 1995). To overcome or cope with these limits, agents need a means of “distal access” (Newell 1990, Rosenbloom 1993): using the knowledge the agent is currently working with to get additional knowledge it needs from its knowledge base.

### **1.1.2. What is Memory Retrieval?**

Memory retrieval is a process that provides precisely this kind of distal access: it is a system for using the knowledge the system has “in mind” to get knowledge that the system has stored “in memory”. More precisely, memory retrieval is a technique for managing information when the amount of information to be managed far exceeds the ability to process or attend to that information. As we have just discussed, most tasks performed by intelligent agents require the use of more information than the agent has the capacity to attend to or process at any given time (Miller 1956). A memory retrieval system deals with this problem by separating the working memory an agent uses to perform its task from the long-term memory the agent uses to store its knowledge, and using information in the working memory to “bring” information from passive storage into active use (e.g., Atkinson & Shiffrin 1968, Baddeley & Hitch 1974, Baddeley 1976, Ericsson & Kintsch 1995; for a computational perspective, see Knuth 1973, Freeman 1975, Tannenbaum 1984).

A memory store cannot be completely separated from the retrieval process that operates over it; even at the definitional level, the concepts of memory and retrieval are

intimately intertwined. *Memory* can be defined as “the total of what one remembers,” and *remember* in turn as “to have (an event, thing, person, etc.) come to mind again; to bring back to mind by an effort” (Webster 1979); thus, memory is simply the sum of things which an agent has the power to bring to mind. This textbook definition presupposes not only a vast store of knowledge — a store far larger than an agent’s ability to attend to at once, and incapable of being processed unless it is being attended to — but a retrieval process: simply, the process of selecting a small portion of that knowledge store and “calling it to mind.”

Therefore, for the purposes of this dissertation, we will view memory retrieval as a metaphor for a certain kind of computational system which implies both representation and process. Assume a cognitive agent has an ephemeral, limited capacity store for processing information called its *working memory* ( $W$ ), and a permanent, essentially unlimited capacity store of knowledge called its *memory store* ( $M$ ). For convenience, we will assume the memory is divided into a set of *memory items* ( $m_{1...n}$ ). A memory retrieval system is a process  $f$  in a cognitive agent which accepts a *query* ( $Q_k$ ), or specification of needed knowledge, and selects<sup>1</sup> a set of items  $m_{i...j}$  in the memory store  $M$ ; these items constitute the *response* ( $R_k$ ) to the query, or “retrieval”.

---

<sup>1</sup> This definition specifies that memory *selects* a set of items, rather than *finds* or *returns* them, in an attempt to avoid needlessly narrowing the memory metaphor. Memory is often accused of implying a process that “goes elsewhere and returns with something” in the matter of a Labrador retriever, but this is not necessarily the case: a moment’s thought about the reality behind the popular Lycos commercial (in

To evaluate a memory system one may ask a number of reasonable questions, such as: What kind of knowledge can its store hold? What kinds of questions can be asked? How long does it take to get an answer? How good are the answers that are returned?

These questions can be broken down into two primary groups: what a memory system does, and how well it does it. “What a system does?” inquires about the functional capabilities of a memory system — for example, what kinds of information can it store (what kind of  $m_i$ s are allowed), and what conditions are necessary for the approach to bring information to mind (what kind of  $Q_k$ s can be processed)? “How well it does it?” asks about the performance characteristics of a memory system — for example, what costs does it incur (what are the performance characteristics of  $f$ ), and what is the quality of its output (what is the quality of  $R_k$  for a given  $Q_k$  according to some metric  $g$  for some task  $T$ )?

Answers to these questions are only meaningful in the context of the task that the agent performs. Given the requirements of a task performed by an agent, we can examine the functional capabilities and performance characteristics of a memory retrieval system and judge its suitability to serve that task.

---

which the faithful retriever “Lycos” is shown dashing off to get items as varied as scuba gear and supermodels) will show that retrieval can simply mean retrieving a pointer to an item rather than returning an item itself. For the purposes of this discussion, unless some functional difference exists the specific mechanism by which a retrieval is made is less important than the content of retrieval itself.

### 1.1.3. Examples of Memory Retrieval in Cognitive Agents

Each task performed by an intelligent agent places its own constraints on the memory retrieval process — on the type and amount of information that must be stored, on the speed of retrieval, and on the information available to inform retrieval. For example, the challenges of retrieving a phone number given a name are far different from those of recalling *Romeo and Juliet* given *West Side Story* (e.g., Schank 1982, p. 166).

What are the practical implications of focusing on problems relevant to general cognitive agents when developing an approach to memory retrieval? Consider the following examples:

- **Medical Advisor:**

A doctor treating a patient with a mysterious ailment turns to his Medical Advisor program for help. In the Advisor he creates a profile for the patient, and it automatically searches for case histories, journal articles, and Web sites with relevant information. The Advisor's initial search returns thousands of documents, most of which the doctor finds irrelevant. Relevance itself is a moving target: as the Advisor searches, the doctor continues to refine his query, and in addition the information available about the case itself continues to increase, in the form of new test results and examination findings. How can the Advisor quickly narrow down its presentation to information the doctor may find relevant, and how can it adapt as new information becomes available?

- **NASA Engineer:**

A NASA mission controller is informed of a serious and unanticipated problem in a spacecraft beginning its approach for a lunar landing. She has partial telemetry about the spacecraft, a wide variety of technical schematics, a set of contingency plans for anticipated problems, twenty years of experience, and three hours to find a solution before the mission computer executes an automatic abort. How can the controller find the relevant information out of her vast store of experience (and vast piles of data) quickly enough to solve this problem?

- **AI Reasoning Server:**

An AI reasoning server receives a problem from one of its customer sites. The server classifies the problem as a very hard, complex planning task, projected to take several hours of computer time — possibly more effort than it is permitted to expend on any one request. To save time, the server considers its large case-base, which contains solved problems in a wide variety of tasks and domains. However, no individual past case immediately solves the problem, and explicit match of a case to a problem is computationally expensive. How can the server quickly zero in on the cases in its case base most likely to solve the problem without expending effort that might better be expended on constructing a solution from scratch?

- **Research Scientist:**

A researcher working on a grant proposal due by the end of the month needs more

information about the application area he proposes to apply his research work to. A search of the web for documents on the application area proves fruitless. Thousands of potentially relevant pages exist, including publications by researchers working on similar problems or web pages on similar areas, but these are buried under hundreds of millions of web pages. How can the researcher (or his search engine) find relevant information quickly and efficiently?

While the details of their tasks differ, the challenges these agents face are similar.

#### **1.1.4. Unpacking The Problem**

Each of these situations features an agent (human, machine or human-machine combination) who must efficiently find answers to vague, poorly specified, or particularly challenging questions. Each is trying to solve a problem in some task or environmental problem solving context. Each has some information on hand, along with methods of access to a variety of information resources, including resources both inside and outside the agent's "head". And each of these agents is finite: the resources available to solve their problems are quite limited. Unpacking each of these properties in turn reveals many constraints upon memory systems for general cognitive agents.

**Agent Information Needs.** Each of these situations features an *agent*, performing a *task* in an *environment*, that encounters some *information need* — a self-contained system, doing something to some degree of success in a world larger than itself, that finds that the information it has is insufficient for it to keep doing what it has been doing.



- **Agent**

An agent is a self-contained system with inputs and outputs, attempting some transformation of input to output related to one or more tasks. The Medical Advisor, the NASA Engineer, the AI Reasoning Server, and the Research Scientist are all agents.

- **Task**

A task is a specification of a desired condition and measure of satisfaction — a specification of a desired state of the agent or its environment, coupled with some metric that gauges how well the agent is doing at achieving that state. For the NASA Engineer, the task is to save the crew, the spacecraft, and the mission, in that order. For the Reasoning Server, its task is to serve the reasoning requests it receives. For the Research Scientist, the task is to write a successful grant proposal.

- **Environment**

The environment is everything beyond the agent — an independent “entity,” beyond the agent’s direct volitional control, with its own state and dynamics. The agent and task are embedded within and dependent on, but logically distinct from, the environment. The NASA Engineer is painfully aware of the independence of her environment, which includes a recalcitrant spacecraft two hundred and thirty thousand miles away. For the Reasoning Server, the environment includes its clients and the history of problems it receives from them. For the Researcher, the

environment includes funding agencies, colleagues, a parent institution, local libraries and bookstores, and the World Wide Web.

- **Information Need**

Each of these agents needs some information, about the environment, the task or even itself, to successfully perform its task or to improve its performance. The NASA Engineer needs to know how to fix what is wrong with the spacecraft. The Reasoning Server needs the answer to the question it has been asked. The Research Scientist needs greater understanding of his chosen target domain.

Furthermore, an agent in a realistic environment may perform multiple tasks or a single task with many distinct subcomponents. As the agent's task structure becomes more rich, its information needs become correspondingly rich as well.

**Limitations.** Each of these agents is limited in the amount of information it can *process*, what it can *pay* for that information and how it can *access* that information. Each agent can attend to a limited amount of information, far less than the sum contained in the information resources available to it. Access to those resources incurs costs, in time, space or money, that the agent has a limited ability to pay. And the methods for information access available to the agent are similarly limited, this time by the design of the agent.

- **Attentional Limits:**

Agents have a limited ability to attend to information. This includes limits on fast

storage space for information “on hand” as well as process limits on actually manipulating that information. The NASA Engineer cannot keep all the schematics of a spacecraft in her head, nor can the Researcher mull over the full text of even one moderately sized paper in his head.

- **Resource Limits:**

Agents have a limited ability to get new information. Access to information resources has costs which agents have a limited ability to pay — either in processing resources, access time, energy or money. The AI Reasoning Server does not have infinite resources to apply to any one request, nor does the Researcher have the hours in his life to read all relevant literature even in a small subfield.

- **Access Limits:**

Agents also have limited ways to get new information. In addition to a finite number of onboard stores of knowledge, agents have access to a finite number of external stores. Obtaining access to new stores also encounters resource and attentional limits. Just as the Researcher cannot read every potentially relevant paper, nor can he travel to every library or even learn the interface to every information retrieval system available. His internal information access methods are similarly limited: he cannot download the contents of a digital library directly into his brain — at least not in time to write his grant proposal.

Attentional limits make it unlikely that an agent in a realistic environment will be able to meet its information need with the information it has on hand. Moreover, access limits mean that agents in complex environments must use the same set of information access methods to service the information needs of a variety of tasks that they perform.

**Task Demands.** The tasks these agents face have certain demands and constraints. Some tasks place resource constraints on retrieval; others need to satisfy, demanding a “good enough” answer; still others need to optimize, finding a “best answer”. Often, the same agent may need to constrain, satisfy and optimize all at the same time

- **Constrained Retrieval:**

Some tasks are resource-constrained and need to limit the effort that they expend on memory retrieval. This can take the form of competition between memory retrieval and task performance for resources, or the form of task demands for immediate action.

- **Competition:**

Information access may compete with the task for resources, or, if no explicit competition exists, excessive time or costs expended on information access may degrade the quality of the solution or performance on the task. The more time the NASA Engineer spends hunting through specifications and manuals, the less time she has to actually devise a solution to the problem. As another example, an agent driving across the country can afford to glance at a map for

a moment, but extensive search of its information resources during task performance would likely lead to extensive damages.

- **Task Demands:**

Requirements of the task may demand that performing the task continue regardless of the status of memory retrieval. Alternatively, the task may simply benefit from continued performance, regardless of the speed or effectiveness of information access. Real-time systems are a good example. For a real-time system the speed of memory is irrelevant: whether or not a retrieval occurs, certain actions must be performed on a fixed schedule and memory retrieval must work around it. The driving agent in the previous example does not need information rapidly; it can stop at any time and review its information resources, most likely without degrading its overall task performance. However, if it is able to continue driving it will reach its destination sooner, and while it is driving, the demands of the driving task force map access to be limited to an occasional quick glance.

- **Satisficing Retrieval:**

Other tasks must satisfice, settling for something “good enough” to allow them to move forward. This can take the form of needing the quickest response possible, the first good response, or simply any response of any kind whatsoever.

- **Fast Guess:**

For example, the constraints of the task may demand a fast answer rather than

waiting an arbitrary amount of time for memory to retrieve the best result. Real-time systems are again an example of systems that may need a retrieval as quickly as possible. The NASA Engineer has a similar problem: if the problem is pressing enough she may need to try *some* solution before it is too late to save the crew.

- **First Guess:**

Other tasks may simply not need the best answer and can settle for any answer in the ballpark. For these systems, a variety of different answers will be “good enough” to allow reasoning to proceed and waiting for a best answer will provide no additional benefit. If an initial literature survey yields some relevant results, the Research Scientist can use it to flesh out his bibliography and move on to presenting the novel technical details of his proposal.

- **Shoot First:**

Still other tasks may simply require some feedback — *any* feedback — from memory to contribute to the reasoning process, helping the agent move forward. These “shoot first” systems (Riesbeck 1993) may ultimately need the best answer, but can use a best guess to get reasoning started while memory continues to churn. The initial memory retrieval can help the reasoner refine both the reasoning state and the memory query. Simply finding some results in a related field can get our Research Scientist started on tackling the problem, even if those references will not be ultimately sufficient.

- **Optimizing Retrieval:**

Finally, systems may need to optimize the results that they get. This can take the form of demanding an optimal result; alternatively, it can take the form of matching results to a continually refined information need or continuing to find results for an outstanding information need.

- **Best Answer**

Some tasks may demand the absolute best item found in the knowledge base. Our NASA Mission Controller may need the absolute best information available if only she can find it in time. If our Research Scientist is completing a journal article or survey paper, he may need to scour the literature for the most on-point prior references.

- **Refined Information Need**

However, the NASA Engineer's understanding of the problem, like the researcher's, develops over time as more information is collected about the problem. Initial sets of diagrams or specifications may not be relevant as the problem statement is refined. Retrieval in these circumstances needs to continue to optimize what is retrieved to match the changing information need. Again, if our Research Scientist can find simply some results in the related field, following up on those results may help him refine his search and focus it more appropriately to the terms and authors relevant to that field.

- **Outstanding Information Needs**

Unlike the NASA engineer, the researcher's need for information may continue over a long period of time, long after the immediate problem of the grant proposal is resolved. The Researcher's memory access methods, both internal and external, should continue to seek additional information relevant to the information need as long as the researcher continues to work on the problem.

For satisficing and optimizing tasks, it may be productive to allow task performance to proceed in parallel with memory search, allowing memory to search incrementally, returning results as it finds them. This kind of memory must be parallelizable or interleavable with reasoning, and must manage the amount of effort it expends on retrieval, exploiting additional resources over time as they become available. Resource constrained tasks could similarly benefit from a memory that can incrementally consume resources to attempt to improve its answer, yet stop at any arbitrary point to return the best answer found so far.

In short, these agents need both quick answers to questions and continued processing of outstanding information needs.

**Large Bodies of Knowledge.** Because of their limits and the constraints of their tasks, these agents need to build large, complex bodies of knowledge. Each of the agents



listed above needs a large body of knowledge to perform its task for closely related sets of reasons.

- **Task Complexity:**

A task may simply be complex, requiring access to a large body of knowledge for successful performance. A summary of the information needed to construct and maintain an Apollo-era manned spacecraft runs over 500 pages (Purser et al. 1964) and thousands upon thousands of pages of information would be readily available to the NASA Mission Controller trying to debug a malfunctioning spacecraft. The human body is even more complex: the Physician's Desk Reference (Barnhart 1991), which only describes available *medications*, nevertheless runs over 2500 pages, and hundreds of thousands of more pages of information about physiology, pathology and treatment are available online to the Medical Advisor (e.g., National Library of Medicine 2000) *not counting information available on the World Wide Web*.

- **Task Constraints:**

A task itself may simply require a large body of knowledge. For example, the task of finding a phone number given a name is fairly simple, yet because of the vast number of persons with phones in any given geographic region a knowledge base that can answer that needs must be vast. Similarly, finding pages on the Web is not a complex task, but there are thousands of millions of accessible pages (Lawrence & Giles 1998, 1999, Broder et al 2000) making indexing and

representing those pages a challenge (Ray et al. 1998, Baeza-Yates & Ribeiro-Neto 1999).

- **Multiple Tasks:**

An agent may perform multiple tasks at the same time and may thus require knowledge bases which contain a variety of information. One researcher may perform tasks as diverse as grant research, proposal writing, reviewing and refereeing, teaching, and advising. The NASA Mission Controller's problem was earlier described as one task, but it could just as easily be described as dozens of different tasks, each presenting its own challenges and its own goals; furthermore, the goals of these tasks may be aligned with each other or may conflict. As another example, a hunter living off the land must perform many tasks — finding food, finding water, navigating between traps and homestead, and avoiding the odd tiger.

- **Ambiguous or Unknown tasks:**

Now, multiple constraints begin to interact: an agent performing multiple tasks yet having limited information access methods may be forced to use those methods in uncertain or ambiguous circumstances. In theory an agent that performs multiple tasks could employ several different sources of knowledge; however, an agent may not always be able to completely partition its work. When an agent initially encounters a challenge in its environment it may not be clear exactly what task the agent needs to perform. When an agent asks for information

in these circumstances its information access methods must be ready to supply information related to any one of the agent's tasks at any time. For example, the Medical Advisor cannot be restricted to searching for just physiology, diseases or treatments in isolation; similarly, the Researcher will not tolerate a search engine that could only access one field of information when another might contain the information that is relevant.

In any of these circumstances, an agent needs an information access method which can support a large knowledge base; moreover, the agent may need use the same method to access heterogeneous bodies of knowledge. These heterogeneous bodies of knowledge may arise within large complex tasks or may arise because the agent performs multiple tasks.

In short, agents solving realistic problems in realistic environments encounter many qualitatively different kinds of information needs which can only be satisfied by accessing large bodies of knowledge.

**Limited Question-Asking Capabilities.** Each agent's ability to ask questions of these bodies of knowledge is limited. When an agent encounters a need for information in problem solving that cannot be met by the information on hand, it must use that information to invoke the methods it has to search the information resources available to it to try to get the needed information. There is no guarantee that information may enable the agent to precisely specify the needed information: the information need may be

ambiguous, may be incorrect, may need refinement, or may simply be a poor match to the content of the available knowledge.

- **Ambiguous Questions:**

Agents may not have enough information on hand to precisely specify the information they need. For example, the Researcher may perform a search on “neural network”, but this term is ambiguous; a brief survey of (Arbib 1995, Rumelhart & McClelland 1986) yields dozens of different specializations of this term, including “cortical neural network” (which describes neural circuits in the brain), “Hebbian neural network” (which describes a particular way of modeling the brain), and “backpropagation neural network” (which describes a machine learning technique inspired by, but often used in ways completely unrelated to, brain studies).

- **Incorrect Questions:**

The agent’s questions may be worse than ambiguous: they may be poorly posed or incorrect. For example, the NASA Mission Controller could assume that two malfunctioning systems may be connected and may embark on a fruitless search for the connection paths; the Researcher’s understanding of a field may cause him to formulate an incoherent question; a doctor may place undue weight on irrelevant symptoms, leading his Medical Advisor astray.

- **Refined Questions:**

As the agent continues to perform its task, it may refine its view of the questions.

The NASA Mission Controller's systematic tests of various possibilities narrows the search space; the Researcher's readings of papers improves his understanding of the subject matter; the doctor accumulates more information and refines his hypotheses about the disease. In all of these circumstances, the initial search queries are likely to be inadequate and need serious refinement.

- **Mismatched Information:**

There may be a mismatch between the information on hand and the content of the knowledge base. To return to the Research Scientist, an information need specified as "neural network" might, for a given document collection, be more accurately formulated as "self-organizing Kohonen map." Both descriptions are true and accurate ways to describe a certain machine learning technique, but asked over the wrong document collection the query "neural network" would return no relevant results. As another example, there is a uniform standard for medical terms that the doctor could use (National Library of Medicine 2000) but in practice physicians do not use this uniform vocabulary to formulate queries (Pottenger, personal communication, 2000).

In short, agents have limited abilities to specify their own information needs, either in general or with respect to given knowledge bases. The result of this limit is that the initial results that the agent receives may not be adequate for satisfying the agent's information needs. These tasks could benefit from a memory which could build upon its initial search effort, harvesting additional results as the search terms are refined.

**The Problem.** To sum up, all of these agents have the same problem:

Agents with limited resources need an information access method that can efficiently retrieve quick but refinable answers from large knowledge bases given poor questions.

### **1.1.5. Why the Problem is Hard**

This problem presents a number of challenges for devising an appropriate memory retrieval system. These include challenges in representing knowledge for the variety of tasks an agent might perform, in providing access to that knowledge, in making that access efficient and useful, in coping with changing information needs, and integrating memory retrieval with the demands of task performance.

- **Challenges of Knowledge Representation:**

Representing knowledge for complex tasks requires developing knowledge stores with rich representational power. For example medical diagnosis may require large and complex semantic network representations (Santos et al. 2000, National Library of Medicine 2000) whereas representing electronic circuits such as might be found in a spacecraft requires complex model structures (e.g., Bhatta & Goel 1993, 1994, Peterson et al. 1994, Goel et al. 1996). Incorporating multiple kinds of knowledge into the same framework present their own challenges in developing knowledge representations which are sufficiently powerful (Lenat & Guha 1990, CYCORP 2000, Clark & Porter 1996, 1997).

- **Challenges of Knowledge Access:**

Beyond representing knowledge lies the challenge of accessing it. If a single access method is a gateway to a store of knowledge that contains complex, heterogeneous bodies of knowledge, that access method must be powerful enough to provide access to the right knowledge, yet flexible enough to apply across the many domains of knowledge in the store. Tradeoffs may need to be made between generality of access methods and task-specific matching criteria.

- **Challenges of Efficiency:**

Providing efficient access to knowledge is also a challenge, one which becomes increasingly important as knowledge bases grow in size and as resource constraints on memory retrieval increase. In general efficiency of representation and generality of knowledge are tackled as separate research challenges; for example, database systems have impoverished representations but are optimized for speed whereas rich knowledge representations do not address the speed issue. A memory access system for general cognitive agents must address both challenges simultaneously.

- **Challenges of Quality:**

As already mentioned, one way to address the issue of efficiency is to provide satisficing answers: first guesses, best guesses, initial answers of some degree. Once the door has been opened to this possibility, however, a new challenge opens: ensuring that fast satisficing answers are good enough answers for the

agent to use. Information foraging theory (Pirolli & Card 2000) suggests that if an answer is below a certain quality threshold it is not worthwhile for an agent to spend any time considering it regardless of the amount of resources available; the challenge of quality is to ensure satisficing answers are good enough to be worth considering.

- **Challenges of Refinement:**

Another challenge arises in supporting refinable memory retrieval queries. Any of the solutions which enable query results to be refined — whether an anytime solution that allows the memory to continue to work on a problem incrementally or a feedback-based solution that enables an agent to refine a query — must provide a language for actually performing that refinement. This includes the ability to specify queries, to update them, and to examine their status (so that refinements are meaningful).

- **Challenges of Integration:**

Finally, a challenge arises in any approach which calls for a memory to act independently from but working with a reasoning task. Many of the solutions hinted at above engage memory and reasoning in a kind of dialogue — the dialogue of query refinement, the dialogue of satisficing results, the dialogue of anytime incremental retrieval. Any of these solutions must address not only the language by which memory and reasoning communicate, but also how the memory process and reasoning process can work in parallel or interleaved with



each other, as well as how the reasoning task should interact with the memory for the best results.

### **1.1.6. Desiderata for a Solution**

Because of these challenges, as well as the properties of the problem itself, an agent facing this problem has stringent demands on its methods for accessing information.

A cognitive agent needs a method (or methods) for information access which applies across the set of tasks it performs. This memory system should be effective enough at retrieval to provide benefit, yet cost-efficient enough not to undermine task performance. It should not consume an excess of resources, ideally working in parallel with reasoning and action and responding if reasoning demands an answer. And ideally, it should exploit any information or extra resources available to provide the best performance.

Because a single agent may have different needs for information at different times — my needs for information when running from a tiger are far different from my needs when completing a survey of related work — a memory system for an intelligent agent may be required to attempt to optimize different performance criteria at different times: optimizing speed to deal with the tiger, optimizing recall to flesh out the section on related work, optimizing precision when answering an on-your-feet question.

Therefore, an information access method serving cognitive agents like those in our examples must satisfy a variety of demands simultaneously: it must store and manage

large amounts of information relating to a variety of tasks and access that information accurately and efficiently in a way appropriate to the situation at hand without consuming excessive resources. Ideally, such a memory could work in parallel with action or reasoning, amortizing the cost of search over a period of time or returning a best guess to allow action to continue. Teased apart into explicit desiderata, these requirements for a memory system for a cognitive agents include:

- **Desideratum 1: Stores a wide variety of information**

A memory system for a cognitive agent should be capable of storing many different kinds of information relevant to performing a wide variety of tasks.

- **Desideratum 2: Provides a general method for accessing that information**

A memory system for a cognitive agent should provide a uniform means of access to all the information it represents, regardless of its content.

- **Desideratum 3: Scales to large multifunction knowledge bases**

The performance characteristics of a memory system for a cognitive agent should be robust, capable of handling large heterogeneous bodies of knowledge.

- **Desideratum 4: Manages cost of retrieval**

A memory system for a cognitive agent should control retrieval cost, enabling an agent in a dynamic world with limited resources to efficiently exploit its past knowledge.

- **Desideratum 5: Preserves accuracy of retrieval (in the face of resource limits)**

While it controls costs, a memory system for a cognitive agent should also ensure that quality of retrieval is as high as possible.

- **Desideratum 6: Exploits task/environmental information**

If additional information is available in the environment or task beyond a particular query, a memory system for a cognitive agent should exploit that to improve retrieval if possible.

- **Desideratum 7: Exploits extra resources if available**

A memory system for a cognitive agent should not merely control retrieval cost; if additional resources are available it should exploit them to improve retrieval.

- **Desideratum 8: Potentially interleavable with reasoning**

A memory system for a cognitive agent should work in parallel or interleaved with reasoning tasks, enabling the agent to continue to respond to its dynamic environment.

- **Desideratum 9: Provides guidelines for reasoning integration**

The design of a memory system for a cognitive agent with the above processes should specify how reasoning tasks should be constructed to optimally work with the memory.

### **1.1.7. Possibilities Raised By Interacting Constraints**

A variety of scenarios discussed earlier — memories that proceed in parallel with reasoning, memories that incrementally consume resources, memories that return anytime

answers, memories which permit updated retrieval specifications — open interesting possibilities.

If memory and reasoning are working in parallel to exploit asynchrony, or if reasoning is waiting on memory in preparation for demanding an anytime retrieval, then information may be generated, either by reasoning about the task or by events in the environment. If the reasoner believes this information is directly relevant to the question it asked, it could potentially update the specifications of the query — if memory provided an interface to do so.

Unfortunately, the reasoner may not know enough to ask questions correctly; after all, the contents of memory are hidden from the reasoner until they are retrieved. However, what is hidden to the reasoner is visible to the memory; in theory, the more information a memory system has about the context of a question the better it can focus its effort on stored items likely to be relevant. Fully exploiting information generated by the task or environment requires a memory that can do so directly without the need to wait for the reasoner to update its specifications.

### **1.1.8. Keys to A Solution**

It would be amazing if a single information access method could satisfy all of these desiderata simultaneously. But one memory system can do all of those things — the memory system inside the human head. If we are to have a prayer of a chance of developing a full-blown artificial cognitive agent, we must further the state of the art of

memory systems and develop an artificial memory retrieval system which can meet the needs of a cognitive agent by looking to humans as our example.

Humans, our best example of general intelligent systems, solve the problem of efficiently finding useful answers to under-specified questions seemingly without effort. Psychological research has shown that to do so, human memory obeys a number of general principles which are useful for acquiring, organizing and retrieving information (e.g., Baddeley 1976, Anderson 1981), such as the power law of learning and chunking of concepts (e.g., Miller 1956, Newell & Rosenbloom 1981, Newell 1983). Most important for this research is the phenomenon of *priming*: the ability to use information in the environment to highlight related information in memory, making it faster to retrieve and more available for reminding (e.g., Collins & Loftus 1975, Anderson 1983, McNamara 1992, Klimesch 1994, McNamara & Diwadkar 1996).

This human capability points a possible way to tackle the problem of efficiently finding answers to vague questions from a large knowledge base: use a priming-based approach that exploits information beyond the questions asked of a memory system to help the memory more quickly and accurately focus attention on potentially useful answers.

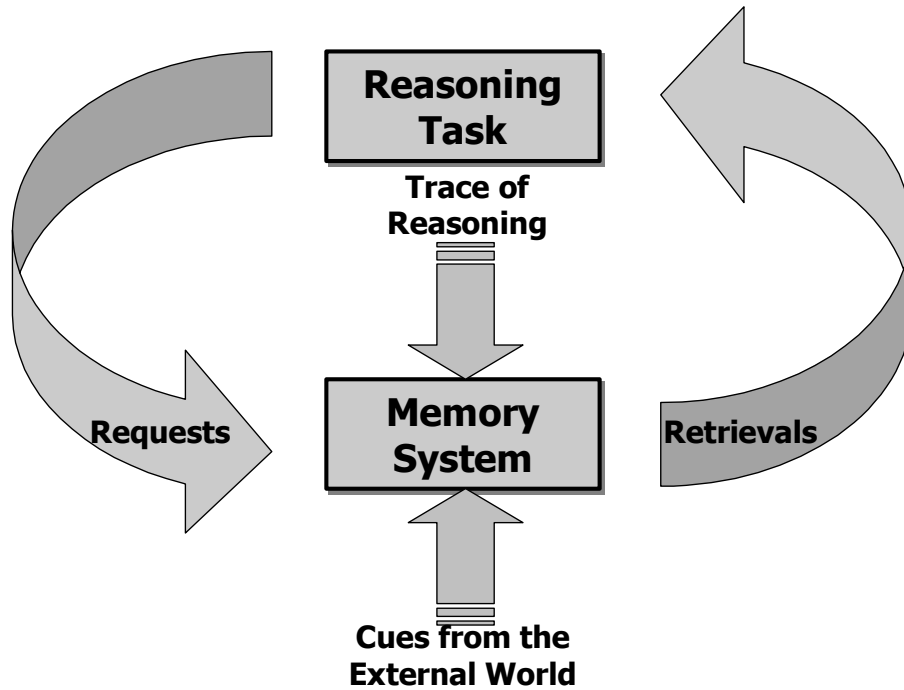
Where might this “information beyond the questions” come from? It may come from outside the agent: for example, when an agent’s environment changes, it may generate additional information which could inform memory retrieval. Work on the task or

problem itself might also generate such cues: for example, by further exploration of the environment (e.g., Kolodner & Wills 1993a, 1993b, Wills & Kolodner 1994, Simina & Kolodner 1995). It may also come from within the agent: as work on the task or problem progresses, new needs for information may be generated whose information access “footprint” may overlap work done answering earlier questions: for example, information generated through situation assessment (Kolodner 1993) or elaborated processing of textual data (Lange & Wharton 1994) can elaborate and build on initial retrieval efforts.

The solution described in this thesis embodies exactly this kind of approach, using human-inspired priming technology to exploit additional information to answer questions that could not otherwise be answered.

## **1.2. The Solution**

Context-sensitive asynchronous memory is a priming-based approach to memory retrieval. It exploits feedback from the task and environment to guide and constrain memory search by interleaving memory retrieval and problem solving (Figure 1.1). Solutions based on context-sensitive asynchronous memory provide useful answers to vague questions efficiently, based on information naturally available during the performance of a task.



**Figure 1.1. Context-sensitive asynchronous memory**

### **1.2.1. Distinguishing Features of Context-Sensitive Asynchronous Memory**

The distinctive properties of context-sensitive asynchronous memory can be illustrated by comparing it with “traditional” memory retrieval approaches, which can be generally characterized as context-free, synchronous approaches to memory retrieval.<sup>2</sup>

All memory retrieval approaches begin by trying to improve upon generative problem solving in some way. An agent without a long-term memory must solve problems using

---

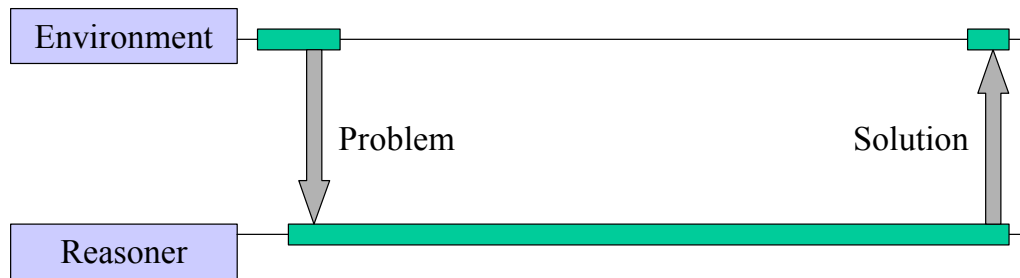
<sup>2</sup> Notable exceptions include models of priming from psychological research, such as (Anderson 1983) and (Klimesch 1994), which can be viewed as context-sensitive systems; also, while Anderson’s (1983) proposal is not a model of asynchronous retrieval in the sense proposed here, it does include a feature whereby more active productions are matched faster.

only first principles and the information available in its working memory store (Figure 1.2). Adding a memory retrieval system to an agent enables it to improve upon this “generative” approach by asking questions of its memory and getting useful answers that improve the quality or speed of the solutions it produces (Figure 1.3).

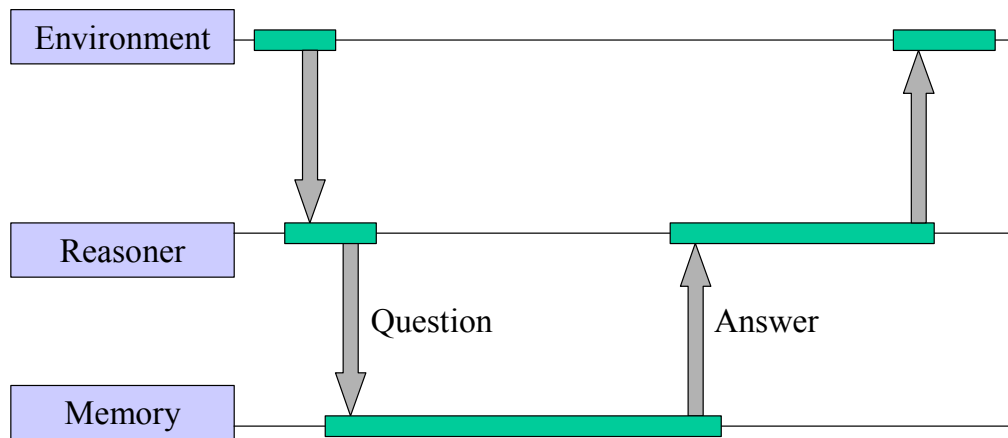
Synchronous memory retrieval can be defined as a memory retrieval approach in which reasoning waits for memory retrieval to return an answer. This describes the relationship between most traditional reasoning systems and their memory retrieval algorithms; memory retrieval is a subroutine call which effectively seizes control of the processor and suspends the reasoner until retrieval is complete (illustrated by the dormancy of reasoning in Figure 1.3).

Context-free memory retrieval can be defined as a memory retrieval approach in which memory retrieval is guided entirely by the information contained in the initial memory retrieval request. This also describes the relationship between most traditional reasoning systems and their memory retrieval algorithms; memory retrieval algorithms are designed to efficiently process the questions they are asked without providing a means to update those questions or guide the search while search is ongoing (illustrated by the limited communication between reasoning and memory in Figure 1.3).



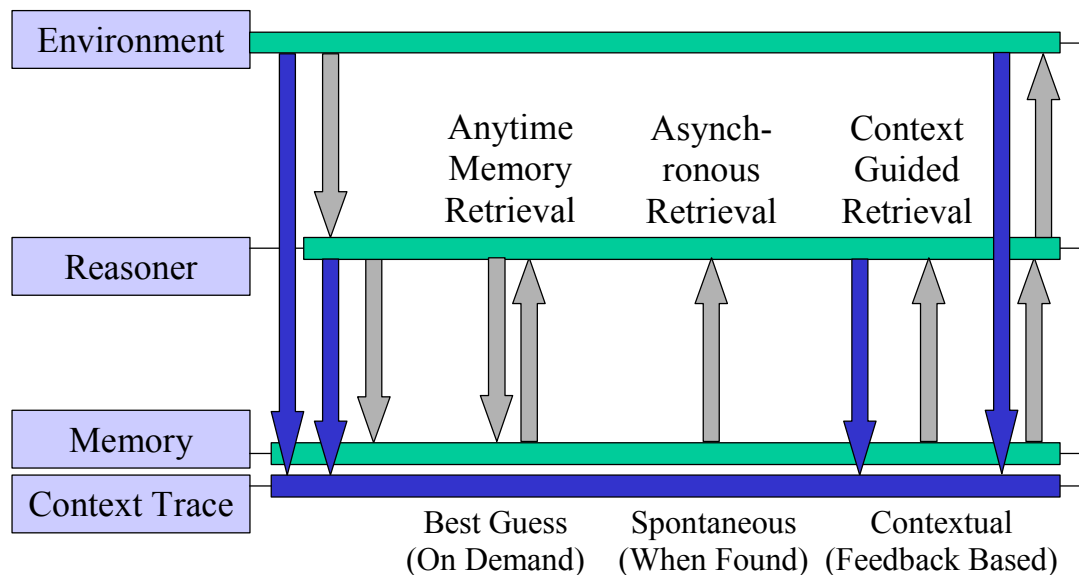


**Figure 1.2 Generative Problem Solving**



**Figure 1.3 Memory-Based Problem Solving**

A context-sensitive asynchronous memory differs from these “traditional” approaches by simultaneously providing a means to allow reasoning to proceed in parallel with memory search as well as a means to guide ongoing memory search. This enables a variety of new ways to retrieve information: *anytime retrieval*, in which the reasoning task demands that memory return its “best guess” at a retrieval so far; *asynchronous* or *spontaneous retrieval*, in which the reasoner waits for the memory to spontaneously return information, and *context-guided retrieval*, in which feedback from the memory system guides both spontaneous and on-demand retrieval by the memory system. Figure



**Figure 1.4 Context-Sensitive Asynchronous Memory Retrieval**

1.4. illustrates how anytime, spontaneous and context-guided retrieval together comprise context-sensitive asynchronous memory retrieval.

Asynchrony and context sensitivity depend on each other in this approach. Asynchrony is required for context sensitivity because unless there is a means to process memory retrieval requests independently of ongoing reasoning or action, there is no source of information to guide a context-sensitive search process; context sensitivity in turn augments asynchrony by providing an efficient way to incrementally search a knowledge base. Understanding how these components work together requires a closer look at the parts of a context-sensitive asynchronous memory system and how those parts fit together.

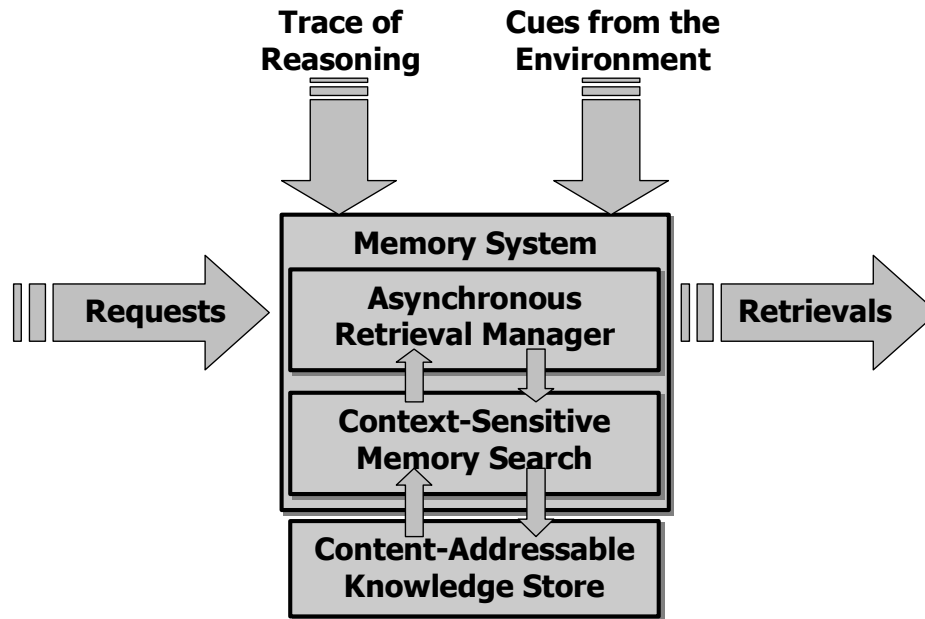
## 1.2.2. The Context-Sensitive Asynchronous Memory Approach

To work with reasoning tasks and exploit feedback from them, a context-sensitive asynchronous memory employs several parts that work together (Figure 1.5). These components include an asynchronous retrieval manager with anytime retrieval capability, a context-sensitive search process, and a content-addressable knowledge store:

### 1.2.2.1. Asynchronous Retrieval Manager

When a reasoning task asks a context-sensitive asynchronous memory a question, the memory system's *asynchronous retrieval manager* actively tries to answer the question while the reasoning task potentially continues to work (Figure 1.6). The retrieval manager continuously matches the retrieval specifications against the results of an ongoing memory search process until it finds a suitable answer, which it then passes back to the asking task. This novel spontaneous or “asynchronous retrieval” approach, so called because the asking of questions is no longer synchronized with the immediate return of answers, amortizes the cost of memory retrieval across several actions or reasoning steps to reduce the impact of the cost of search.

The key to asynchronous memory retrieval is a novel data structure called a *reified retrieval request queue*. A reified retrieval request queue is a buffer of declarative representations of requests for information — the “reified” retrieval requests — which the memory maintains and constantly attempts to satisfy through incremental search of memory, independent from the activity of other tasks. In theory an asynchronous memory manager can work with any incremental search process — including linear



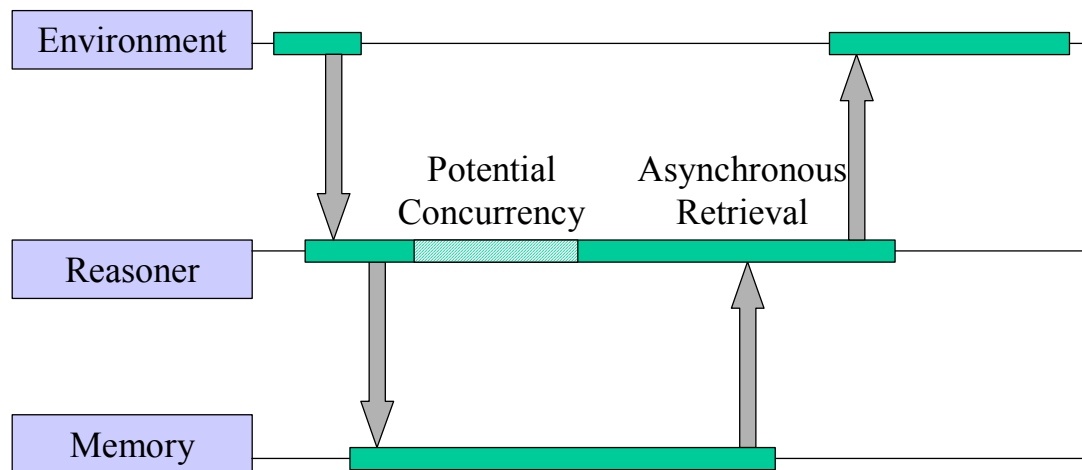
**Figure 1.5. Components of a context-sensitive asynchronous memory**

search, random search (the British Museum algorithm), or stochastic sampling (the shotgun algorithm) — but it was designed for and enables the context-sensitive approach described later in this chapter.

Asynchronous memory retrieval helps an agent cope with limited search resources or large knowledge bases: it allows memory to search for the best item in a large knowledge base as comprehensively as possible without forcing other tasks in the system to block. Asynchronous memory search also enables the memory to take advantage of additional information (explicitly provided or gleaned from the context) to improve retrieval.

#### **1.2.2.2. Anytime Retrieval Capability**

Explicitly representing requests for information enables a second capability: *anytime retrieval*. If a reasoner cannot wait on the asynchronous retrieval manager to autonomously provide a retrieval, it may demand — at “any time” — the best result that the retrieval manager has found so far (Figure 1.7). This enables reasoning tasks to make



**Figure 1.6 Asynchronous Memory Search**

an explicit tradeoff between a quick satisficing answer and a slow optimal answer based on a comprehensive search of memory.

The key to anytime retrieval is explicitly representing requests for information along with their satisfaction states. These “reified requests” enable a memory to return a “best guess” item based on a weighted match of the items it has examined so far. If a retrieval is requested and a suitable item has been found it should be returned in constant time; alternatively an anytime matching system can retrieve the *first* suitable item it finds in a “shoot first” fashion while the asynchronous memory continues to search for a better answer, as called for by Riesbeck (1993).

Anytime memory is crucial for some tasks. It is necessary for processes that need an initial seed of knowledge to begin reasoning and is useful to processes that have algorithms to refine a query based on an initial retrieval. Anytime matching is particularly useful when a knowledge base is large and contains many distractors

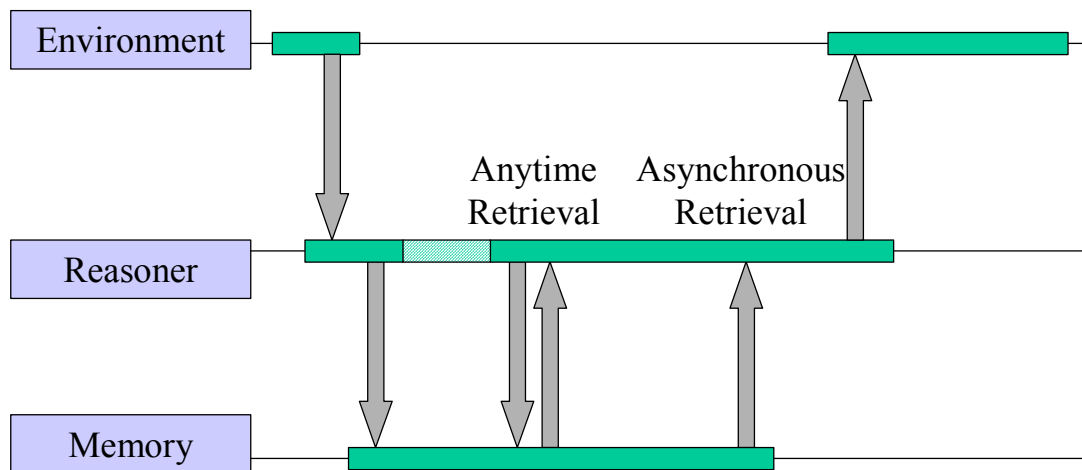
(irrelevant or weakly relevant pieces of information) which would be difficult to exhaustively match and rank.

### **1.2.2.3. Context-sensitive Search**

In a system in which memory and reasoning (or action) perform active work in parallel, context-sensitive search of memory becomes possible. This novel approach to memory search is based on a context-sensitive search process that guides its ongoing search of a knowledge base using information from the task, from the environment, and from any outstanding requests for information (Figure 1.8).

The keys to context-sensitive memory are a novel spreading activation algorithm called *context-directed spreading activation* which dynamically changes the weights of connections in the knowledge base based on the reasoning context, the asynchronous retrieval system which helps provide that context, and an knowledge store structured such that the connections between knowledge define a set of weighted links which the spreading activation algorithm can usefully adjust.

Asynchronous retrieval manages the ongoing context-sensitive search process, using it to search the store of knowledge incrementally while other reasoning tasks proceed, thus enabling search to proceed hand in hand with reasoning and environmental events in the hope that information generated from memory or reasoning could further guide the search and thus result in a successful retrieval.



**Figure 1.7 Anytime Memory Retrieval**

Context-sensitive search can improve the speed and precision of search by focusing the memory system's asynchronous search on the most relevant portion of the knowledge base. This also enables a single memory system to efficiently search a knowledge base used for several tasks, focusing the system's search dynamically on the portions of the knowledge base relevant to the task at hand.

#### **1.2.2.4. Content-Addressable Knowledge Store**

Finally, in order for a context-sensitive search process to function, the knowledge base that it operates over must be represented in such a way that the search process can use information provided as part of the context to find information likely to be relevant to parts of the problem. This requires a content-addressable knowledge base, in which elements in the knowledge base can be located via their semantic connections to other pieces of knowledge. As illustrated in Figure 1.4, the content addressable store is potentially a distinct component from the context-sensitive asynchronous memory retrieval system itself.

The key to a content-addressable store is a rich knowledge representation called a grounded bidirectional semantic network with reified relations; this representation both maintains links between pieces of knowledge based on their semantic relationships and provides sufficient information about those links to enable a search algorithm to dynamically adjust their weights based on the current context.

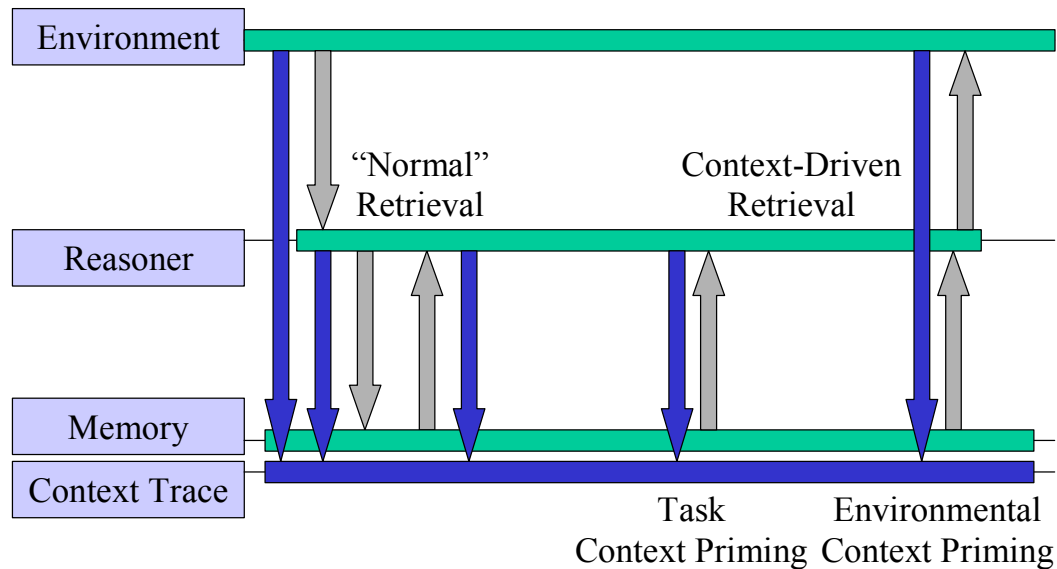
The rich structure of a content-addressable store enables the representation of knowledge about a wide variety of tasks within the same knowledge base, and supports the ability of memory search to focusing search dynamically on the portions of the knowledge base relevant to the task at hand.

#### **1.2.2.5. How these Components Work Together**

Together, context-sensitive search and asynchronous memory retrieval enable a memory system to expend as little effort on memory retrieval is required, exploiting information dynamically as it arrives to constantly tune the search to target relevant information.

Vague questions which are initially difficult to answer can be answered efficiently by using context to focus search, ensuring that only the most relevant parts of the knowledge are traversed and only the cream of the crop of possible retrievals are ever brought to the attention of reasoning. And when new knowledge is generated as a result of reasoning, it is added to the content-addressable store, which updates its internal connections to enable that knowledge to be efficiently traversed and retrieved in the future.





**Figure 1.8 Context-Sensitive Retrieval**

### 1.3. Applying The Solution

Unfortunately a context-sensitive asynchronous memory cannot simply be added to an existing reasoner without modification. By design, most existing reasoners ask a question of memory, wait for an answer, and then forget about the question; this approach does not provide a context-sensitive memory any priming information, does not allow for the possibility that a retrieval would be delayed, and does not permit reasoning to continue while memory search is ongoing — all key features of context-sensitive asynchronous memory. To take advantage of context-sensitive asynchronous memory, reasoners must be able to function in parallel with an active memory retrieval process, provide it implicit or explicit feedback, accept new asynchronously returned retrievals as they arrive, evaluate their quality, and integrate them into their current processing.

Each of these capabilities can be built into a reasoner by hand when it is constructed. However, some of these capabilities — such as providing feedback and working in parallel — are more or less standard across sets of tasks. *Experience-based agency* is a general architecture for intelligent systems that provides a uniform way to handle these problems. An experience-based agent is built around a context-sensitive asynchronous memory and a global store of knowledge, using a global task communication system and a parallel task controller to support the task-independent parts of exploiting a context-sensitive asynchronous memory (Figure 1.9). Experience-based agency also provides an outline of how to structure reasoning tasks to operate within this framework by incorporating components such as request generators, retrieval handlers, and integration mechanisms.

Experience-based agency thus provides a framework for the construction of complete intelligent systems built on context-sensitive asynchronous memory principles. The key principles of the experience-based agent approach are:

- **memory retrieval is asynchronous:**

memory retrieval operates in parallel with other processes within an agent, searching the knowledge store continually and incrementally, enabling it to respond to retrieval requests either in an anytime fashion, returning the “best guess” found so far, or in an spontaneous or asynchronous fashion, retrieving answers as they are found.

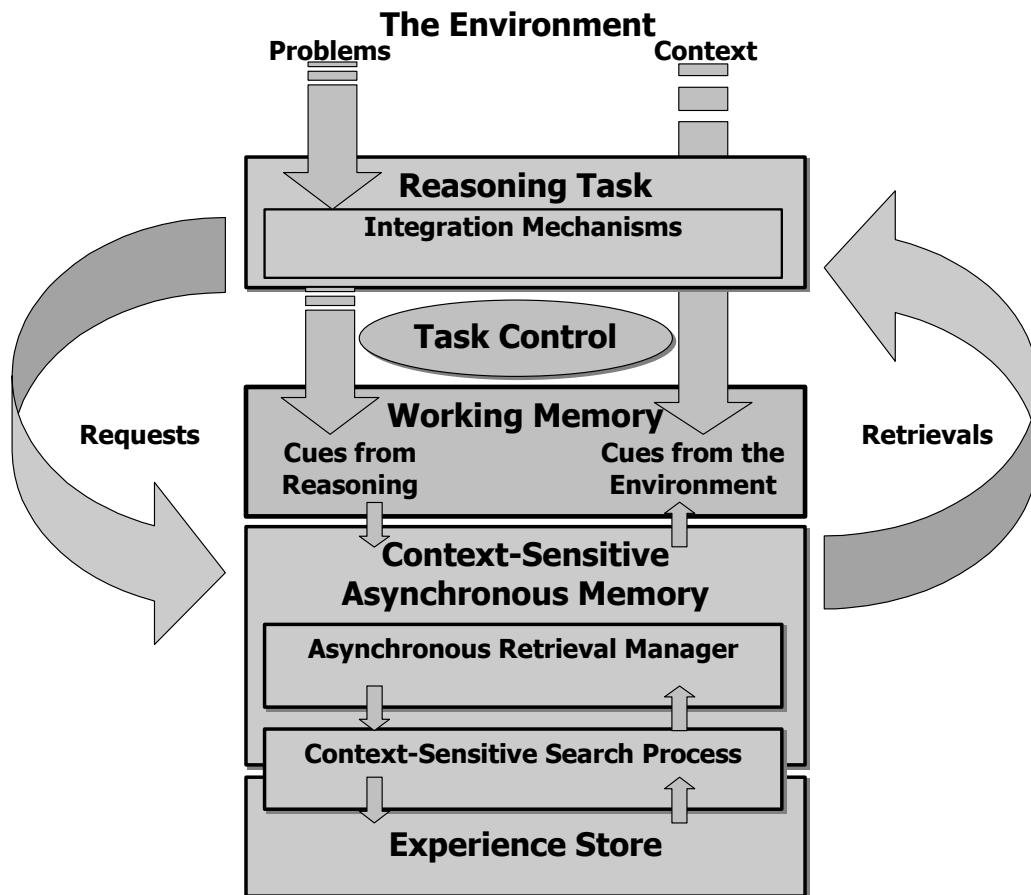


Figure 1.9 Experience-Based Agency

- **memory search is context-sensitive:**

memory search unobtrusively or explicitly monitors activity in reasoning tasks or the environment and modifies its search of the knowledge store in an attempt to focus search on the most relevant items.

- **memory search is cost-controlled:**

memory search and retrieval consume a limited amount of resources during any given time slice, allowing reasoning tasks to execute; however, memory can

continue to search if given more time and resources in an attempt to improve quantity or quality of retrieval

- **knowledge storage is unified:**

memory search operates over a single store of knowledge, which serves as a repository for all the knowledge that reasoning tasks in the system might attempt to retrieve from memory, ensuring that the search for answers to a vague question can be dynamically focused on appropriate answers when feedback becomes available

- **reasoning is communicative:**

reasoning modules communicate to memory search not only their needs for information but also additional information about their reasoning state and the quality of retrieved items, enabling context-sensitive memory to focus on appropriate knowledge

- **reasoning is integrative:**

reasoning modules have the capability to accept information as it is retrieved, using integration and control techniques to filter out irrelevant retrieved items and incorporate relevant ones, ensuring that spontaneously or asynchronously retrieved items can profitably contribute to task performance.

- **communication and control are globally provided:**

the architecture provides facilities to support the control of processes operating in

parallel through a global task controller and support communication between memory and reasoning through a global working memory

These principles provide a template for implementing a context-sensitive asynchronous memory system and integrating it with a reasoning module to perform a real task. They illustrate how an agent must be constructed to dynamically focus memory search using information from a reasoning module, and in turn point out how reasoning modules must be designed or augmented to plug into this architecture. An implemented architecture based on these experience-based agent principles would serve as a toolkit for constructing a wide variety of agents based on context-sensitive asynchronous memory.

## **1.4. Scope of the Approach**

Viewed purely as a memory retrieval approach, the context-sensitive asynchronous memory approach can be applied to a wide range of tasks and domains. However, some potential targets are more useful than others. A context-sensitive asynchronous memory could be used to retrieve solutions to the 8-puzzle or to look up phone numbers by name in the phone book, but competitive approaches already exist which solve those problems. Other problems, such as finding information resources relevant to a human user's moment-to-moment interests or planning cases relevant to a complex planning state, are more difficult to solve effectively and efficiently with existing approaches and thus are more appropriate targets for the context-sensitive asynchronous memory approach.

Applying the approach appropriately requires a clear specification of the kinds of problems for which it provides the greatest benefit.

The scope of an approach to performing a task can be defined by analyzing the approach along dimensions such as the constraints of the task, the properties of the environment, the profile of relevant problems, and so on, and identifying the set of features along each dimension which indicate the approach will be particularly effective. Choosing the dimensions of evaluation of an intelligent system in turn depends on the perspective from which the system is analyzed.

An intelligent system can be viewed from a number of perspectives: as an agent in an environment, as a black box system with certain functional properties, or as an architecture which processes knowledge. From an agent-centered perspective it is important to look at a system in its environment, the profile of tasks and problems the agent will face in that environment, and the constraints the environment places on performing those tasks. From a functional perspective, it is important to look at the task a system performs in terms of its inputs, outputs and required transformations between them, as well as the way that task is instantiated into domains of content and the kinds of problems that arise in each domain. From an architectural perspective, it is important to look at the knowledge and processes necessary to transform the task's inputs into its outputs, as well as the knowledge available to each subprocess and how the processes can communicate with each other.

Examined through these perspectives, context-sensitive asynchronous memory is most applicable to tasks and domains which exhibit the following criteria:

- **Constraints from the Task:**

The task an intelligent system performs places major constraints upon suitable memory retrieval methods. The inputs of the task determine what information is available to ask questions; the computations required to transform the inputs to the outputs determine what kinds of questions need to be asked and hence what memory needs to store, and so on. When a single agent performs multiple tasks, the kinds of questions that need to be asked and information that need to be stored become increasingly varied and demanding. The context-sensitive asynchronous memory approach is most applicable to tasks which require a rich language of questions and answers to satisfy their needs for information.

- **Constraints from the Environment:**

The environment that a task is performed in can both constrain and liberate memory retrieval. The properties of an agent's environment determine the resource constraints upon task performance; the properties of the environment also determine how likely it is that a prior experience will be relevant to a new problem, and whether there will be any observable indicators of that relevance. Context-sensitive asynchronous memory requires environments where past information relevant to current problems exists, along with indicators which

connect current situations to relevant past information; it is especially applicable to environments where the resources available to inform retrieval are constrained.

- **Profile of Problems:**

Environments pose challenges to agents, problems they need to solve to effectively perform their tasks. The profile of problems an environment presents to an agent also affects memory retrieval. Even in a regular environment, some problems do not require the application of past experience. Problems that can be solved from first principles, like the 8-puzzle, do not need memory at all. Problems which require straightforward answers, like providing a phone number, can be solved by simple lookup. Difficult problems, however — problems which require a lot of computational effort to resolve, which may not recur exactly but which are similar enough to past problems to allow the reuse of previously generated knowledge — present opportunities for memory retrieval algorithms. This profile of problems is also a good indicator for case-based reasoning approaches (Kolodner 1993). By asking the right questions of an efficient and effective memory of past experiences, an agent may be able to avoid computational effort and solve problems more quickly or more effectively than solving them from scratch. Context-sensitive asynchronous memory is therefore most applicable to tasks which pose difficult problems.

- **The Questions Asked of Memory:**

Looking inside the box at how a reasoning process might communicate with a



memory process, the questions that reasoning must ask to solve its problems affects what memory retrieval approaches are appropriate. Crisp, well-defined questions can be answered efficiently. For example, “What name is attached to this Social Security number?” can be answered by a simple lookup if the knowledge is available. Fuzzy, poorly-specified questions are more difficult to answer. For example, answers to “What literary works are alluded to in the movie *Star Trek II: The Wrath of Khan*?” cannot be found by simple lookup, because no direct index is available. However, the question can still be answered as long as there is a way to evaluate potential answers — in the example, one might answer the question by watching the movie and examining a set of sample texts, which would give one a basis to say that *Moby Dick*, *A Tale of Two Cities*, and *Paradise Lost* are alluded to, while *The Invisible Man*, *Oliver Twist* and *The Odyssey* are not. Context-sensitive asynchronous memory is most relevant to tasks which pose poorly specified yet answerable questions.

- **The Information On Hand to Ask Questions:**

Again looking inside the box, the information available to a reasoning process determines a lot about how it can ask questions and thus what memory retrieval approaches are relevant. The information contained in an agent’s working memory determines not only how the specifications of the question can be formulated but also what other information is available to inform search. In our *Star Trek* example, even if the specification of the question is held fixed, simply watching the movie and hearing key lines of dialogue will provide cues which

could be used to suggest sample texts. Information about the task and the environment can thus not only shape questions but, beyond the questions, inform the process of finding answers. Context-sensitive asynchronous memory is most applicable when this kind of information beyond the questions is available.

- **Quantity of Available Knowledge:**

The prior knowledge available to the agent similarly determines how questions can be answered. If an agent has not had the opportunity to accumulate knowledge or has not retained the knowledge it has encountered, answering questions will be difficult regardless of the memory algorithm employed; conversely, if the amount or distribution of available knowledge makes answering questions easy then complex memory retrieval algorithms are not required. For example, if the prior knowledge base is very small then few approaches will complete with exhaustive search or simple hashes and indexes; if on the other hand a knowledge base is very large but nine out of ten items serve as a sufficient answer to the average question, then the overhead of a complex memory retrieval algorithm cannot compete with pure random selection.. The context-sensitive asynchronous memory retrieval approach is most appropriate for large knowledge bases with a paucity of useful answers.

- **Structure of Available Knowledge:**

Too much information without any structure can make questions difficult to answer — imagine searching the Library of Congress without an index. The

structure of knowledge determines how easy information is to find — either explicit structure in the form of labels and indices, or implicit structure embedded in the logical structure of knowledge. Returning to our example, the “cued suggested texts” strategy listed earlier presumes that the listener is familiar with Ahab’s curse upon *Moby Dick*, the opening line of *A Tale of Two Cities*, and Satan’s cry against heaven in *Paradise Lost*. Regardless of whether you see these connections as explicit indexing of key features of the work or implicit connections established as a result of reading a text, these connections make it possible to watch a scene-chewing villain in outer space and have Captain Ahab percolate to mind. These connections are also a necessary feature for context-sensitive asynchronous memory.

- **Limited Resources:**

The resources available to an agent to perform its task — and thus the resources available to the processes within the agent — also affects what memory retrieval approaches are relevant. Only a limited amount of time, storage or computational resources may be available, and these limits interact with each other. For clearly specified questions which can be quickly matched, a large knowledge base can be searched exhaustively in near-constant time with enough computational power; however, as time constraints increase or available computational power drops exhaustive search may no longer be possible. Even nearly unlimited resources for memory search are not helpful if the knowledge base can grow to be arbitrarily vast. The problem of retrieval cost is worse if the bottleneck is matching poorly

specified questions against complex potential answers. Returning to our example, even someone spending their life tackling this single problem could not afford to match every work of literature known to man against the text of the film — hence the importance of the “cued suggested texts” strategy. The challenge for these kinds of questions is finding the best potential answers to limit the costs of matching. Context-sensitive asynchronous memory is more competitive with competing approaches when resource limits are an issue.

The context-sensitive asynchronous memory approach is most applicable to agents performing classes of tasks instantiated in classes of domains which satisfy the above profile; however, context-sensitive asynchronous memory does not depend on any knowledge of a particular task or domain or any particular environment. Context-sensitive asynchronous memory is domain independent: it is applicable to a wide variety of tasks and problems from simple search applications to complex cognitive agents.

## **1.5. The Claims**

The core claim of the context-sensitive asynchronous memory approach is that the problem of efficiently answering poorly specified questions can be solved by a context-sensitive asynchronous memory working with an integrative reasoner in a rich, challenging but regular environment. This general assertion can be unpacked into a series of more specific claims about the usefulness, properties and sources of power of context-sensitive asynchronous memory and what reasoners must do to take advantage of it.

The key claims of the context-sensitive asynchronous memory approach are:

- **Claim 1: Interleaving Memory with Reasoning and Action is Useful**

An efficient, domain-independent solution to the problem of retrieving useful answers from large knowledge bases given under-specified queries is to interleave memory retrieval with task performance and use feedback from the task or environment to guide the search of memory.

- **Claim 2: Context-Sensitive Asynchronous Memory Enables Interleaving**

Interleaving memory retrieval with and exploiting feedback from task performance can be achieved in a domain-independent way using a context-sensitive, autonomous memory retrieval process capable of spontaneous and anytime retrieval.

- **Claim 3: A Rich Representation Makes Context-Sensitive Asynchronous Memory Domain-Independent**

A rich, reified, grounded semantic network representation enables context-sensitive memory retrieval processes to retrieve useful information in a domain-independent way for a wide variety of tasks.

- **Claim 4: Communication, Integration and Control are Required to Use Context-Sensitive Asynchronous Memory**

To effectively use a context-sensitive asynchronous memory to retrieve useful answers, a task must be able to work in parallel with a memory process,

communicate with it, provide feedback to it, and must possess integration mechanisms to incorporate asynchronous retrievals provided by the memory.

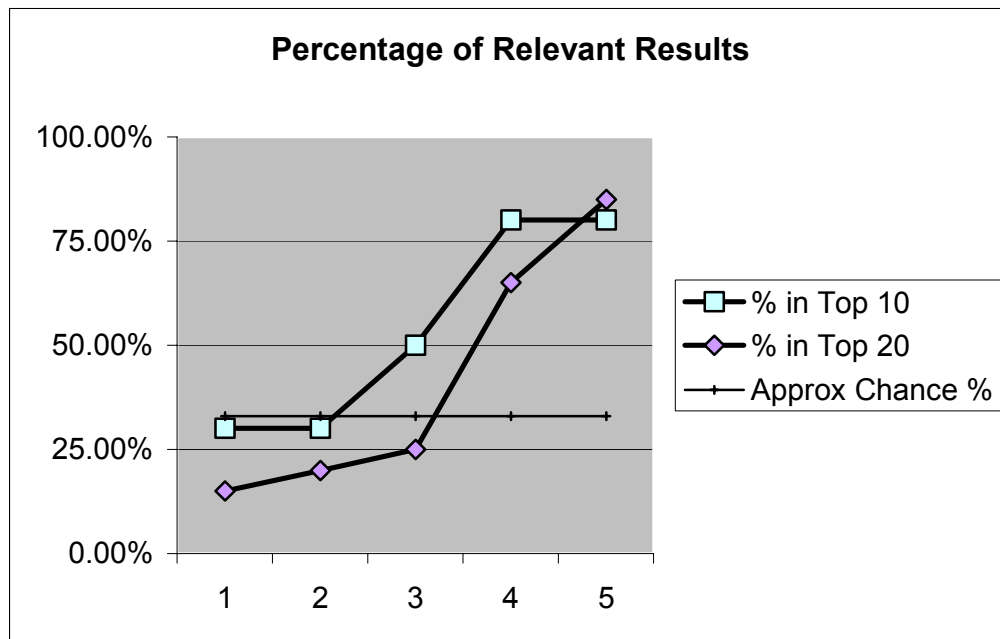
## **1.6. The Evidence**

To test these claims about the context-sensitive asynchronous memory approach and its companion, the experience-based agent architecture, I constructed the Nicole system to serve as a testbed for a series of experiments and evaluations. The Nicole system instantiates the experience-based agent approach: built around a context-sensitive asynchronous memory retrieval process operating over a rich, reified, grounded semantic network representation and controlled by a parallel task control architecture, it allows the construction of agents based on context-sensitive asynchronous memory as well as the analysis of the properties and sources of power of the context-sensitive asynchronous memory approach itself.

### **1.6.1. Claim 1: Interleaving Memory with Reasoning and Action is Useful**

*An efficient, domain-independent solution to the problem of retrieving useful answers from large knowledge bases given under-specified queries is to interleave memory retrieval with task performance and use feedback from the task or environment to guide the search of memory.*

The first claim is a claim about the problem, namely, that it can be solved by interleaving memory retrieval and task performance and using feedback from the task to



**Figure 1.10. Quality of Recommendations On Web Search**

improve memory retrieval. This solution is both useful and broadly applicable: the approach can provide real benefits to real users and real problems, and can be applied to a variety of tasks and domains. The evidence for this claim can be divided into two parts: evidence of utility, and evidence of general applicability.

- **SubClaim: Exploiting feedback is useful**

*Interleaving memory retrieval and problem solving to exploit user feedback can improve memory retrieval.*

**Evidence:** To evaluate this claim, I used Nicole to develop an information retrieval application called Nicole-IRIA, which interleaves search for information with user browsing and exploits user feedback to improve memory retrieval.

Nicole-IRIA was evaluated on two distinct information retrieval domains: general web search and personnel management.

Evaluation of Nicole-IRIA on these domains demonstrated empirically that Nicole-IRIA could use feedback from browsing to improve the quality of memory retrieval as measured by relevance to user interests. Figure 1.10 illustrates a typical set of results on the general web search domain; the X axis represents number of user clicks tracked and the Y axis represents the percentage of recommended pages matching the user's current interests. Using feedback from user browsing, Nicole-IRIA was able to recommend pages which matched user interests with a significantly greater degree of accuracy than chance and a higher degree of accuracy than the random ordering produced by a source search engine.

- **SubClaim: Exploiting feedback is generally applicable**

*An approach based on interleaving memory retrieval and problem solving to exploit user feedback can retrieve useful information for a wide variety of tasks.*

**Evidence:** To evaluate this claim, Nicole's context-sensitive asynchronous memory module, MOORE, was used to retrieve useful information for three distinct tasks, including the experience-based agent systems Nicole-IRIA (hypertext information retrieval) and Nicole-MPA (planning), as well as the standalone ISAAC story understanding system (Moorman 1997). In each of these systems, Nicole-MOORE successfully performed memory retrieval sufficient to enable task performance, including successful information retrieval in Nicole-

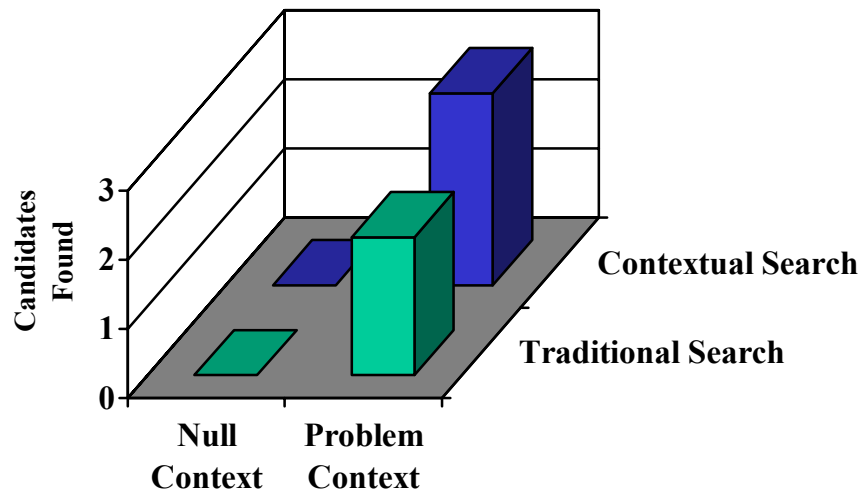


IRIA and successful case-based planning in Nicole-MPA. While ISAAC did not directly take advantage of MOORE's context-sensitive asynchronous retrieval properties, MOORE nonetheless functioning as a "traditional" memory system did successfully support ISAAC's retrieval needs as it read real science-fiction stories at human-level performance (Moorman 1997).

### **1.6.2. Claim 2: Context-Sensitive Asynchronous Memory Enables Interleaving**

*Interleaving memory retrieval with and exploiting feedback from task performance can be achieved in a domain-independent way using a context-sensitive, asynchronous memory retrieval process.*

The second claim is a claim about the solution: that interleaving memory retrieval and task performance and exploiting feedback between them can be accomplished by the context-sensitive asynchronous memory architecture. This claim can be decomposed into a variety of subclaims, illustrating the capabilities of context-sensitive asynchronous memory to exploit information from a variety of sources to improve retrieval and provide benefit compared to competing approaches.



**Figure 1.11. Exploiting Context in Retrieval**

- **SubClaim:** Context-sensitive asynchronous memory usefully exploits feedback

*A context-sensitive asynchronous memory can use contextual information to improve quality and quantity of retrieval.*

**Evidence:** To test this claim, I conducted two evaluations. First, I implemented a planning system called Nicole-MPA, which was able to use contextual information derived from problem statements and problem solving traces to improve the quantity of retrieved cases over a default retrieval approach. Figure 1.11 shows the results of one experiment with Nicole-MPA: when attempting to find a small number of relevant cases in a knowledge base populated with irrelevant cases the use of contextual information facilitated successful retrieval.

Ablating the feedback links between reasoning and memory (the “null context” condition) severely degraded the system’s retrieval capabilities, and using a context-insensitive form of spreading activation rather than the context-directed spreading activation approach also degraded retrieval.

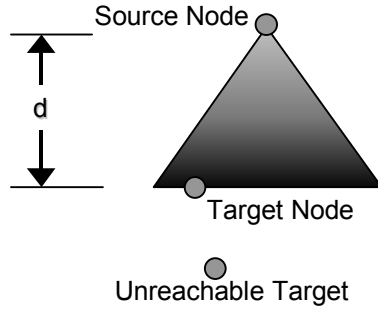
Then, I evaluated Nicole-IRIA on a “site search” domain in which Nicole-IRIA recommended information based on user browsing. When Nicole-IRIA was provided with information about potential relationships between user browsing and target concepts, it produced retrievals superior (as measured by relevance to user interests) to those in a control condition.

- **SubClaim: Context-sensitive asynchronous memory exploits feedback generally**

*A context-sensitive asynchronous memory can exploit contextual information from a variety of sources.*

**Evidence:** To evaluate this claim, I conducted evaluations which demonstrated that Nicole’s contextual memory system can function using several different classes of contextual information. This contextual information included data extracted from problem statements and intermediate problem solving steps in Nicole-MPA, implicit priming information extracted from user browsing traces in Nicole-IRIA, and explicit user hints in Nicole-IRIA.

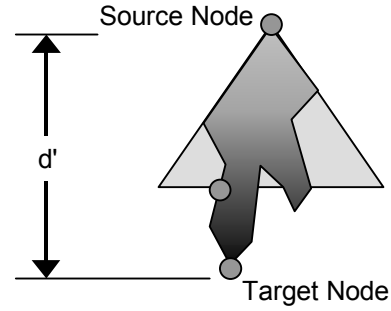
### Traditional Spreading Activation



Maximum Semantic Distance:

$$d = \frac{\log P_{initial} - \log T_{cutoff}}{\log b}$$

### Context-Directed Spreading Activation



Maximum Semantic Distance:

$$d' = \frac{\log P_{initial} - \log T_{cutoff}}{\log(b_{total} R_{base} + b_{active} R_{gating}) - \log(R_{base} + R_{gating})}$$

**Figure 1.12 Comparing Traditional and Context-Directed Spreading Activation**

- **SubClaim: Context-sensitive asynchronous memory provides superior performance**

*A context-sensitive asynchronous memory can exploit contextual information to retrieve information not accessible to other approaches given the same effort.*

**Evidence:** To demonstrate this claim, I performed a theoretical analysis of *context-directed spreading activation*, the mechanism for context-sensitive memory search. This analysis shows that given a fixed amount of retrieval effort, context-directed spreading activation can exploit contextual information to increase the range of concepts reachable over standard spreading activation (Figure 1.12).

### 1.6.3. Claim 3: A Rich Representation Makes Context-Sensitive Asynchronous Memory Domain-Independent

*A rich, reified, grounded semantic network representation enables context-sensitive memory retrieval processes to retrieve useful information in a domain-independent way useful to a wide variety of tasks.*

The third claim is that operating over a particular kind of richly represented knowledge base called an experience store enables a context-sensitive asynchronous memory to provide benefit to a wide range of tasks or to a complex single agent performing a variety of tasks. This claim can be decomposed into several subclaims: first, that an experience store enables context-sensitive asynchronous memory retrieval to operate, second, that it can support a variety of tasks; third, that it can support highly complex representations.

- **SubClaim: Rich knowledge representation is required**

*A reified, bidirectional, grounded semantic network enables context-sensitive asynchronous memory retrieval.*

**Evidence:** I have demonstrated this claim using Nicole by ablating key properties of the knowledge representation, such as bidirectional links or reified relations. Removing these elements degrades the quality and quantity of retrieval or disables the context-sensitive asynchronous memory system entirely.

- **SubClaim: Rich knowledge representation enables generality**

*A reified, bidirectional, grounded semantic network can support the representation needs of a wide variety of tasks.*

**Evidence:** To demonstrate this claim, I used Nicole's rich knowledge representation for knowledge representation in planning (Nicole-MPA) and hypertext information retrieval (Nicole-IRIA), and contributed Nicole's knowledge representation to the ISAAC story understanding project (Moorman 1997).

- **SubClaim: A rich knowledge representation supports complex tasks**

*A reified, bidirectional, grounded semantic network is expressive enough to support complex logical expressions and rich enough to support a wide variety of world knowledge.*

**Evidence:** To demonstrate this claim, I performed a theoretical analysis which shows that reified grounded semantic networks have the same expressive capacity as Nth-order logic and equal representational capacity to knowledge representations such as the Knowledge Machine and CYC.

#### **1.6.4. Claim 4: Communication, Integration and Control are Required**

*To effectively use a context-sensitive asynchronous memory to retrieve useful answers, a task must be able to work in parallel with a memory process, communicate*

*with it, provide feedback to it, and must possess integration mechanisms to incorporate spontaneous or asynchronous retrievals provided by the memory.*

The final claim states that reasoners need certain properties to effectively use context-sensitive asynchronous memory.

- **SubClaim: A unified knowledge store is necessary**

*To effectively use a context-sensitive asynchronous memory to retrieve useful answers, a task must represent knowledge in the experience store.*

**Evidence:** I demonstrated this empirically in Nicole-MPA through ablation experiments of its plan tagging mechanism. Because it extends the SPA single-case case-based planning system and uses SPA’s native plan representation, Nicole-MPA’s plans are opaque to the experience store. To overcome this limitation, Nicole-MPA uses “plan tags” to make the content of plans visible to the experience store. Ablating plan tags — effectively removing the content of Nicole-MPA’s knowledge from the experience store — severely degrades memory retrieval.

- **SubClaim: Memory/task parallelism is necessary**

*To effectively use a context-sensitive asynchronous memory to retrieve useful answers, a task must pass requests to and operate in parallel with the asynchronous memory.*

**Evidence:** I demonstrated this empirically in Nicole-MPA by ablation of the memory retrieval request system. Ablating the memory retrieval request system or interleaved processing of memory retrieval and task performance disable memory retrieval entirely.

- **SubClaim: Feedback is required**

*To effectively use a context-sensitive asynchronous memory to retrieve useful answers, a task must provide feedback to the context-sensitive memory system.*

**Evidence:** I demonstrated this empirically by varying contextual information available to Nicole-MPA and Nicole-IRIA. In Nicole-MPA, eliminating sources of contextual information degrades the quantity of retrieved items, and in Nicole-IRIA, eliminating sources of contextual information degrades the quality of retrieved items.

- **SubClaim: Integration mechanisms are required**

*To effectively use a context-sensitive asynchronous memory to retrieve useful answers, a task must have integration mechanisms which enable it to accept spontaneous or asynchronous retrievals and incorporate them into its current processing state.*

I demonstrated this empirically in Nicole-MPA and Nicole-IRIA by ablating parts of the integration system. In Nicole-MPA integration mechanisms are necessary to solve difficult planning problems in certain domains, and in Nicole-IRIA the



evaluative function of integration mechanisms is necessary to preserve the quality of retrieved items.

## **1.7. Benefits and Contributions**

The research described in this thesis provides benefits for real users and real tasks as well as contributions to artificial intelligence research.

### **1.7.1. Benefits to Users and Tasks**

Context-sensitive asynchronous memory enables users performing real tasks to find answers to questions which would otherwise be unanswerable. By using feedback from the environment and task performance, a context-sensitive asynchronous memory can find answers which would not have otherwise been found. When answers are plentiful, a context-sensitive asynchronous memory can bring the most relevant answers to the forefront, ensuring that the best answers are not overlooked.

Context-sensitive asynchronous memory enables agents to efficiently balance resources between task processing and memory retrieval. By incrementally searching the knowledge base, a context-sensitive asynchronous memory supports anytime retrieval of the best answer found so far and thus enables agents to satisfice, terminating the search with the first suitable answer found.

Context-sensitive asynchronous memory retrieval furthermore enables agents easily use additional information to improve retrieval. By explicitly recording and maintaining

the state of a search, a context-sensitive asynchronous memory supports incorporating new retrieval specifications, thus enabling agents to refine their questions in the hope of getting better answers.

From the end-user perspective, these capabilities enable an intelligent system based on a context-sensitive asynchronous memory to exhibit flexible, accurate, human-like performance: rather than simply returning fixed answers in response to questions, a system based on context-sensitive asynchronous memory can work with a user, unobtrusively watching the user's performance on the task to help it reorganize its presentation of information or explicitly taking hints from the user to help it find better answers — all without discarding the work it has already performed.

### **1.7.2. Contributions to Artificial Intelligence**

Beyond simply being useful for solving real problems, this research makes theoretical and technical contributions to several areas of artificial intelligence research, including agent systems, memory, knowledge representation, task control, planning, and information retrieval.

The experience-based agent architecture enables designers to construct agents with context-sensitive asynchronous memories. By providing an architecture for knowledge representation and task processing that supports a context-sensitive asynchronous memory, along with a specification of how to construct reasoning tasks which work with a context-sensitive asynchronous memory, the experience-based agent architecture

provides a framework for building complete systems which incorporate both context-sensitive asynchronous memory principles and real performance tasks.

One of the most important contributions of this framework is the asynchronous memory architecture itself. Asynchronous memory enables a novel style of memory retrieval combining anytime retrieval, revisable specifications and spontaneous reminders. By providing an explicit language for information requests, treating them as first-class objects, and processing them incrementally, an asynchronous memory enables the memory to incrementally consume resources attempting to find an answer, to provide answers when demanded, to accept revised specifications when provided, and to proactively alert reasoning when a result is found.

Hand in hand with this contribution is the context-sensitive memory search architecture. Context-sensitive memory search improves the precision of memory search. By using cues from ongoing reasoning to guide search, context-sensitive search focuses the search effort on the portion of the knowledge base most likely to yield useful answers.

The context-directed spreading activation algorithm is an important enabling factor for this approach. Context-directed spreading activation enables scaleup of the context-sensitive approach to more complex knowledge bases. By using the existing set of active nodes to change the weights of connections dynamically, context-directed spreading

activation reduces the effective branching factor of spreading activation and enables the same knowledge base to be used in a variety of ways.<sup>3</sup>

Another important enabling factor is the experience-store knowledge representation. The experience store enables the context-sensitive asynchronous memory approach to be applied to a variety of tasks. By providing a rich knowledge representation capable of representing highly complex knowledge which supports the features required for the context-directed spreading activation algorithm, the experience store provides a task-independent basis for context-sensitive search.

Another contribution of the framework is the parallel task control system. The parallel hierarchical task network facilitates integrating memory and reasoning systems. By coupling a task decomposition system that supports complex task logic with a parallelism and synchronization language which supports complex task coordination, the parallel hierarchical task network enables the construction of reasoners which can provide feedback to a parallel memory process and accept proactive retrievals from those processes.

This research contributes to the field of planning as well. The multi-plan adaptor algorithm enables dynamic recombination of multiple planning cases. By providing a

---

<sup>3</sup> A patent has been applied for at Georgia Tech on certain aspects of context-sensitive asynchronous memory, the CDSA algorithm, and the experience-based agent architecture embodied in Nicole.

way to clip the relevant parts of a case and splice them into an active planning process, the MPA algorithm provides a method for dynamically combining multiple planning cases to improve performance.

This research also contributes to the field of information retrieval. Nicole-IRIA provides a way to use a user's browsing efforts to find information relevant to their interests. By representing information resources in an experience store and mapping user queries and user browsing to information requests and contextual feedback, Nicole-IRIA can find information in memory relevant to the user's goals and rank it according to a user's current interests.<sup>4</sup>

Finally, this research has contributed theoretical and practical analyses of each of the components of the experience-based agent framework, providing a way to predict the classes of tasks and situations in which the approach will provide benefits, enabling users and designers to evaluate the utility of the approach for their task and applications.

## **1.8. Outline of the Thesis**

The thesis is divided into four major sections – Introduction, Approach, Evaluation, and Impact.

---

<sup>4</sup> Patents have been applied for at Enkia Corporation on certain aspects of the recommendation engine embodied in Nicole-IRIA.

The Introduction section establishes the general research area and the problem this thesis addresses. This chapter introduced the problem of efficiently finding answers from large knowledge base using context, illustrated why this problem is important for cognitive agents and other intelligent systems, and reviewed the context-sensitive asynchronous memory approach to a solution. Chapter 2: Memory discusses the problems of memory in greater detail, unpacking the desiderata discussed earlier in greater detail, discussing related work in memory with respect to the desiderata, and identifying what is novel about the approach to memory retrieval advanced in this dissertation.

The Approach section discusses the technical meat of the thesis. Chapter 3: Context-Sensitive Asynchronous Memory discusses the approach in detail. Chapter 4: Integration Mechanisms discusses the impact of the context-sensitive asynchronous memory approach on reasoning and how reasoners must be structured to exploit such a memory. Chapter 5: Experience-based agency closes the loop, discussing how to support integrating context-sensitive asynchronous memory with reasoning tasks with a general architecture and proposing the experience-based agency theory as one solution.

The Evaluation section discusses how this technical work has been evaluated. Chapter 6: Methodology discusses the approach behind the evaluations. Chapter 7: Implementation introduces the implementation of the experience-based agent idea in the Nicole system. Chapter 8: Case Study: Planning discusses an investigation in applying this approach to planning in the Nicole-MPA system, while Chapter 9: Case Study:

Information Retrieval discusses an investigation in applying this approach to information retrieval in Nicole-IRIA. Finally, Chapter 10: Feasibility addresses how the results of the case studies, along with the results of additional experiments conducted on Nicole itself, demonstrate the feasibility of context-sensitive asynchronous memory and experience-based agency.

Finally, the Impact section discusses what this research has contributed. Chapter 11: Conclusions summarizes the major claims of the research, the strengths and limitations of the approach, and draws conclusions about what we have learned from this research and where this research can lead in the future.

Now, on to the problems of memory, and how this research addresses them.

# CHAPTER II.

## RELATED WORK

---

*What is memory retrieval, what approaches to it exist, and what does this research add?*

Finding good answers from large knowledge bases given poor questions is just one of the challenges that general cognitive agents pose for memory retrieval. To cope with complex problems and multiple tasks, general cognitive agents require large stores of knowledge capable of holding complex information. However, when agents are faced with a problem they often lack the information they need to precisely specify an answer and almost always lack the time or processing power to exhaustively search all of their knowledge. This problem is complicated in agents which must continue to think or act to maintain adequate performance on their tasks; such systems need ways to integrate the memory retrieval problem with ongoing performance of their tasks.

As discussed in Chapter One, these demands can be teased apart into a set of explicit desiderata for memory retrieval systems:

- **Desideratum 1: Able to store a wide variety of information**
- **Desideratum 2: Provides a general method for accessing that information**
- **Desideratum 3: Scales to large multifunction knowledge bases**
- **Desideratum 4: Manages cost of retrieval**



- **Desideratum 5: Preserves accuracy of retrieval (in the face of resource limits)**
- **Desideratum 6: Exploits task/environmental information**
- **Desideratum 7: Exploits extra resources if available**
- **Desideratum 8: Potentially interleavable with reasoning**
- **Desideratum 9: Provides guidelines for reasoning integration**

Satisfying as many of these desiderata at once is important for constructing not only general cognitive agents but for any system faced with complex but vague questions, large and rich knowledge bases, and limited resources to find answers.

Context-sensitive asynchronous memory addresses these challenges simultaneously. This chapter demonstrates the value of this unified approach by comparing and contrasting it with prior work in memory retrieval.

The chapter begins with a brief overview of existing approaches to memory retrieval and their limits, then sets out a framework for analyzing memory retrieval systems in terms of functional, performance and technological considerations, highlighting with a broad brush context-sensitive asynchronous memory's contributions in each of those areas. The chapter then examines potential tradeoffs between functional, performance and technological aspects of memory retrieval systems.

With this foundation laid, the balance of the chapter compares and contrasts context-sensitive asynchronous memory and prior research with respect to the desiderata listed above. The chapter examines the major issues of each desideratum in depth, first presenting functional, performance and technological metrics for evaluation, then discussing prior work addressing the desideratum, identifying limitations of that prior work, and finally examining how context-sensitive asynchronous memory builds on prior work to overcome those limitations.

## **2.1. Existing Approaches and their Limits**

In general, existing approaches to memory retrieval target one or more of these desiderata separately. Separate research traditions address large knowledge bases, complex knowledge, use of knowledge for several tasks, or speed, efficiency and contextual issues, but no existing system addresses more than three or four of these issues at the same time. Instead, most systems trade off one or more of these capabilities in exchange for the others.

Painting with a broad brush, we can characterize the bulk of approaches to memory retrieval as *task-specific*, attempting to satisfy the desiderata of one task or one class of tasks (e.g., Goel et al. 1994, 1996, Hanks & Weld 1992, 1995, Peterson et al. 1996 Veloso 1995). *Case-based* and *dynamic memory systems* focus more broadly on the properties of memory systems, but don't directly address efficiency issues (e.g., Schank 1982, Kolodner 1983, Bareiss et al. 1988, Domeshek 1992). A variety of *performance*

*memory systems* have built on the case-based and other approaches, exploiting parallelism to achieve fast performance at the expense of efficient use of processing resources (e.g., Kolodner 1988, Kettler et al. 1993). *Computational models of human memory* point the way to efficiently exploiting information in the environment, but are often very narrowly focused on single memory tasks (e.g., Gentner & Forbus 1991, Jones & Langley 1995, Lange & Wharton 1994, Law et al. 1994, Thagard et al. 1990); general *cognitive architectures* apply these memory approaches to a wider range of tasks but are not structured for use in more general artificial intelligence applications (Anderson 1983, 1990, Newell 1990). Finally, a variety of approaches focus on developing *general knowledge bases*, but these systems focus most on general representations and overall ontologies without considering retrieval problems in great detail (e.g., Lenat & Guha 1990, Clark & Porter 1996).

Most of these systems were not designed to serve as a memory system for a cognitive agent, although some of them are clearly steps on the path towards that goal. By examining existing approaches and their strengths with respect to the desiderata for memory in general cognitive agents, we can determine where existing approaches fall short of the requirements and outline how the state of the art should be extended.

## **2.2. Evaluating Memory Approaches**

A vast array of potential approaches exist, from relational databases to cognitive architectures, from search engines to associative memories, and from dynamic case

libraries to connectionist neural networks. How can these different approaches to memory be evaluated or compared with respect to these desiderata?

To evaluate a memory system one may ask a number of reasonable questions, such as: What kind of knowledge can its store hold? What kinds of questions can be asked? How long does it take to get an answer? How good are the answers that are returned? How does it find those answers? These questions can be broken down into three primary groups: what a memory system does, how well it does it, and how it does it.

“What a system does?” inquires about the *functional* capabilities of a memory system — what conditions are necessary for the approach to bring information to mind? “How well it does it?” asks about the *performance* characteristics of a memory system — what costs does it incur, and what is the quality of its output? “How it does it?” asks about the *architecture* of a memory system — what technologies were used to create it, and how easily can they be extended to other tasks?

Other metrics for analyzing memory systems exist, such as degree of fit to human data (for cognitive models of memory systems), availability on operating system platforms (for software modules for application systems), customizability, extensibility, and so forth. While these issues can be very important, they are not relevant to the core issues of this research and we will not discuss them further in this dissertation.

### 2.2.1. Functional Capabilities of Memory Systems

The most significant questions about a memory system's functional capabilities are its external characteristics: what are its inputs, what are its outputs, and what relationships exist between input and output (e.g., Marr 1982)? Secondly, one may ask questions about a system's internal structure. For example, what representations and processes does it use? Of course, internal representations and processes are only significant if they are exposed through a system's functional interface or if they are relevant to extending an approach to new tasks or domains.

One of the most important capabilities of a memory system is its *representational power*: the expressiveness of its query and representation languages and the generality of its search and matching algorithms. Also important is its *knowledge organization*: how concepts and data are structured in the knowledge base to facilitate retrieval. Another important dimension is a system's *functional autonomy*: does the memory act like a subroutine, requiring a questioner to wait until a response is generated, or does it act like a thread, processing a query without intervention and returning answers on its own when found? Of particular interest to this research is whether a memory system can return results in an anytime fashion and whether a query can be refined without effectively discarding the results of the original search.

Viewed from a functional perspective, context-sensitive asynchronous memory attempts to maximize the representational power of the knowledge base and to exploit

that representation to minimize the need for explicit knowledge organization; furthermore, context-sensitive asynchronous memory attempts to break new ground on functional autonomy by enabling memory to operate in parallel with other reasoning tasks.

### **2.2.2. Performance Characteristics of Memory Systems**

Given a memory system with a specific set of functional capabilities, how can its performance be evaluated? One of the most important dimensions is a memory system's *computational complexity*: the time and space cost of adding items to a knowledge base and the time and space cost of querying the knowledge base. Another important consideration is a memory system's *efficiency*: how effectively the memory expends the time and space cost that it consumes. Memory systems may provide *guarantees* about their behavior: for example, memory search may be comprehensive or partial, and if partial may have lesser or greater degree of precision in search; matching can be exact or approximate; and overall retrieval itself can be deterministic, random, or context-sensitive process. Finally, the *quality* of a memory retrieval system's results are also important: we can evaluate a memory system by measuring the recall and precision of the information it retrieves with respect to a query, the accuracy and quantity of information it returns, or the amount of information gain that can be achieved per unit of effort.

One could argue about the precise breakdown of these categories: one could argue that (for example) asynchrony of retrieval is more akin to a performance characteristic, or

that guarantee of retrieval is a functional capability. For the purposes of this axiology, we view a memory system as an opaque software component — a “black box” with a particular applications programming interface, or “API”, intended to fulfill some behavior specification, or “contract”.

We categorize the features of the API of a memory system (such as the matching language) as functional capabilities, and categorize the contract of the API (such as retrieval guarantees) as performance characteristics. We categorize more traditional metrics used to measure memory systems, such as speed or precision, in the performance characteristics grouping. The reason for this breakdown is that the inputs and outputs of a memory — the API — come attached to the black box, whereas “guarantees” provided about how those inputs and outputs work are really statements about the system’s behavior which can be empirically measured and evaluated — statements which might be wrong.

Viewed from a performance perspective, context-sensitive asynchronous memory attempts to hold computational complexity constant by relaxing guarantees of comprehensiveness and accuracy in retrieval, and within those constraints exploiting feedback to try to improve the precision of search and the quality of retrieval.

### 2.2.3. Technologies for Memory Retrieval

The task of memory retrieval is supported by a variety of technologies, each of which can be used in a variety of systems but which have specific properties that make them more or less suited for different tasks.

**Knowledge organization** approaches such structure the knowledge base using trees or labeled graphs in an attempt to enable specific classes of questions to be answered efficiently; examples include the redundant discrimination networks of CYRUS (Kolodner 1983), shared feature networks used in a variety of machine learning approaches (for a discussion see Kolodner 1993), and more specialized structures designed for individual systems, such as combination of goal indexing and discrimination networks used for plan retrieval in PRODIGY/ANALOGY (Veloso 1995), the dimension indices used to retrieve relevant cases in HYPO (Ashley 1990, 1991), and the multiple knowledge sources used for efficient retrieval of different classes of knowledge in KRITIK (Peterson et al. 1994, Goel et al. 1996).

**Symbolic associative memories** attempt to build symbolic matching networks which enable information to be quickly retrieved based on its content; examples include highly optimized production systems such as OPS-5 (Forgy 1981) as used in Soar (Newell 1990).

**Semantic networks** can be considered to be implementations or close relatives of symbolic associative memories. Semantic networks employ symbolic network structures



for storage, representation and retrieval of information. They are distinguished from other representations usually by processing algorithms which exploit their graph structure, such as spreading activation and marker passing approaches. Examples include (Quillian 1966, Woods 1975, Anderson 1983, Jones 1989, Jones 1993, Jones & Langley 1995, Wolverton 1994) and further discussion can be found in (Sowa 1984, Sowa 1991 particularly Schubert 1991; also Russel & Norvig 1995).

**Case libraries** use larger grained data structures as the fundamental units of knowledge and employ a variety of labeling, indexing and retrieval schemes, many of them overlapping with the techniques described above. For an overview, see (Kolodner 1993); examples of systems using case libraries include (Kolodner & Simpson 1988, Kolodner 1988, Bareiss et al. 1988, Hammond 1989, Kettler et al. 1993).

**Nonsymbolic associative memories**, like symbolic associative approaches, also attempt to enable information to be quickly retrieved based on its content; in nonsymbolic approaches, however, the indices, representations or both are distributed across feature arrays or other nonsymbolic data structures. A canonical example is (Kohonen 1984)'s proposal for self-organizing associative memories; Domeshek's (1992) proposal for fast matching of feature vectors can be viewed to as a member of this category.

**Neural networks** are typical examples of nonsymbolic associative memories, distributing representations across subsymbolic network structures and using activation

from novel inputs to reinstantiate past patterns. For an overview and a representative set of examples of neural network systems, see Rumelhart & McClelland (1986).

**Other approaches.** Construed broadly, a variety of other approaches to memory retrieval exist. Caching memory architectures collect past queries and answers to improve speed of memory retrieval, but in general caching memory systems have very simple query and representation languages. Database systems have similarly simple knowledge representations, but support more complex query languages and specialized memory retrieval algorithms which make particular classes of queries more efficient. These limitations on knowledge representation and access methods make many of these approaches unsuitable for artificial intelligence applications.

**Combined approaches.** Many of these technologies do not constitute complete memory retrieval systems in and of themselves and must be instantiated in more complete systems to serve the needs of a reasoning task. For example, the OPS-5 production system (Forgy 1981) serves as the associative memory in the Soar architecture (Newell 1983); spreading activation itself is usually embedded in larger systems like ACT\* (Anderson 1983) or Eureka (Jones & Langley 1995); and so forth. Also, many of these approaches are often combined: for examples, a variety of systems have explored integrating neural and symbolic approaches (e.g., Hendler 1989, Holyoak 1991, Holyoak & Hummel (in press), Lange & Wharton (1994), and Rosenbloom 1993).

**Where context-sensitive asynchronous memory fits in.** Technologically, context-sensitive asynchronous memory is a semantic network approach employing spreading activation as its memory retrieval mechanism. Furthermore, the approach imposes constraints upon its knowledge representation to ensure that spreading activation exhibits the content-addressable behavior of an associative memory approach, and exploits features of its rich knowledge representation to emulate some of the advantages of knowledge organization approaches without requiring system designers to know how to structure their knowledge bases. This combination of technologies and features of technologies is designed to make the context-sensitive asynchronous memory approach not only applicable to a wide range of intelligent systems, but also feasible to apply without extensive customization.

### **2.3. Apparent Tradeoffs and Opportunities**

Viewed from these perspectives — functional capabilities, performance characteristics, and implementing technologies — existing systems have a wide range of functional capabilities and equally wide range of performance characteristics. Interesting tradeoffs are often made between particular capabilities and improved performance along one or more dimensions, sometimes exploiting the particular strengths of a technology to do so (Figure 2.1).

For example, one typical tradeoff is between efficiency of memory retrieval and expressiveness of representation and query languages. Generality of search and matching

algorithms are another capability often sacrificed for efficiency: for many tasks, special-purpose memory algorithms have been developed which are highly efficient and effective at fulfilling their functional role.

While this research does not wish to downplay the role of single-task memory systems in producing effective applications that solve people's problems, if the ultimate goal of artificial intelligence remains the creation of general-purpose intelligent systems capable of performing multiple tasks at human-level competence, general memory retrieval algorithms are required.

Context-sensitive asynchronous memory focuses on general memory retrieval, but not necessarily from the perspective of "what tradeoffs are required?" Instead, the context-sensitive asynchronous memory model proposes that a potentially beneficial relationship exists between several aspects of memory systems: an expressive representation and matching language, coupled with an appropriately designed autonomous search process, can improve the precision of memory search and the efficiency of retrieval. This synergism can be exploited without sacrificing the generality of memory retrieval.

Approach	Task-specific Methods	General Methods	Performance Approaches	Cognitive Architectures	Multifunction KBs	Experience-based agents
EXAMPLES	PRODIGY ROUTER	CYRUS, ORCA	PARADYME PARKA	SOAR ACT*	CYC KM	Nicole-MPA Nicole-IRIA
Retrieval Efficiency	<b>yes</b>	<b>yes</b>	<b>yes</b>	<b>yes</b>	<b>no</b>	<b>yes</b>
Multitask Applicability	<b>no</b>	<b>yes</b>	<b>yes</b>	<b>yes</b>	<b>yes</b>	<b>yes</b>
Query Generality	<b>no</b>	<b>no</b>	<b>yes</b>	<b>n-a</b>	<b>yes</b>	<b>yes</b>
Cost/ Accuracy Control	<b>n-a/ no</b>	<b>no/ no</b>	<b>n-a/ no</b>	<b>no/ no</b>	<b>no/ no</b>	<b>yes/ yes</b>
Task Inter-action Policy	<b>no</b>	<b>n-a</b>	<b>yes</b>	<b>n-a</b>	<b>no</b>	<b>yes</b>
Exploit Context	<b>yes</b>	<b>no</b>	<b>no</b>	<b>yes</b>	<b>no</b>	<b>yes</b>
Strengths	exploit past experience	not tied to a single task	general query methods	general processing	general representation	general architecture
Weaknesses	tied to single task, query	tied to a single query model	KB size, efficiency	limited retrieval scope	retrieval unspecified	single-task methods faster

**Figure 2.1: Comparison Chart of Memory Systems**

The key is to view memory as one piece of an autonomous agent, composed of multiple concurrently running processes, performing a task or set of tasks in a dynamic environment. Representing all the knowledge of a multiple-task agent in a single data structure requires a rich, expressive knowledge representation language, and an appropriately designed memory retrieval algorithm can use that rich representation to guide search of memory based that the context of the agent's reasoning and the recent history of its environment. This context-sensitive search is more precise than existing

approaches and enables the system to find information more efficiently. Furthermore, if memory is allowed to operate as an autonomous process, context information will be naturally generated as part of the agent's reasoning and experience to further guide information retrieval.

This approach addresses the challenge of managing information access to large knowledge bases by exploiting the resources identified earlier: the knowledge base, its experiential structure, the problem itself, and the recent history of task performance leading up to solving the problem. This approach is distinguished from existing memory approaches by its attempts to use these resources to improve efficiency while retaining generality. It is also distinguished by the particular methods it uses to exploit the relationship between representation, autonomy, precision, and efficiency.

## **2.4. Meeting the Desiderata**

The desiderata for memory for general intelligent agents presents challenges to traditional approaches, sometimes because of the tradeoffs existing approaches make between different areas of performance and functionality and sometimes because no existing approach fully addresses the desiderata.

Taken individually, each desideratum for memory systems for general intelligent agents applies constraints to one or more of these dimensions of evaluation. For example, storing a wide variety of information (desideratum 1) places requirements on the

representational power of memories for cognitive agents; interoperating with reasoning (desideratum 8) places requirements upon the functional autonomy of memory systems for cognitive agents. Taken collectively, the desiderata increase the requirements on memory systems by demanding that multiple constraints be met simultaneously by the same system.

The axes of functional, performance and technological evaluation identified earlier make it possible to make the desiderata much more explicit, which in turn makes it possible to examine existing approaches to memory retrieval in detail, determine how well they meet the desiderata, and identify where new approaches may be needed.

#### **2.4.1. Desideratum 1: Able to store a wide variety of information**

The foundation of memory retrieval is knowledge representation: the format for the knowledge that is stored and retrieved. Like the study of memory retrieval in humans, the study of knowledge representation in machines is a rich field, with areas of focus ranging from how to extend logical representations to handle complex human concepts to how to simplify complex representations to improve efficiency.

General cognitive agents place many demands on the knowledge representations of their memory systems: they need a language expressive enough to support the information needs of individual tasks, flexible enough to support the storage of knowledge that can be used across tasks, and organized enough to be efficiently accessed by the memory system.

The first requirement for a representation language for a general cognitive agent is that it must be expressive enough to make constructing individual tasks practical. The practical requirements of real world tasks may require the manipulation of literals or strings, scalars or vectors, or matrices or graph structures; therefore representation languages for general cognitive agents should be flexible enough to support a wide range of knowledge and datatypes or should enable the embedding or linking other data structures into the primary knowledge representation. This may take the form of “grounding” a symbolic representation in lower-level program language or perceptual concepts.

A representation language for a cognitive agent must also be flexible enough to apply across the set of tasks it performs. The knowledge needed for a particular task may be very different from that needed for other tasks, but all of these kinds of knowledge should be accessible through the same mechanism. A knowledge representation for a cognitive agent should have a highly expressive logical kernel.

Finally, a knowledge representation for a general cognitive agent must support the retrieval of that knowledge. While this is highly dependent on the method available to access that knowledge, the representation language itself can support the task of memory retrieval, either through the logical structure and organization of the knowledge base, through “hidden” indexes and structures which improve access, through annotation records on individual items to aid retrieval, or through some combination of the above techniques.



**Evaluating Approaches to Storing Knowledge.** The representational power of memory systems vary in complexity, ranging from simple strings of bits to structured representations intended to carry meaning. Simple caching algorithms which manipulate bits appear less expressive than database systems which manipulate tuples of typed data, which in turn are less expressive than case-based systems which have structured frame representations, all of which stand beneath the high-end richly interconnected networks of knowledge such as used in CYC or the Knowledge Machine.

*Representational Power.* This intuition about increasing complexity can be operationalized in terms of the *logical expressive power* of a representation. The simplest representation is a knowledge base filled with literals: single elements out of a universe of possibilities, like words, symbols or numbers. A more expressive representation composes literals of the same class into lists or arrays, like text documents composed of characters. Database systems use a richer representation of formatted tuples where each element is of a specific type. Tuples and attribute-value representations are rich enough to store predicates and thus have the logical representational power of first-order propositional logic (for discussion and examples see Sowa 1991, esp. Schubert 1991; also Russel & Norvig 1995 and Stefik 1995). Augmented first-order propositional logic representations, like simple frame systems and semantic networks, add additional constraints and or features, such as inheritance mechanisms or explicit interconnections between items in the knowledge base, but have basically the same logical representation power. Beyond these augmented first-order propositional logic representations, however,

lie more complex frame systems, semantic networks and other formalisms capable of representing arbitrary higher-order logical expressions (again see Sowa 1984 and Sowa 1991, especially Schubert 1991; also Rumelhart & Norman 1985), such as the frame/network/constraint language used in the CYC project (Lenat & Guha 1990, CYCORP 2000) and the CRYSTAL knowledge representation language discussed later in this dissertation. Finally, specialized systems exist whose sole purpose is to push the boundaries of logical representation, such as the conceptual graphs workbench PIERCE (Ellis & Levinson 1992).

*Support for Tasks.* Another way of judging a representation is how easily it supports task-specific information. A representation may be completely general, such as a database system which supports arbitrary records or a frame system which supports arbitrary knowledge. At the other end of the spectrum a representation may be tied to the task that its parent system is designed to solve — for example, items in a geographical database may contain privileged fields for representing GPS coordinates, while items in a visual dictionary may contain images. (In theory a task-specific element might be encodable into simple literals or predicates, but in practice this may be a highly nontrivial process.) Between these two extremes lie knowledge representation approaches which are general but contain optimizations which make it easy to construct representations that support specific tasks.

*Support for Memory Retrieval.* A third way to examine a knowledge representation is how well it supports memory-specific features. Again, a representation may be

completely general, such as a flat-file database or an unannotated list of predicates. Alternatively, a representation may contain elements specific to the task of memory retrieval — for example, privileged index or cross-reference fields, statistics on the item's usage history, and so forth.

**Existing Approaches.** The bulk of approaches to memory retrieval are task-specific: solutions targeted at specific classes of tasks with well-specified properties and attempt to satisfy the desiderata of their own individual task. Others are general purpose knowledge representation systems that attempt to represent knowledge for a wide range of tasks.

*Special-Purpose Memories.* Most AI systems have special-purpose memory modules which serve the information needs for the single task they perform. (Since almost all AI systems have *some* need for memory retrieval and most of them do not focus on the general problem of memory retrieval, it is not surprising that special-purpose mechanisms proliferate!)

The advantage of a task-specific memory system is that it can be designed to the task. Such a memory can, by design, enable an agent or system to store, recall and exploit the kinds of past experiences it needs to solve new problems, and can, also by design, contain special features which enable efficient and effective performance on the typical kinds of queries which the agent generates or encounters during its processing. Together, these two capabilities features enable a system to take advantage of its memory to effectively perform its task.

Examples of task-specific memories include the planning-specific case library of SPA (Hanks & Weld 1992, 1995) which operates by matching problems to variabilized cases; the more complex planning-specific case library found in PRODIGY/ANALOGY (Veloso 1995), which uses a variety of techniques, including plan footprints, goal indices and discrimination nets to find relevant cases efficiently; the topological models used for case storage and retrieval in ROUTER (Goel et al. 1994); and the knowledge sources of the KA system (Peterson et al. 1994, Goel et al. 1996) which adopts as an explicit hypothesis the use of several task-specific knowledge sources. This list could be indefinitely extended. For example, Kolodner (1993) reviews many systems and their knowledge retrieval strategies. Of the systems listed above, many are discussed in greater detail at other points in this thesis.

*General knowledge representations.* However, an important thrust within the knowledge representation field is the development of general-purpose representation languages sufficient to support a wide range of tasks. The most famous of these is perhaps CYC (Lenat & Guha 1990) but this family also includes the Knowledge Machine (Clark & Porter 1996, 1997) the KL-ONE and KODIAK family of representation languages (Brachman 1985, Wilensky 1986) as well as a variety of other approaches (e.g., Ellis & Levinson 1992, Genesereth & Fikes 1992, Shell & Carbonell 1989). General-purpose representation languages are founded on rich representations which can represent subtle concepts across a variety of domains and for a variety of uses, and

support a variety of means of accessing that knowledge to support the performance of a variety of tasks.

**Limits of existing approaches.** The disadvantage of the design-to-task approach is that specialized memory systems tailored to one task are in general limited to retrieving specific kinds of information in response to specific queries. Specialized memory systems cannot retrieve general information from a knowledge base. Even when the knowledge base contains a shared ontology and several classes of knowledge, the design-to-task approach often leads to partitioned knowledge bases and specialized memory retrieval mechanisms for each class of knowledge. This is useful from a performance perspective, means that individual systems are more difficult to expand and generalize, resulting in the addition of new memory subsystems as the tasks that top-level systems tackle become more complex. Furthermore, depending on the constraints of the task the design-to-task approach does not necessarily address the issues of controlling costs of retrieval or using available information to improve accuracy.

General purpose representation languages, however, are not general purpose memory retrieval systems either. Methods of access focus on the properties of pieces of knowledge or well-specified logical queries, rather than general memory retrieval based on sparse or incomplete information. While efficiency is an important issue for general-purpose representation languages, the focus of efficiency efforts is generally on finding algorithms that make logically accurate queries as fast as possible, rather than an explicit cost control policy that guarantees a result within a specific amount of time or effort.

Furthermore, general representation languages by their very nature focus on representation without addressing issues of processing, interaction with other tasks, or native memory retrieval support.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory addresses these limitations in a variety of ways. First and foremost, it employs a general knowledge representation, enabling it to be applied to a variety of tasks. (Technically, this is a property of experience-based agency; the context-sensitive asynchronous memory algorithms could be applied to a task-specific system). Second, provides rich support for task-specific features, building its knowledge representation on top of a general programming language. Third, it provides rich support for memory-specific features enabling a variety of memory retrieval algorithms and optimizations to be applied.

#### **2.4.2. Desideratum 2: Provides a general method for accessing information**

Representations are only as useful as the algorithms which allow access to them. The access algorithms which enable reasoning tasks to interact with a knowledge base should be as general as the knowledge base itself. This need for generality applies not only to the basic access methods that enable processes to inspect and manipulate individual pieces of knowledge in working memory, but also the memory retrieval processes that bring information from the long term store.

A memory retrieval algorithm for a general cognitive agent should support a very broad query language — although this language need not be arbitrarily complex. The search mechanisms which search this store of knowledge should be comprehensive enough to find a good set of potential items even if the question asked is vague or poorly specified. And the algorithms which match individual items to the questions should be powerful and efficient, capable of matching the query language to arbitrary pieces of knowledge.

**Evaluation Metrics.** There are a number of ways to evaluate methods to access information; these questions parallel our earlier discussion of knowledge representation properties. These metrics include the language queries can be written in, the algorithm for searching the knowledge base, and the algorithm for matching items to queries. A general representation language is an important component of a memory retrieval system for a general cognitive agent, but the architecture must specify more than just a language: it needs to specify a query language and memory retrieval mechanism which enables an agent to use the kinds of information it has on hand to find relevant information in its knowledge base. An internal representation is only as rich (or as poor) as the algorithms that access it. A record in a flat-file database may store knowledge of incredible richness — a passage from Feynman, the Torah, or Aristotle’s *Poetics*. Conversely, an expressive representation is only significant to the degree that it is exploited by the algorithms that query and search the store and by the algorithms that match items found in the store against the query.

*Query Language.* What kinds of questions can be asked of the store? Again, there is a hierarchy: simple memory caching systems only allow a query of an address; a database system permits a richer matching of fields or of keywords. Artificial intelligence systems typically enable matching based on variable unification or similarity. Alternatively, a query may not resemble a simple question, instead corresponding to reminding based on meaning, associations with working memory, or other content-addressable approaches.

It is somewhat more difficult to rank approaches to query languages. The simplest are perhaps keyword searches, which can be augmented with Boolean search operations. The next stage includes logical and mathematical expressions such as that used by the SQL language. Russel (1996) illustrates a series of increasingly capable of matching and unification languages for AI systems. Kolodner (1993) lists a variety of approaches for similarity-based matching of cases. Like representation languages, query languages may contain task-specific elements, such as searches for synonyms or adjacent words in text retrieval or searches by location in a geographical database; query languages may also contain memory-specific elements, such as a request for the most frequently retrieved item or for a count of all relevant items in the memory. Perhaps the ultimate query language would be asking questions in a natural language like English; however, this brings in a host of issues in natural language processing which are beyond the scope of this thesis and hence we will not discuss this possibility further here.

An example of the interaction between representation and query language is the ABBY system, which uses a rich query language to access simply represented store



(Domeshek 1993). ABBY's cases are flat text representations annotated with memory-specific indices. The indices themselves are simple vectors. However, ABBY uses a rich query representation called the Universal Index Frame which enables it to make very 'deep' reminders.

This naturally leads us to the flip side of "What questions can we ask?", namely, "How are those questions answered?" Logically, this question can be split into three parts: how the store is organized, how it is "searched" in response to a question and how the items in store are matched to the question.

*Knowledge Organization.* Knowledge organization can make access to knowledge easier or more difficult; as an example there is the typical time-space tradeoff between parsimonious representations, which may be minimal in size but require linear or logarithmic access times, and indexed representations, which may support constant or near-constant time access but require large amounts of overhead. As knowledge structures and needs for access become more complex, it is possible to build additional structural support into a knowledge base to enable fast access of the right knowledge at the right time. For example, when constructing a case library, if certain features are known in advance to be good labels for cases they can be incorporated directly into the structure of the representation to aid easier access; as another example the PROTOS system (Bareiss et al. 1988) incorporated "difference links" to enable the memory retrieval system to quickly narrow in on a correct case after it had found a near miss. Issues that must be addressed include indexing (the structure of the knowledge base),

labeling (the content of indices) and processing (how those indices and labels are manipulated). The issue of knowledge organization is discussed further in (Schank 1982, Kolodner 1983, 1993, Hendler 1989, Domeshek 1992).

*Search Algorithm.* A wide variety of algorithms exist to search a memory system's knowledge store. When queried, a memory system may search its store exhaustively or partially. The order and range of a partial search may be determined dynamically or may be preordained from the start. And regardless of whether the search is exhaustive or not, a memory may choose a variety of search orders for its trek through the knowledge base. Several metrics are required to capture these intuitions: search comprehensiveness, search partitioning, and search strategy.

Comprehensiveness refers to the completeness with which a memory system searches its store; a memory search may be exhaustively comprehensive (searches everything), potentially comprehensive (searches a subset, which potentially could range over the whole knowledge base) or non-comprehensive (searches are deliberately restricted to portions of the knowledge base). Note that an exhaustive search may not be a "search" at all in the case of a content-addressable store, and the "non-comprehensive" search is not relevant for a fixed-address memory.

Search partitioning and search strategy are related. Partitioning refers to the way that the subset of a knowledge base searched in non-exhaustive search is determined. Partitioning can be done statically (at knowledge base creation or update time) or

dynamically (at query time). Search strategy refers to the order in which items are searched; this may be a fixed-order “brute force” search or may be dynamically determined by the properties of the query. Generally partitioning strategies and search strategies are complementary; as one example, consider a database system which for efficiency sorts its contents into separate subsets based on predefined categories known at design time; as another, consider a semantic memory which uses a relatedness measure, embodied in a spreading activation mechanism, to conduct its search.

The result of search is the identification of candidate retrieval items: the next logical step is matching those candidates to the query.

*Matching Language.* Matching is the process of taking a knowledge item, however it is found, and judging its suitability as an answer to a query. Even some fixed-address or content-addressable memory approaches which have no real “search” process may nonetheless do something like “matching.” For example, the MAC/FAC retrieval system (Gentner & Forbus 1991) does not “search” its memory; instead, when it is presented with a target for retrieval, it compares the target with each source stored in memory using a fast similarity test based on a dot product of content vectors (Many Are Called) and returns the most highly matching candidates (Few Are Chosen).

Like query languages, matching languages enable the user of a retrieval system to specify a piece of knowledge; unlike query languages, matching languages do not specify how to find knowledge but simply how to evaluate an individual piece of knowledge’s

suitability. In general, a system's matching language is closely tied to its retrieval language, although the specification for matching may be decoupled from the specification for search and hence may be viewed as more or less expressive than the query language as a whole. Thus, matching and search are closely related (and may be blurred): while search may simply collect candidate items for matching, a "smart" search could incorporate matching or indexing steps to ensure that the candidates returned match the search criteria as closely as possible.

Hence the expressiveness of a matching algorithm parallels the expressiveness of query languages. Again, the simplest are perhaps keyword matching languages and their Boolean, logical, and mathematical extensions. Russel's (1996) taxonomy of matching and unification languages is equally relevant here, as is Kolodner's (1993) similarity taxonomy. Kolodner (1988) illustrates a complex case-matching language. Some of the most complex matching algorithms include structure matching for conceptual graph structures (for a discussion of conceptual graph systems, see Sowa 1984, Ellis & Levinson 1992, Levinson & Ellis 1993). Like query and representation languages, matching languages can judge retrievals based on task-specific criteria, such as a visual matching algorithm for candidate retrieved images. Matching languages may treat a query as a logical test, returning all items which pass the test, or may view it as a ranking of similarity function where only the "best" or " $n$  best" items are returned.

**Existing approaches.** Three approaches that focus on general query languages are case-based and dynamic memories, common knowledge bases, and performance memory systems.

*Case-based and dynamic memories.* A more general-purpose approach to the problem of memory retrieval can be found in dynamic memory and its successors in general case-based reasoning methods. Case-based memories are methods for information retrieval inspired by models of human memory and applicable to multiple tasks. Examples include systems like CYRUS (Kolodner 1983), PROTOS (Bareiss et al. 1988) and ABBY (Domeshek 1992), which of course performed specific tasks but which introduced general memory methods such as redundant discrimination networks (CYRUS), discrimination links (PROTOS) and universal index frames (ABBY) as general methods to organize knowledge for easy access to memory.

General case-based memories explicitly considered more general methods, not limited to a single performance task, for agents to exploit past experiences (e.g., CYRUS, PROTOS) and addressed other related issues such as strategic retrieval and knowledge organization.

However, case-based memories generally focus on storing and retrieving episode-sized chunks of knowledge, not models of general semantic retrieval. Many case-based memories are designed in ways to make them as efficient as possible, but do not operate under any specific cost control policy. CYRUS explicitly addressed the issue of strategic

interaction with ongoing reasoning tasks to refine retrieval, but does not define a protocol for interaction between the ongoing memory retrieval process and simultaneously active task processing or environmental events.

To build upon the lessons of general case-based and dynamic memories, we must generalize their principles to support storage and retrieval of general knowledge, not just episode-sized chunks, must incorporate more comprehensive mechanisms for controlling cost of retrieval, and must specify how these systems will interact with ongoing reasoning or environmental action.

*Common knowledge bases and performance memory systems.* Discussed in greater detail in the next section, both common knowledge bases and performance memory systems attempt to provide support for the memory retrieval task in a variety of contexts. Therefore, even when their focus may not be on the memory retrieval language itself, they nonetheless must provide some general memory retrieval language to access their memory support.

**Limits of Existing Approaches.** These approaches provide more general methods for agents to exploit past experiences not dependent on a single performance task, and in the case of case-based and dynamic memories begin to address some issues in strategic retrieval and knowledge organization. However, case-based and dynamic memories are still focused on storing and retrieving only episodes or cases, have limited strategic cost policy, and in general do not deal with integration with ongoing reasoning processes.

Memories for cognitive agents need a more general approach to storage and retrieval, a comprehensive cost management policy, and a policy for interacting with ongoing reasoning tasks.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory, working in concert with experience-based agency, contains several features which enable it to provide a general method for information access. First, as mentioned in the last section it employs a general knowledge representation; this is augmented with a domain-independent query language and a domain-independent memory search strategy. Second, the experience-based agency theory builds on top of this foundation a general-purpose inter-task communications language and general-purpose task control mechanisms, enabling the context-sensitive asynchronous memory algorithms to integrate well with a variety of tasks.

### **2.4.3. Desideratum 3: Scales to large multifunction knowledge bases**

A cognitive agent needs knowledge representation, storage and retrieval schemes which can handle large amounts of knowledge. General cognitive agents may perform dozens of tasks using the same knowledge or may have thousands of items of knowledge specific to a particular task, and hence require knowledge bases that can scale far beyond toy problems.

**Evaluation metrics.** Scalability is a question of computational complexity: what impact does increasing a size parameter of a knowledge base have on the cost of

knowledge base operations? Several parameters are relevant: the absolute size of the knowledge base, the absolute size of any individual item (or, in a semantic network, the number of connections between one item and another item), the number of items in any individual class, and the number of classes or sets of items

As these parameters are increased, a scalable knowledge base must control its costs in a variety of areas. Storage of additional information must remain space efficient as items and connections increase. Direct access to the logical content of an individual piece of knowledge must remain time efficient as well. Ideally, the cost of searching the knowledge base, matching individual candidate items, and overall retrieval should remain efficient.

While constant-time scalability may not always be possible, exponential scalability is clearly unacceptable. For knowledge base parameters which may become large but are not likely to grow without bound, such as the size of individual items or the complexity of matching specifications, linear or polynomial performance are probably acceptable. For parameters which do grow without any effective bound, such as absolute knowledge base size, linear or sublinear scalability is ideal.

Another important issue is the generality of a system's memory retrieval mechanisms: can it be used effectively for a variety of tasks? Like representation and query languages, the processing algorithms of memory may be designed to specifically support a small set of tasks. At the one extreme might be a database which only supported retrieving book



listings by author or ISBN and which structured its representations, query languages and processes specifically to support only those series of tasks. A more general system might be something like an SQL database, which has optimized representations supporting a narrowly-specified range of functionality, but which can be nevertheless be used in service of many tasks. At the other extreme would be a full general-purpose memory system whose representations and processes are optimized not to serve any one task but instead to provide as wide a set of features as possible so that the single uniform memory system it provides can support a wide range of tasks and processes.

**Existing approaches.** A variety of research traditions have tackled the problem of large multifunction knowledge bases in various guises. These include what this research terms *performance memory systems* that try to produce very fast knowledge bases for very large sets of data, *common knowledge bases* that serve as single repository of knowledge for many different tasks, and *cognitive architectures* which serve as standard platforms for constructing a wide range of tasks.

*Performance Memory Systems.* At the opposite end of the spectrum from task-specific memory systems are “high-performance” memory retrieval architectures, which address directly the general problem of memory retrieval for large knowledge bases. The explicit hypothesis these systems explore is that the key to fast, effective information retrieval is powerful parallel hardware. Examples include PARADYME (Kolodner 1988) and PARKA/CAPER (Kettler et al. 1993), as well as the memory architecture behind the ABBY system (Domeshek 1992).

Performance memory approaches by their very nature tend to focus on the general problems of memory retrieval rather than needs of specific tasks. They make a comprehensive effort to limit costs and achieve good performance, and often incorporate very general query languages enabling the approaches to be applied to many different tasks.

While experimental results confirm that system such as PARKA have achieved stunning memory retrieval performance, whether parallel hardware will continue to outpace knowledge base size is an open question, both in general and for the specific amount of horsepower available to any given application. Even assuming that parallel processing power will eventually outpace knowledge base size (perhaps when hardware becomes so advanced that the difference between a processing unit and a memory unit is no longer meaningful), a number of questions will remain. First, does the memory use that processing power as effectively as possible, exploiting all available knowledge to answer questions as efficiently as possible to free processing power for higher level-tasks; second, how does the memory cope with poorly specified questions which do not give the memory enough information to uniquely specify a response; and third, given that memory retrieval will continue to take a finite amount of time (at least for the near future) how does the memory system interact with or respond to ongoing retrieval and environmental processes.

While performance memory systems as a whole make a comprehensive effort to limit costs and achieve good performance, and include general query languages for knowledge

access, they nonetheless still have limitations with respect to our desiderata. These include not exploiting information from the reasoning task or the environment to improve precision of search, and not fully addressing the problem of interacting with the other components of a functioning agent in an environment. And to some extent these parallel systems are brute-force, brawn over brains approaches (not to allege that considerable brains are not involved). The potential exists to expend search effort more effectively depending on context of a query. Also, because retrieval still takes finite time, parallel systems could possibly benefit from a policy for interacting with ongoing reasoning processes, enabling them to exploit additional time when that becomes available. After all, knowledge base size and/or resource constraints often outrace parallel horsepower; for example, even though the AltaVista search engine uses massive computing clusters redundantly distributed across the globe to provide fast lookup of web sites (Ray et al. 1998, p279), it nonetheless has to limit the amount of effort it expends on queries to handle both the bulk and the load of the Web simultaneously. For this reason, the fabled “found 200,000 hits” count of results returned by AltaVista is usually only an approximation (ibid., p. 77). Therefore, it is worth considering approaches to more efficiently use resources or available time could help even a parallel system cope with increased knowledge base size or load.

*Large Knowledge Bases.* Some of the research discussed earlier in knowledge representation also tackled the issue of common knowledge bases for multiple tasks, including systems such as CYC (Lenat & Guha 1990), the Knowledge Machine as used

in the Botany Knowledge Base project (Clark & Porter 1996), and PIERCE (Ellis & Levinson 1992). To adequately represent complex concepts, detailed plans and linguistic structures for a variety of areas, these systems required powerful representational schemes capable of representing complex conceptual relationships and higher-order logical expressions; CYC and PIERCE also employed groundings between those expressions and features of the “real world” outside the representation (Levinson & Ellis 1993).

The limitations of common knowledge bases with respect to our desiderata are that they do not explicitly address issues such as performance or exploiting information from the reasoning task or the environment to improve precision of search.

*Cognitive architectures.* The study of human memory is not limited to the study of human memory in isolation. Cognitive architectures are attempts to model the entire human cognitive process. The two most famous of these approaches are Soar (Newell 1990) and ACT (Anderson 1983).

In tackling the general problem of cognition, systems like Soar and ACT also necessarily address the issue of memory retrieval. Here, the fact that many memory tasks share similar properties, such as the power law of performance, is often the starting point for deriving a clean, general architecture for memory retrieval and/or reasoning which exhibits those properties as a natural consequence of fundamental design decisions. Because cognitive architectures are designed to be applied to a wide variety of tasks,

their representation languages and retrieval mechanisms are designed to be general. Furthermore, the way in which memory retrieval interacts with other processes within the agent is carefully delineated as part of the theory.

However, cognitive architectures have some limitations when viewed as models for artificial systems. Like other computational models of memory, they focus on modeling specific theoretical properties rather than the overall performance features of the architecture. For example, the Soar architecture makes explicit theoretical commitments about the semantics of each stage of the decision cycle and as a consequence does not directly control the cost each cycle. For example, the elaboration phase of each decision cycle is conceptualized as the gathering of all information the agent has available to it; as a consequence all productions which can match do so, often in sequence (see Newell 1990, pp.170-171). Because this process is not limited in any way the cost of an elaboration cycle could potentially become arbitrarily expensive. Of course, this is not relevant for Soar's use as a cognitive model of reasoning tasks but does become relevant when Soar is used as an artificial intelligence system. Aware of this issue, the Soar group has explicitly tackled the problem of cost as a research topic, investigating the areas in which increased knowledge could lead to unacceptable costs and, when necessary, designing strategies to improve the performance of the system where those costs arise (e.g., Tambe et al. 1990, Tambe et al. 1992, Doorenbos 1993, Tambe & Rosenbloom 1994).

Furthermore, each cognitive architecture brings to the table specific assumptions about the memory retrieval process which arguably make them less appropriate as architectures for artificial agents. Each cognitive architecture addresses a certain scope. For example Soar focuses on the so-called “cognitive band” of mental operations ranging in time from  $\sim 10\text{ms}$  to  $\sim 10\text{sec}$  (Newell 1990 Chapter 3) and ACT\* (Anderson 1983) has a similar range. As a result, cognitive architectures do not address features that arise outside of that scope, such as strategic memory retrieval or self-monitoring needed for long-term tasks such as academic studies, invention or design.

Each cognitive architecture supports memory retrieval processes derived from the basic theoretical assumptions of the theory. For example, Soar has no model of declarative memory retrieval: all memory retrieval in Soar is procedural based on productions or “chunks”; this architecture is effective at modeling skill acquisition but makes it difficult to model priming of memory retrieval or learning of large chunks of knowledge like cases.<sup>5</sup>

---

<sup>5</sup> This is a variant of the data chunking problem (Simon, T. 1994, personal communication). Learning declarative facts in Soar requires learning productions which map from the features to be indexed to the features to be retrieved in a process called data chunking. In data chunking, the input index is fed into a substate which generates the target result in a combinatorial process and then “hides” the details of construction to enable a production to be learned. In an unpublished feasibility study (Moorman & Francis 1994) conducted under the direction of Tony Simon, a case-based system called the Captain’s Advisory Tool (CAT) was ported to the Soar architecture to demonstrate the difficulty of learning new cases. The study showed that the procedural logic of CAT could be readily translated into the Soar framework but that

Each cognitive architecture embodies a specific model of processing which is derived from the basic theoretical assumptions of the theory. For example, memory retrieval and processing in Soar are explicitly interleaved as part of Soar's decision cycle, whereas in ACT semantic memory retrieval operates in parallel with production firing. ACT thus can model the interaction of memory and ongoing reasoning processes, but only processes which are implemented within the ACT processing model. As another example, Soar's architecture makes an explicit commitment to pulling in all available information as part of the elaboration phase of each reasoning cycle; for this reason, it is difficult to see how Soar could model the priming effects observed in human memory retrieval (e.g., Meyer & Schvaneveldt 1971), though Newell (1990) does identify priming as an outstanding research area for Soar and takes a first stab at possible solutions. Soar also has no cost management policy, though it is a long standing goal of the Soar group

---

the data chunking of CAT cases would result in an expensive combinatorial explosion. This was partially caused by limits in data chunking itself; after our study was conducted Rogers (1997) developed techniques to make data chunking more efficient involving a two-stage chunking process using an additional recursive substate (also see the tutorial in Hastings 1997). Despite this technique, a second combinatorial explosion arises because of the nature of data chunks themselves. To enable cases to be flexibly retrieved, CAT used multiple index networks to store cases. However, since each data chunk in Soar is a complete source to target mapping, each potential retrieval pathway in CAT's index network must be learned separately as a unique data chunk in Soar. For these reasons, learning large concepts which may be retrieved in a variety of circumstances still requires more effort in Soar than in frameworks which natively provide features to store declarative representations. The value of being able to perform this kind of flexible retrieval is, of course, one of the open points of debate between the Soar and CBR communities.

to continue to improve the system's performance (e.g., Tambe et al. 1990, Tambe et al. 1992, Doorenbos 1993, Tambe & Rosenbloom 1994).

**Limitations of Existing Approaches.** Cognitive architectures serve as a good model for architectures for memory retrieval for artificial agents with the following caveats. They specify a general memory retrieval language and how it interacts with reasoning; however, memory retrieval in an artificial agent architecture must be general enough to support the full variety of tasks the memory will be used for, rather than the specific kinds of memory retrieval that fall within the scope of an architecture. Similar comments hold for more focused models of human memory retrieval: for artificial intelligence systems to benefit from the useful properties of these models, the features of these models need to be embedded in a useful functional interface even though they might fall outside the carefully targeted scope of a model human architecture. Finally, because AI systems may integrate a variety of components with a variety of properties, an agent architecture for memory retrieval should have the capacity to interact with a variety of other processes, not just processes implemented within the architecture's framework.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory (working in concert with experience-based agency) addresses the issue of scaling to large, large multifunction knowledge bases in the following ways. It employs general representation and memory retrieval methods to enable supporting multiple tasks within the same knowledge base, employs modified spreading activation process to cope with increasing fan-outs found in large



multifunction knowledge bases, and employs a variety of techniques to make tradeoffs between cost of retrieval, accuracy of retrieval and available information to aid search of large knowledge bases.

#### **2.4.4. Desideratum 4: Manages cost of retrieval**

More important than pure efficiency is cost management. Unless a memory is infinitely efficient and powerful, resource limits may always put the squeeze on the cost of retrieval. Some tasks performed by agents need immediate response of some form. For these kinds of agents, it is important to have a memory retrieval system which can bound the cost of retrieval even when the costs incurred in searching large knowledge bases or matching complex specifications would normally exceed the available resources.

**Evaluation Metrics.** The two most important metrics for this desiderata are the raw computational complexity of storage and retrieval and the presence of cost control metrics to limit computational complexity in circumstances where it may run out of control.

Computational complexity addresses the expected time and space costs of storage — how long does it take to add a new piece of information to the knowledge base and how many bits does it take? What are the (expected) time and space costs of retrieval — how quickly can an item be retrieved and how much information must be stored while retrieval is occurring?

Cost control policies recognize that different tasks and domains may present unusual circumstances to a memory retrieval system — thousands of items in one category, nodes with unusually large fanout, extremely complex retrieval specifications, and so on — that may overwhelm even a system with a good computational complexity profile. A cost control policy attempts to ensure that even when these unusual circumstances occur, the cost of memory retrieval is controlled to the point that the agent can still successfully benefit from memory retrieval and complete its task.

**Existing Approaches.** Many task specific memory systems, especially those for planning systems that learn, focus on ways of reducing the cost of retrieval innately and controlling the cost of retrieval in unusual circumstances. Some systems incorporate a variety of techniques which monitor the utility of knowledge in an attempt to control costs, such as incremental retrieval in PRODIGY/ANALOGY (Veloso 1995) and knowledge base winnowing in Minton's system (Minton 1988, 1990); other systems, such as ROUTER, incorporate efficiency techniques such as knowledge organization, but hold their properties constant in an attempt to analyze the tradeoffs inherent in memory retrieval (Goel et al. 1994)

**Limitations of existing approaches.** Most systems do not address the issue of managing cost of retrieval at all. This may be because the systems did not target tasks with large knowledge bases. For example, compare the retrieval system for the comparatively small case library of SPA (Hanks & Weld 1992, 1995) with the much more complex case library of complex case library of PRODIGY/ANALOGY (Veloso

1995). SPA's case library simply matches the input goals of a problem against variabilized cases in a library and returns the best matching one, whereas PRODIGY/ANALOGY uses redundant discrimination networks and a variety of other techniques to focus search effort on the most relevant cases. Another reason for not managing the cost of retrieval may be because the systems targeted accurate models of specific psychological phenomena in favor of performance considerations; for example, compare the broad access of memory in MAC/FAC (Gentner & Forbus 1991, Law et al. 1994) and ARCS (Thagard et al. 1990) with the more limited searches employed in EUREKA (Jones & Langley 1995) and REMIND (Lange & Wharton 1994). MAC/FAC and ARCS focus on modeling analogical memory retrieval phenomena and use comprehensive examination (MAC/FAC) or extensive searches of memory (ARCS) to find all information relevant to analogical transfer. EUREKA and REMIND instead focus on modeling memory retrieval in service of problem solving and language understanding tasks with an explicit commitment to model limited reasoning and retrieval processes.

Systems that do contain cost controls generally drive those controls based on specific utility metrics related to the single task the system performs and the expected utility of retrieved items. For example, Minton's work in PRODIGY-EBL (1988, 1990) uses a utility metric specific to speedup on the planning task. Most systems with cost controls do try to optimize retrieval based on the information in the problem, but none of the

systems just discussed attempt to exploit additional information about the task or the demands of current situation to tune retrieval.

In general, designing a system to effectively answer a single kind of query does not address the issue of the memory system interacting with other ongoing reasoning tasks or environmental situations. A task-specific memory system is often “blind” to the rest of the agent — its search algorithms are generally limited to responding to specific queries related to the task, whether or not information is available in the environment which would help answer the question. Similarly, a task-specific system’s cost controls focus on utility metrics relevant to the task without considering the utility of knowledge in the context of the broader needs of the agent. Experience with human cognitive agents suggests that they do not suffer from these limitations (e.g., Meyer & Schvaneveldt 1971, Collins & Loftus 1975, Klimesch 1994, Kolodner & Wills 1993a,b; Wills & Kolodner 1994).

Ideally, a memory system for a cognitive agent would incorporate as many of the positive features of existing task-specific memory systems as possible — effectiveness at retrieval, efficiency, cost control with respect to available utility metrics, and so on — without the limitations. A memory for a cognitive agent should be able to answer general queries across the agent’s whole knowledge, should incorporate general cost control policies that take into account all activity in the agent or its environment, and should exploit whatever additional information is available in the agent or environment to improve the quality of retrieval.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory (working in concert with experience-based agency) addresses the issue of managing cost of retrieval in the following ways. It employs a spreading activation process, which limits the amount of effort expended in memory search; it employs heuristic strategies for controlling cost of memory retrieval request processing; and it employs additional heuristic cost-control strategies to limit costs of memory search.

#### **2.4.5. Desideratum 5: Preserves accuracy of retrieval (in the face of resource limits)**

While cost should be controlled, it should not be at the expense of quality. Even if resource limits are extremely tight, it is a rare task that can continue to successfully function if memory returns pure garbage.

As a consequence of coping strategies for resource limits, comprehensiveness of knowledge base search may suffer. While measures such as recall and quantity of retrieval will then degrade, accuracy of retrieval — ensuring that the best matching items are retrieved — should be preserved as much as possible. As search of the knowledge base becomes less and less comprehensive to cope with increased size and limited resources, precision of search becomes ever more critical for preserving accuracy.

**Evaluation metrics.** There are a variety of ways of evaluating the quality of retrieval, including comprehensiveness of search, precision of search, retrieval guarantees,

exactness of matching, determinism of the retrieval process, and recall, precision, accuracy and quantity of responses to a query.

*Comprehensiveness of search.* Does the search of memory cover the entire knowledge base? Comprehensiveness is a strength in that the memory will return a relevant item if found; it is a weakness in that if only a few items are relevant most of the search effort will be wasted.

*Precision of search.* How much of the search effort is spent on relevant items? For example, an exhaustive search of an ordered list for an item is highly imprecise (on the average only  $O(1/n)$  of the search effort is expended on the right item), whereas a binary search on the same list is far more precise (where only  $O(1/\log n)$  of the effort is “wasted”).

*Guarantee of retrieval.* Is the search guaranteed to return a matching item if one exists? A comprehensive search will, and under the right conditions a precise limited search algorithm will; but most non-comprehensive algorithms cannot guarantee that a matching item will be found if one exists.

*Exactness of matching.* Matching algorithms may be exact, in which only an item which precisely matches the specifications of the input query will be returned, or approximate, in which a “close” match to a specification, as measured by some ranking function, will count as “matching” the specifications of the query. This “close” match

may be chosen on the basis of some threshold for degree of match, or may simply correspond to the best or  $n$  best close matches found.

*Determinism.* Presuming that no new items are added to the memory in the interim, will the memory return the same response to a query every time? A memory retrieval system which specified a strict ordering of search and matching strategy would be may be perfectly deterministic. A non-deterministic memory, in contrast, might incorporate some random element in search, or a priming strategy which changed the weights of items in the knowledge base, or might learn the matching weights of items based on feedback from previous queries.

*Recall, precision, accuracy and quantity of responses to a query.* The set of items a memory retrieval system returns can be evaluated two ways. Its precision is the proportion of items that it returns which actually match the specifications of the query; its recall is the proportion of matching items that it returns out of all the possible matching items in the knowledge base. Similar measures include the accuracy of retrieval — does the memory return the best matching items in the knowledge base — and the quantity of recall — what is the absolute count of items that the memory returns, ignoring how many it missed?

**Existing approaches.** Evaluation of most existing memory retrieval systems focus on pure speed and correctness of logical matches, without handling fuzzier quantities like exactness of matching or precision of retrieval.

These issues have been dealt with more extensively in the Web (Cheong 1996) and information retrieval fields (Jones & Willet 1997). One finding which arises again and again in a variety of contexts is that simple approaches, such as the vector similarity model, can get approximately correct results of reasonably high quality; furthermore, improving upon these approximate results is difficult with general approaches, requiring domain knowledge and/or considerable additional computational efforts (see discussion in Baeza-Yates & Riberio-Neto 1999).

**Limitations of existing approaches.** While several existing memory retrieval systems, such as REMIND (Lange & Wharton 1994), allow approximate matches, evaluations of these systems have focused on cognitive modeling and not on preserving the accuracy of retrieval under varying resource conditions.

While information retrieval approaches have addressed quality and efficiency issues, they generally map simple queries (from an AI perspective) to relatively unstructured documents and would not form the foundation for an AI memory retrieval system.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory (working in concert with experience-based agency) attempts to address these issues more directly. It employs a modified spreading activation algorithm which attempts to make memory search more accurate based on context, and employs an incremental/anytime processing strategy which can attempt to find better and better retrievals over time.



#### **2.4.6. Desideratum 6: Exploits task/environmental information**

To improve the precision of search and preserve the accuracy of retrieval, a memory system should use all information available to it. If the questions asked of a memory system contain all of the information available, then a memory system must make do with that; however, there is often a lot of information available in the environment or in the task state which goes beyond the specifics of a question that is asked.

The only system in an agent which can fully specify the information needs of a reasoning task is the reasoning task itself. Conversely, the only system in an agent which can know whether an arbitrary piece of information is relevant to finding items in memory relevant to an information retrieval specification is the memory retrieval system itself. Therefore, a memory system should exploit any information in the task or environmental context that it finds useful for improving the precision of its search.

Exploiting information goes beyond merely improving the precision of search, however; if specific information becomes available to the reasoner which could be used to update the specifications of a memory retrieval request the memory should exploit it to narrow the focus of its search or improve its matching. This requires a more elaborate interaction between reasoning and memory, discussed in detail in desiderata 8 and 9.

**Evaluation metrics.** Obviously one of the most important issues in evaluating a memory system's ability to exploit contextual information is the support it has for using contextual information. Only slightly less important is the style with which the memory

operates — is it configured in such a way that it can actually use information from the environment? In other words, how autonomous — capable of working independently of other tasks and/or the environment, and thus capable of exploiting information from those sources — is the memory?

The “autonomy” of a memory system can be judged with respect to three axes: whether memory retrieval is “memoryless” or maintains an independent state, whether memory retrieval is a synchronous or asynchronous process, and whether memory must be polled or can return responses without an explicit query. Essentially, does memory have a memory of what it has been doing? Is memory a subroutine that is called in response to a query, or is it an independently running process or coroutine that operates in parallel with (or interleaved with) other processes or threads? Does it return information when only when queried or polled, or does it have facilities to autonomously signal the process which requested the information?

Does memory have a memory of the queries it has seen? Most memory retrieval algorithms are implemented like subroutines, which do not store a memory of past queries: the subroutine receives a query and control of the process, processes the query, and exits, returning a response. Examples of this approach include the memory retrieval routines in PRODIGY/ANALOGY (1994) and SPA (Hanks & Weld 1992, 1995). An alternative formulation views memory as a coroutine which maintains its own state and alternates control with a reasoning coroutine. The dynamic memory approach (Schank 1982) views memory as a dynamic process in which parts of the reasoning process occur

in the memory and modify it *in situ*. Finally, memory may be viewed as a self-contained object, opaque to the reasoner, with a history, state and algorithms hidden behind the messages and responses by which queries are processed. From this perspective, the memory's patterns of behavior become most important — does it exhibit recency, priming or frequency effects like human memory? Can it explicitly reason about the queries it has been asked? From an efficiency perspective, the issue of “the memory of memory” is most significant if multiple queries may repeat some of the same work.

A closely related issue is whether memory can be queried synchronously or asynchronously. When a synchronous message is sent between two processes, the calling process must block and wait for the called process to return, as in the case of a subroutine or co-routine. An asynchronous response, in contrast, may be sent at any time, and the calling process may choose to either wait or continue processing. Most memory retrieval systems are synchronous, implemented as or like subroutines; for example, when you submit a search to Yahoo, you have to sit back and wait for it return a set of hits. However, asynchronous operation might make sense if the reasoning task has something it *needs* to do (such as real time performance), if the cost of memory retrieval is high and the reasoning task has something that it *can* do, or if extra processing power is available and the reasoning task has something *useful* to do.

This raises the issue of how results are returned from memory. The simplest sequence of messages is query-response, as used in subroutines and synchronously running processes. When asynchronous processes are introduced, a number of other

options become available, including query-poll-response and query-alert. In the query-poll-response strategy, memory accepts a query and does not return a response until it is explicitly polled by the reasoning task. In the query-alert strategy, memory accepts a query and proactively alerts the reasoner when a retrieval is found. These strategies may be combined in a variety of ways.

**Existing Approaches.** The primary source of models which exploit task and environmental information are in psychological models of memory and advanced artificial intelligence models of complex cognitive phenomena.

*Psychological Models of Memory.* The approaches to memory retrieval we have discussed so far have examined memory as it serves the needs of artificial agents. An alternative though not incompatible approach is to model the memory systems found in natural agents, namely humans. Computational models of memory have a rich history in cognitive psychology and artificial intelligence, and it is not possible to summarize all of that tradition in a few sentences. Painting with a broad brush, however, computational models of memory retrieval attempt to model specific properties of human memory retrieval, such as the power law of performance or the fan-out effect (examples include John Anderson's ACT series (Anderson 1983, 1990), or to enable specific capabilities not found in existing computer memory systems, such as cross-domain reminders (as found in the KDSA approach (Wolverton 1994)). In helping to illuminate interesting properties of human memory, computational models of memory retrieval often

incorporate features not present in typical AI systems, such as priming, forgetting, and reconstructive memory.

Psychological models of memory attempt to recreate in computational systems specific human memory phenomena, pointing the way to achieving new capabilities in more general artificial intelligence systems. Excluding cognitive architectures such as ACT and Soar discussed earlier, the most relevant of these computational models of human memory are spreading activation retrieval approaches such as REMIND (Lange & Wharton 1994), and the Connectivity Model (Klimesch 1994), as well as approaches that apply spreading activation to other tasks, including problem solving in EUREKA (Jones 1989, 1993; Jones & Langley 1995), cross-domain analogies in KDSA (Wolverton 1994), metaphor in Sapper (Veale 1995), and others including general cognitive architectures such as ACT\* (Anderson 1983).

Spreading activation fulfills at least two distinct roles in these memory systems: *selective activation* of a small but potentially relevant set of candidates in memory using threshold-limited spreading activation (Jones & Langley 1995), and *evidential activation* or ranking which determines the relative relevance of items in the candidate set (Lange & Wharton 1994). For example, the REMIND system uses spreading activation to selectively restrict its view to a portion knowledge base believed to be relevant and uses the same activation as evidence of what item is most relevant. Similarly, the Knowledge Directed Spreading Activation approach (Wolverton 1994) uses spreading activation to combat combinatorial explosions in retrieval and uses the activation of concepts as

evidence that items should be retrieved for consideration as “beacons” to serve as new starting points for spreading activation. The Sapper system also uses selective activation to limit retrieval effort and evidential activation to select concepts, this time to determine what new analogical bridges to create. Models such as ACT\* and the Connectivity Model use spreading activation in a slightly different way, viewing the activation of a node to determine how quickly productions or items of declarative memory become active.

Similar to spreading activation approaches are marker passing approaches (Charniak 1983) which pass symbolic markers from node to node. In a marker passing approach it is possible to use the markers to determine what source node activated any given node. Some spreading activation approaches have similar “activation signatures” that, like markers, enable the system to determine what nodes activated a node. For example, REMIND uses two qualitatively different kinds of activation, the “evidential activation” discussed earlier which both selects and ranks items, and a separate “signature activation” used to determine what nodes activate other nodes. Another system which mediates between the two approaches is Sapper (Veale 1995) in which spreading activation is propagated using waves with both an evidential amplitude and a signature frequency. In Sapper, signature frequencies are represented by prime numbers representing each node; as activation spreads through the network signature frequencies multiply together, enabling the path an activation wave took to reach a node to be partially extracted by prime factorization.

Other relevant approaches include the analogical retrieval systems MAC/FAC (Gentner & Forbus 1991, Law, Forbus & Gentner 1994) and ARCS (Thagard et al. 1990). Both MAC/FAC and ARCS attempt to find analogical targets in memory similar to some presented source. MAC/FAC, as discussed earlier, applies a fast but approximate numeric similarity metric across its memory to quickly find the best set of items to retrieve; this is similar in spirit to the fast approximate matching algorithms used in the context sensitive asynchronous memory approach. ARCS traces links between components of a source item and components of target items in memory, attempting to find a set of candidate sources most likely to be relevant matches; this is similar in spirit to the selection of a buffer of relevant candidates used in the context sensitive asynchronous memory approach.

*AI Models of Cognitive Phenomena.* There are a number of other approaches which, while not strictly psychological, nonetheless address memory effects in complex cognitive phenomena like creativity, design, analogy and scientific discovery. For example, Wolverton's KDSA approach uses a modified spreading activation process under strategic control to try to find distant analogies for creative cross-domain problem solving. Ram's AQUA (1989) system represents open questions that arise during story understanding as explicit knowledge objects which can be recalled later as new stories are processed, to aid both answering the questions and understanding the new stories. Simina's ALEC (1999) system simulates the creative process of an inventor tackling extremely difficult problems through a process called *enterprise-directed reasoning*, in

which an agent maintains several long-term enterprise goals and attempts to opportunistically exploit both interactions between enterprises and exogenous events; to take advantage of events going on in the environment, ALEC uses a working memory with buffers of recent problems and viewed objects to help simulate recency effects. Moorman's ISAAC (1997) system, in addition to its use of the Nicole-MOORE memory system, uses a rich ontology of information as part of a creative understanding process, enabling it to creatively exploit the information it retrieves from memory even when that knowledge cannot be exactly applied as is. MINSTREL (Turner 1992, 1994) could similarly perform "creative recall" in an attempt to find relevant items if initial retrieval failed. CYRUS (Kolodner 1983) uses high-level retrieval strategies to elaborate, to fill in missing features, and to guide search of the structure of memory. DAYDREAMER (Mueller & Dyer 1985) used a strategic memory retrieval process to both retrieve old episode and generate hypothetical scenarios.

*Information Retrieval Problems.* A variety of approaches in the information retrieval and Web fields have attempted to create autonomous query systems that actively try to process queries over time without human supervision. Most notable among these are perhaps the Continual Query project (Liu & Pu 1996, Pu & Liu 2000) and Veda<sup>TM</sup>'s Karnak<sup>SM</sup> (Vedasoftware 2000). Approaches such as relevance filtering also maintain a query as a distinct object that can be extended or augmented based on feedback (Robertson & Sparck Jones 1976, Sparck Jones 1979, Salton & Buckley 1990; also see Sparck Jones & Willet 1997 for an overview).



**Limits of Existing Approaches.** While psychological and cognitive models focus on beneficial features of human memory, like priming, that are rarely present in general AI systems, they do have a number of limitations.

Psychological models of memory retrieval often focus only on the accurate modeling of the memory retrieval task with performance and usability often not as important as fit to data. Because they focus on modeling the observed properties of specific phenomena, most computational models focus on modeling and predicting theoretically significant variables without reference to the performance of the implementation. (See the earlier discussion of Soar in Section 2.4.3). Furthermore, because of the effort involved in the modeling process, psychological models gloss over interactions with other reasoning tasks making psychological models hard to use for actual AI applications.

For example, many computational models of memory focus on modeling the data for specific subtasks of memory retrieval which can easily be tackled by experiment, such as word retrieval, concept retrieval or episode retrieval, and thus as a matter of tractability of mechanism do not address the general memory task (despite the fact that the data indicate that many memory subtasks share essentially the same properties). Furthermore, many computational models do not incorporate all of the features necessary to perform the complete memory retrieval task. Examples picked on at random include the holographic memory retrieval system CHARM (Eich 1985), the AWM “random memory retrieval” model (Landauer 1975), and the prototype category learning system ALCOVE (Krushke 1992). All of these theories propose novel mechanisms to explain common

human memory phenomena and achieve remarkable fit to the data doing so. However, each of these system requires some additional memory mechanisms outside their core in order to successfully perform their tasks.

For example, in CHARM memory storage is based on a “holographic” model in which all remembered items are convoluted into a single trace, yet actual retrieval depends on matching a reconstructed trace with a discrete list of possible responses (Eich 1995). In Landauer’s AWM model, working memory items are stored at points in a “space” traversed by a randomly moving marker and retrieved from that marker later based on an attentional radius; while this model achieves remarkable fit to human data it maps less well to the storage and retrieval needs of computer systems — even with a sparse data structure the AWM space would contain large amounts of redundant information. In ALCOVE (Krushke 1992) a probabilistic response function mediates between its connectionist model of category learning and actual human response probabilities. In each of these systems, in order to successfully perform their tasks the core mechanisms underlying the theory have been elaborated with different mechanisms which cannot be readily mapped back to the theoretical layer.

In fairness, these limitations are consequence of design: to make the problem tractable computational modelers often focus on specific memory tasks and subtasks in isolation, attempting to isolate or limit the influence of other processes or phenomena within the human agent. This motivates the distinction between “low-level” and “strategic” memory processes (e.g., Reder 1987, 1988). In contrast, a memory for an

artificial intelligence system must stand on its own, its mechanisms functioning as a unit to successfully provide memory retrieval behavior.

What can the study of memory for artificial systems take away from the study of memory for natural systems? Obviously a model of memory for artificial systems must solve the complete problem of memory retrieval tractably with respect to the performance needs of the task at hand and must take into account the functional needs of other processes operating within the agent. Furthermore, the theoretical underpinnings of an artificial memory are not as important as its functional properties. But many of the functional properties of human memory — priming memory with related information to improve retrieval, robust retrieval based on partial specifications, forgetting of irrelevant information, and so on — would be useful for complex intelligent agents.

To take advantage of research in models of human memory when constructing memories for artificial agents, we need to capture the useful functional properties of human memory in an efficient approach which interacts with and serves the needs of the other components of the agent.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory (working in concert with experience-based agency) addresses the issue of exploiting task and environmental information in a variety of ways. It employs a modified spreading activation algorithm which can exploit contextual information to improve the quantity and quality of retrieval and employs a

incremental/updateable retrieval strategy which can exploit additional specifications to improve the quantity and quality of retrieval. Furthermore, the experience-based agent approach supports these techniques for exploiting context by employing domain-independent task communication mechanisms which enable context to be drawn from a wide range of tasks and sensed environmental events.

#### **2.4.7. Desideratum 7: Exploits extra resources if available**

Sometimes an agent or system within an agent faces resource limits; at other times it faces a resource bounty. If extra resources are available, a memory system should use them to improve the comprehensiveness of its memory search and thus hopefully the accuracy of its retrievals. For example, if task or environmental performance does not demand an immediate response from memory, a memory system should exploit the extra temporal resources available to it. Similarly, in a system whose processing resources can be flexibly allocated to different subsystems, memory retrieval should exploit any idle processing cycles as they become available.

**Existing approaches.** Few existing memory retrieval systems deal with exploiting additional information directly, although systems such as PRODIGY/ANALOGY that search their knowledge bases incrementally in search of a “good enough” retrieval could easily be modified to do so.

One research tradition that deals with exploiting additional resources is anytime algorithms, which attempt to produce valid and incrementally improving solution over

the course of the reasoning process (Grass 1996). This enables reasoning to be cut off at any time and still yield a good solution, or alternatively for a system to continue to reason indefinitely until an excellent solution is produced. Examples of applications of anytime algorithms include (Haddawy 1996, Horvitz et al. 1989, Zilberstein 1993, Zilberstein & Russel 1993).

Autonomous information retrieval systems, such as the Continual Query project (Liu & Pu 1996, Pu & Liu 2000) and Veda<sup>TM</sup>'s Karnak<sup>SM</sup> (Vedasoft 2000) discussed in the previous section, by their nature exploit additional resources over time; however these systems take as input and produce as output human-readable queries, as opposed to specifications for memory retrieval requests.

**Limits of Existing Approaches.** While incremental memory retrieval systems such as PRODIGY/ANALOGY, continual information retrieval systems such as Karnak, and the field of anytime algorithms all approach the problem of exploiting additional resources to improve information retrieval, no single system directly addresses the problem of building a general memory retrieval system for artificial intelligence applications that can automatically exploit additional resources as they become available.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory (working in concert with experience-based agency) addresses the issue of exploiting additional resources by employing an

incremental/anytime processing strategy which enables the system to exploit additional time or information to improve retrieval quality and quantity.

#### **2.4.8. Desideratum 8: Potentially interleavable with reasoning**

In order for memory retrieval systems to exploit information from the task or environmental context to improve the precision of their search, and for memory retrieval systems to exploit additional temporal or processing resources when available, a memory retrieval system must be structured in such a way that it can run in parallel with a reasoning task — either as a directly parallel system or as an interleavable coroutine. This places demands on the autonomy of the memory process; it must now maintain an internal state which it can incrementally process and it must provide an external interface which extends beyond simply answering questions to cover control by — or control of — other processes.

**Existing Approaches.** Enterprise-directed reasoning approaches, such as used in ALEC (Simina 1999), maintain long-term goals called enterprise goals which the system attempts to satisfy over time, possibly through the efforts of reasoning tasks or as a result of exogenous events. Autonomous information retrieval systems like the Continual Queries project and Karnak maintain an explicit representation of queries and provide an interface that enables other “processes” like human users to interact with query as the query system continues its improvement attempts.

**Limitations of Existing Approaches.** Enterprise-directed reasoning provides a framework for the long-term processing of knowledge goals, including in theory memory retrieval goals; however, it does not directly address the issue of an ongoing memory retrieval process actively trying to satisfy those goals. For these reasons, enterprise-directed reasoning and the context-sensitive asynchronous memory approach are complementary (Simina, 2000, personal communication).

Autonomous information retrieval systems, while providing an interface that human users can use, do not provide a query language usable by memory retrieval systems nor an interaction language usable by artificial intelligence systems.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory addresses interleaving with reasoning process by employing an incremental/anytime processing strategy which can work in parallel with other tasks in the agent. Experience-based agency supports this by employing domain-independent task communication mechanisms which enable memory to communicate with a wide range of tasks.

#### **2.4.9. Desideratum 9: Provides guidelines for reasoning integration**

A memory system which is at least partially autonomous may not behave in exactly the same way as a traditional memory as far as reasoning tasks are concerned. New questions have to be answered about the interaction between reasoning and memory. Should the memory be polled for answers or will it autonomously return them? If a

memory is incremental, how many “reasoning cycles” must it be given to produce a useful answer, and after how many cycles should the reasoner give up on getting a new answer? If a memory needs additional information about the task or environmental context, does the reasoner need to provide that information or will the system’s architecture do so automatically? If it is not automatic, how much reasoning effort should the reasoner expend before providing new information? And if the memory system autonomously returns answers, how should the reasoner incorporate an answer which arrives at an unexpected time into its reasoning context?

While answers to these questions are technically not part of the memory itself, they are necessary to fully take advantage of an advanced memory retrieval system. A memory retrieval system which satisfies the advanced desiderata listed earlier should also supply guidelines for reasoning tasks to exploit those features of the memory.

**Existing Approaches and their Limitations.** The final desideratum, providing an outline of how a general, autonomous memory retrieval system could be integrated with reasoning processes, is an issue which only arises for systems that attempt to address most or all of these issues simultaneously. No existing system serves as a fully functioning general autonomous memory retrieval system, and as a consequence no existing system addresses the issue of how to integrate one into a reasoning process.

**Benefits of the Context-Sensitive Asynchronous Memory Approach.** Context-sensitive asynchronous memory does not address reasoning integration directly; it is a



theory of memory. Experience-based agency serves as a framework built around context-sensitive asynchronous memory which addresses the issue of reasoning integration, providing a “design pattern” for integration mechanisms to specify how to integrate reasoning tasks with memory, along with sample integration mechanisms for planning and information retrieval.

## **2.5. Conclusion**

The desiderata we have outlined for memory retrieval in cognitive agents place many requirements upon the functional capabilities, performance characteristics and applicable technologies for memory retrieval systems. Most existing systems focus on one or more of these variables with respect to at most a handful of these desiderata, generally as an implicit consequence of other aspects of their design.

In contrast, the context-sensitive asynchronous memory approach, working within the experience-based agent framework, attempts to address all of these desiderata directly and simultaneously. To do so the components of a context-sensitive asynchronous memory work together (and with the features of an embedding experience-based agent if present) to satisfy the desiderata, rather than solving the desiderata by standing on their own.

To more clearly illustrate how the context-sensitive asynchronous memory approach satisfies these desiderata and thus serves as a memory architecture suitable for a

cognitive agent, we now turn to a detailed description of context-sensitive asynchronous memory itself.

# PART II.

## APPROACH

---

### *The Context-Sensitive Asynchronous Memory Approach*

Context-sensitive asynchronous memory is a departure from existing approaches to memory retrieval for a variety of reasons, requiring modifications to how memory systems normally function, modification to reasoners to deal with these changes in memory, and modifications to overall agent design and control to enable memory and reasoning to work together.

Chapter 3: Context-Sensitive Asynchronous Memory discusses the core of the approach itself, presenting its structure and algorithms, analyzing the approach's technical properties, and analyzing qualitatively how to use a context sensitive asynchronous memory and under what conditions that use would be most useful.

Then, Chapter 4: Integration Mechanisms discusses the impact the context-sensitive asynchronous memory approach has on reasoning tasks, presenting a design for integration mechanisms that enable reasoning tasks to cope with asynchronous retrievals efficiently.

Finally, Chapter 5: Experience-Based Agency puts memory, reasoning, and reasoning integration together into a whole picture, outlining how to construct a complete agent built around a context-sensitive asynchronous memory.

# CHAPTER III.

## CONTEXT-SENSITIVE ASYNCHRONOUS MEMORY

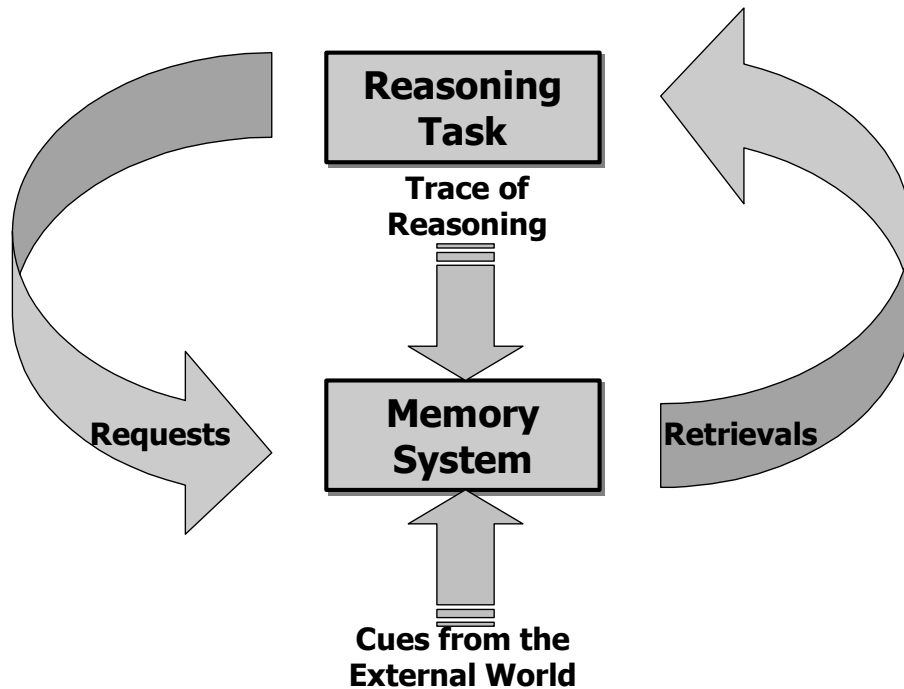
---

*An experience-based memory retrieval architecture for cognitive agents*

This chapter presents a general solution to the problem of getting useful answers from large knowledge bases under resource constraints. This solution combines two features: it interleaves memory search with the performance of the task that demanded the answers, and it uses feedback from the task and the environment to guide search of the memory that holds the answers. Figure 3.1 illustrates the overall structure of this solution.

This chapter presents both processes and representations that realize this interleaved, feedback-driven approach. First, the chapter presents *context-sensitive asynchronous memory* itself, a method for memory retrieval that combines asynchronous retrieval management, context-addressable search and content-addressable storage. Context-sensitive asynchronous memory provides memory-level support for interleaving memory retrieval with task performance while simultaneously exploiting feedback from that task.

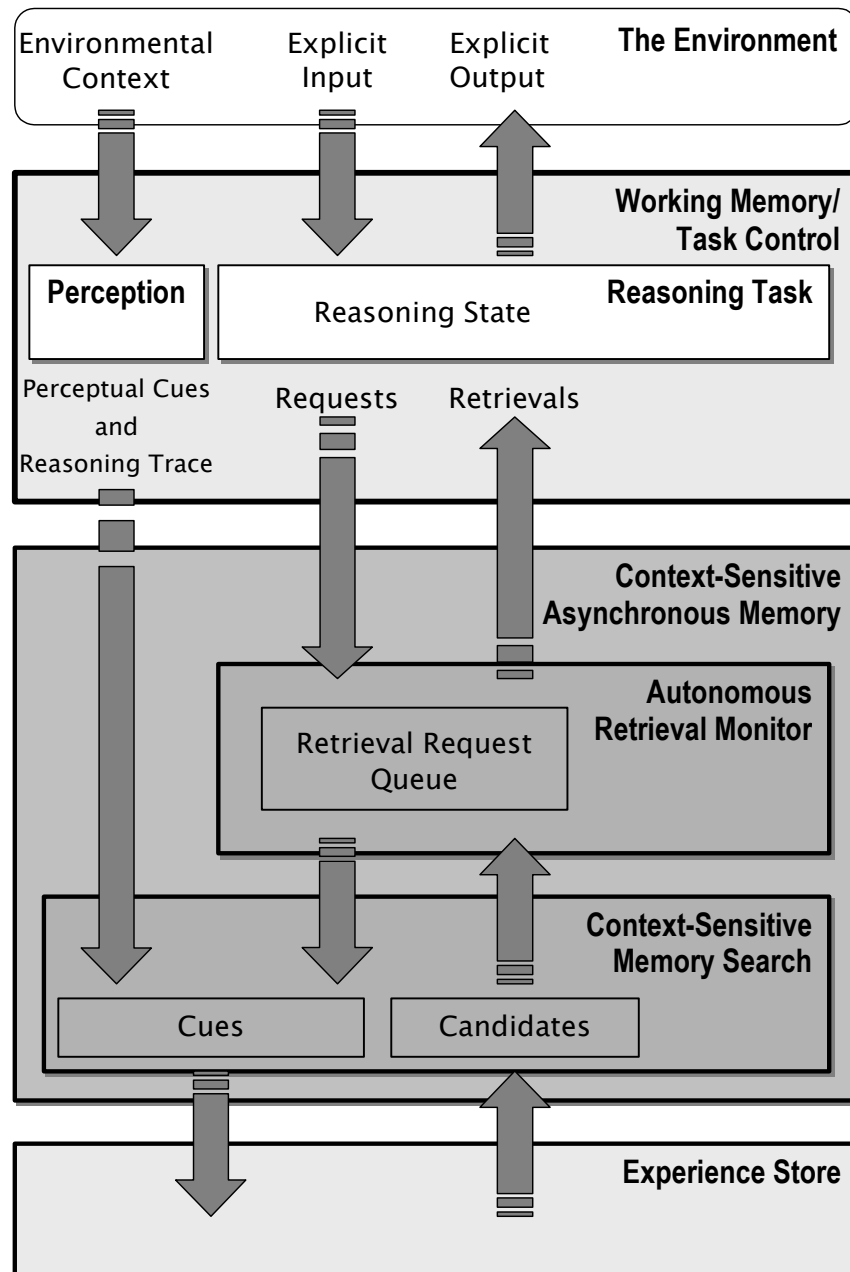
Then, this chapter presents the *experience store*, a representation which enables context-sensitive asynchronous memory to be applied to a wide range of tasks. The experience store is based on a rich, reified, grounded semantic network knowledge representation. The experience store instantiates the principles of content addressable



**Figure 3.1. Interleaving Thinking, Acting and Remembering to Guide Retrieval**

storage required by context-sensitive asynchronous memory in such a way that it can represent knowledge effectively for a variety of tasks and domains and can enable context-sensitive asynchronous memory to be applied to those domains.

Together, a context-sensitive asynchronous memory operating over an experience store forms the core memory contribution of this dissertation. However, applying this memory approach places additional requirements on tasks and architectures that use it. Later chapters discuss the demands context-sensitive asynchronous memory places on reasoning and how they can be solved, and how some of these demands can be answered in a general way using the experience-based agent approach.



**Figure 3.2. Major Components of Context-Sensitive Asynchronous Memory**

Figure 3.2 illustrates the nested structure of this approach: the core components of context-sensitive asynchronous memory (dark shading) are supported by general architectural components of the experience-based agent architecture (light grey); these in

turn support components specific to reasoning tasks and to the interface with the external environment (boxes in white).

The chapter begins by examining the requirements of the problem and how these requirements create opportunities for a solution. Next, the chapter outlines the overall structure and functioning of the context-sensitive asynchronous memory approach, and follows that by more detailed discussions of its major components: asynchronous retrieval, context-sensitive memory search, cost control policies, and the computational complexity implications of these policies. The chapter then turns to representational support for context-sensitive asynchronous memory, discussing the experience store representation and how it enables the context-sensitive asynchronous memory approach to be applied to a variety of tasks.

With the major structure of context-sensitive asynchronous memory thus outlined, the chapter then turns to how a context-sensitive asynchronous memory can be used to get useful answers and to the conditions under which the approach will be effective. The chapter concludes by summarizing the benefits of the context-sensitive asynchronous memory approach, making specific claims about how it can provide a general solution to the problem of finding good answers to bad questions.



### 3.1. Possibilities Inherent in Memories for Cognitive Agents

This thesis investigates problems in memory retrieval which face general cognitive agents, in particular finding the right answer to poorly-specified questions efficiently by exploiting information available in the agent's task and environment but more generally any problem of a similar profile involving limited information on hand, large amounts of information on store, and limited resources to pursue them.

A memory system designed to serve the information retrieval needs of a cognitive agent should be as domain and task independent as possible. This applies equally to general properties of memory such as representational power and to specific tasks performed by memory, such as finding good answers to bad questions, the signature memory task investigated in this dissertation. One approach to this problem is not to attempt to design the memory in the abstract but instead to design a memory system as part of the design of a comprehensive architecture.

From that agent-oriented perspective, two possibilities become immediately apparent: the potential for reasoning tasks to ask questions and continue working while awaiting an *asynchronous retrieval* response, and the potential for memories to use information from the environment or feedback from the reasoner to make retrieval *sensitive to context*.

One feature of cognitive agents particularly relevant to memory retrieval is an agent's need to constantly perform some action or reasoning to cope with its changing world. Reasoning tasks cannot must continue to act once a question has been asked, regardless

of whether or not an answer has been received. This constant action opens the door to a potentially beneficial scenario: memory can operate in parallel with ongoing reasoning and action, continuing to search for appropriate information rather than limiting its response to each query to what it can accomplish in a single “time slice.” This “asynchronous” memory search includes the possibility of *spontaneous retrieval* when memory finds useful information and *anytime retrieval* upon demand.

Another feature of cognitive agents relevant to memory is the need for a cognitive agent to perform multiple tasks in its environment, which in turn requires expressive representation and matching language to serve the information needs of those tasks. The structure of that representation may capture important information about the structure of the environment not necessarily relevant to performing a task but possibly relevant to guiding memory retrieval. A potentially beneficial relationship exists between autonomous memory search and expressive representation: an appropriately designed memory retrieval algorithm can use that rich representation to guide search of memory based on feedback from the agent’s reasoning and the recent history of its environment. As parallel search goes on, the memory can inspect reasoning processes in the agent or events in the environment in search of additional cues or other information that could guide its search.

A memory that shares these two features — asynchronous retrieval guided by feedback from the environment — is a context-sensitive asynchronous memory. This approach both satisfies the requirements on memory systems for cognitive agents and

exploits the possibilities inherent in them. In doing so, it provides a solution to the problem of getting the right answer to poorly-specified questions efficiently by exploiting information available in the agent's task and environment in a general way to improve retrieval performance.

### **3.2. The Context-sensitive Asynchronous Memory Approach**

A context-sensitive asynchronous memory is a memory retrieval system which can operate as an independent task, searching for information without the oversight of another task, yet which yet remains sensitive to cues from reasoning and cues from the outside world. How can these functional properties be achieved?

In one sense context-sensitive asynchronous memory is dependent upon the architecture within which it is embedded. Without agent architecture in which tasks can operate side by side, asynchronous retrieval is meaningless, and without a mechanism for the memory to receive feedback from the task or environment, context sensitivity cannot work. But even given a friendly architecture, the bulk of the effort required in achieving a context-sensitive asynchronous memory lies within the memory itself.

What is required of a general memory retrieval architecture for context-sensitive asynchronous memory? Obviously a means to autonomously process memory retrievals is required, as is a means to accept feedback from whatever sources provide it. However an approach for full cognitive agents must satisfy a wide range of additional desiderata:

storing a wide variety of information, providing general access to that information, scaling to large knowledge bases while managing cost of retrieval, and so on. Summarizing these desiderata, the three most important desiderata are asynchronous retrieval, context sensitivity, and scalability:

- **asynchronous retrieval:**

*Asynchronous retrieval is the autonomous processing of memory retrieval requests. A memory system can perform asynchronous retrieval by using reified retrieval requests and a retrieval monitor which operates in conjunction with the agent's task controller. Asynchrony logically includes spontaneous retrieval (retrieval at the discretion of memory) and anytime retrieval (retrieval on demand of the reasoner).*

- **context sensitivity:**

*Context sensitivity is using feedback to guide memory search. Context sensitivity can be achieved through a process, called context-directed spreading activation, which operates hand in hand with the agent's working memory.*

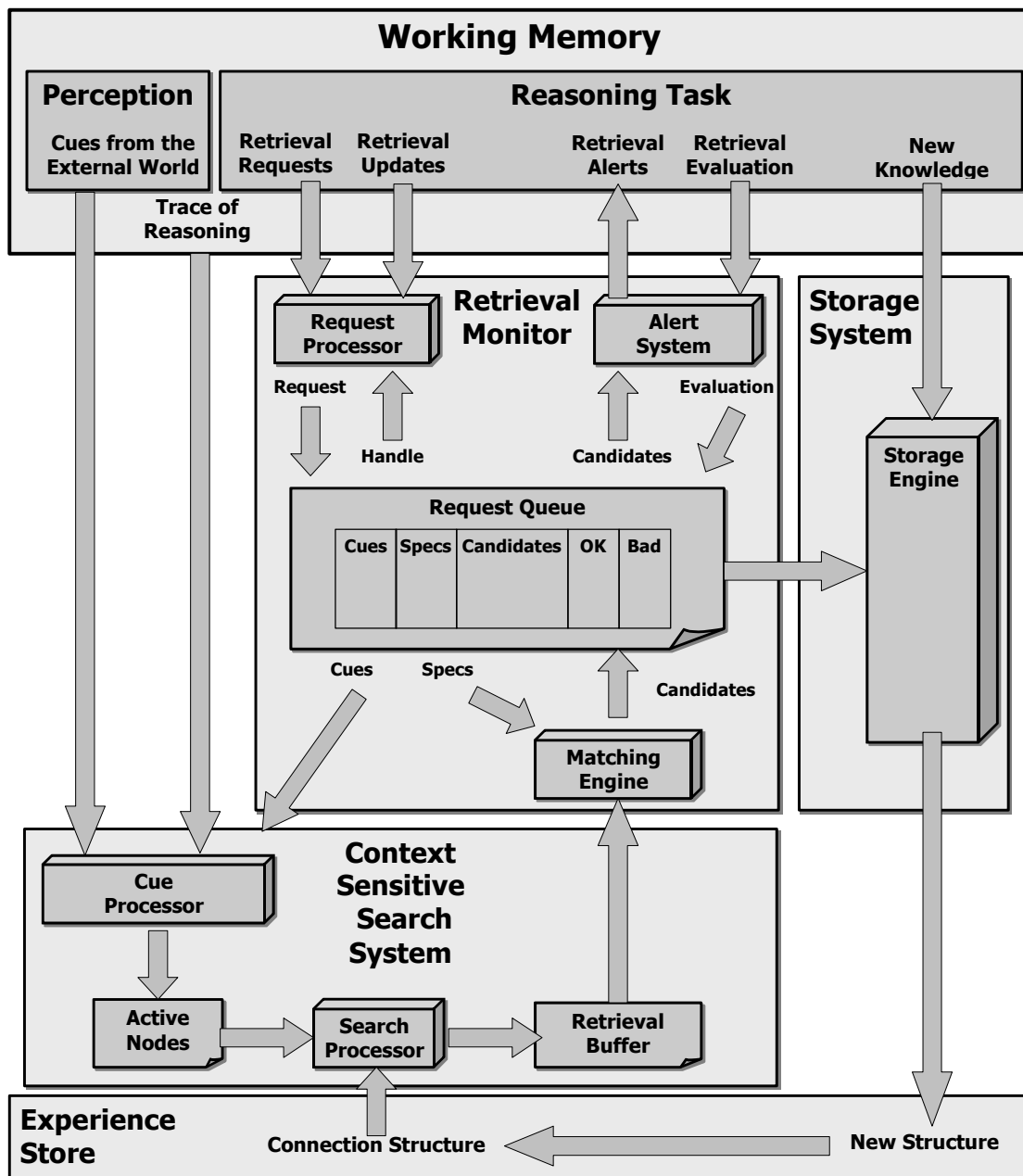


Figure 3.3. Detailed Structure of a Context-Sensitive Asynchronous Memory

- **scalability:**

*Scalability is efficient access to large knowledge bases using the resources that are available. Scalability can be achieved by applying a cost control policy to the operation of the retrieval monitor and context-directed spreading activation processes.*

These features of a context-sensitive asynchronous memory demand certain features of the embedding agent architecture. Asynchrony requires concurrently running tasks and some task control scheme; context sensitivity requires a working memory or other feedback mechanism; and scalability requires a knowledge representation that can support large knowledge bases. Experience based agency is designed to address these issues, but a context-sensitive asynchronous memory could function equally well in any similarly equipped intelligent system.

This discussion paints context-sensitive asynchronous memory with a fairly broad brush; in reality there are many detailed components that must work together to achieve context-sensitive asynchronous memory retrieval (Figure 3.3). Asynchrony is the primary responsibility of the retrieval monitor, which accepts requests and updates and provides a language to enable reasoning tasks to be alerted about retrievals and to evaluate retrieval results. Context sensitivity is the primary responsibility of the context sensitive search system, which accepts cues from reasoning and the retrieval monitor and generates lists of candidate retrievals; context sensitivity is also aided by the structure of knowledge in the experience store and storage processes that add to that knowledge. Scalability is a feature of all portions of the system, affecting how the retrieval monitor expends its effort, how the search system's search algorithms are designed, and how the experience store is constructed.

To better understand how these components work together to make a context-sensitive asynchronous memory function, let us examine asynchronous retrieval, context sensitivity, and cost control in more detail.

### **3.3. Asynchronous Retrieval**

Asynchronous retrieval is the autonomous processing of memory retrieval requests — the processing of a request for information without the oversight of the task that asked for that information. The context-sensitive asynchronous memory approach specifies that a memory system can perform asynchronous retrieval using reified retrieval requests managed by a retrieval monitor; so, what are reified retrieval requests, and how does a retrieval monitor process them?

#### **3.3.1. Reified Retrieval Requests**

Reified retrieval requests are first-class knowledge objects in the experience store which encapsulate requests for information. A retrieval request records the information state of a retrieval request — the type of request, a specification of the information wanted, what task asked for the information, what candidates have been found so far, and so forth — and provides an API which allows that information to be maintained and updated. Retrieval requests are essentially glorified knowledge goals, coupled with records of their own satisfaction, given a privileged position within the experience store through the operation of the retrieval monitor.

**Features of Retrieval Requests.** Retrieval requests contain specifications, cues, importance, current candidates, candidate history, retrieval alerts, successful candidates, and rejected candidates:

- **specifications**

A memory retrieval request's specifications define what kinds of memory items will satisfy the reasoning task's need for information. Specifications are expressed in a language which allows the memory retrieval system to easily match candidates for retrieval against the specifications; ideally, the specifications should also enable the memory retrieval system to compare and rank candidate retrievals.

- **cues**

Cues are context: information about a question beyond its specifications. A cue points to a set of items which might contain potential answers to a question, but does not provide any information about whether a specific item out of that set actually is an answer.

- **importance**

The importance of the request is a guide to how much effort a memory retrieval system should expend on the request. High importance requests should receive the lion's share of the available search resources; low importance requests should be processed incrementally over a longer period of time.



- **current candidates**

The current set of candidates for a memory retrieval request are the set of items which the context-sensitive search process has found that might serve as an answer to the question. The current set of candidates is always changing as the context-sensitive search system incrementally sweeps the knowledge base.

- **candidate history**

The list of candidates a memory retrieval request has examined, including items which were examined and passed over, items which were suggested to reasoning tasks (retrieval alerts), items which the reasoning task accepted (successful candidates), and items the reasoning task rejected (rejected candidates). The memory retrieval system maintains this list for two reasons: first, to prevent the system from sending a retrieval alert which has already been accepted or rejected, and second, to enable the storage module to modify the knowledge base to improve retrieval in the future.

- **retrieval alerts**

New items found in the current candidates that satisfy the retrieval request specifications. These candidate retrievals are suggested to the task that posted the retrieval request in the form of an alert (an asynchronous message between tasks).

- **successful candidates**

Candidate retrievals which the reasoning task has verified are correct answers to the question. The retrieval specification language is memory-specific, and is

designed to be quickly and easily processed by the memory; as such, it may not encapsulate the complex reasoning steps necessary to really match a candidate against a reasoning task's needs. Therefore, the memory retrieval system allows reasoning tasks the opportunity to accept or reject the results of its work in an attempt to further tune its retrieval.

- **rejected candidates**

Candidate retrievals which the reasoning task has rejected as incorrect answers to the question it asked.

**Matching Language.** The specifications of a retrieval request should enable it to determine whether a candidate item is good enough to serve as a retrieval for this request, yet should be flexible enough permit the return of an approximate guess if the reasoning task so desires and should be parsimonious enough to be processed efficiently.

Specifications are written in a matching language that depends on the representation language of the knowledge store. The specifications should support a variety of features: a language for inspecting the structure for complex knowledge items, a variety of match thresholds for features including “exact”, “member”, “fuzzy” and “any,” a set of logical and composition operations enabling conditions to be applied, and optionally “fuzzy” matching criteria that allow for inexact or partial matches of conditions. For more details on a matching language, see the details of the Nicole implementation in Section 7.

### 3.3.2. Features of a Retrieval Monitor

The retrieval monitor is an autonomous process which maintains an active list of reified retrieval requests and attempts to satisfy them. The retrieval monitor keeps track of requests for information made by reasoning tasks using a priority queue of reified retrieval requests, and actively attempts to find information which satisfies those requests without the oversight of the source reasoning task. The retrieval monitor manages the amount of effort it expends on retrieval, retaining the option to terminate or suspend low-priority requests if too many resources are being expended upon retrieval. The retrieval monitor continuously attempts to satisfy its requests in a process called the *retrieval cycle*, in which it collects a set of candidate retrievals from the experience store using a *selection task*, and then attempts to *match* those items with the specifications in the retrieval request queue. The selection task in an experience based agent is a context-sensitive memory search process, but in some other asynchronous memory architecture the selection task could just as easily be an indexed, exhaustive or even random search process. When the retrieval monitor finds a candidate that matches a specification, it notifies the requesting task using an *alert*, implemented in an experience-based agent by posting information to the agent's working memory.

**Capabilities of Retrieval Monitors.** Viewed as an object, a retrieval monitor must, of course, support the traditional methods of a memory retrieval system: it must accept requests and return responses. But in an asynchronous memory system additional features are required. The retrieval monitor does not have access to the reasoning task's

criteria of success, and thus it can determine neither whether an individual candidate retrieval is actually useful, nor whether its overall response to a retrieval request has been successful, nor even whether the retrieval request is still relevant to the task's current processing. Therefore, when an alert is posted, a retrieval monitor must provide methods to allow a task to *accept* or *reject* the candidate retrieval, as well as methods that allow tasks to *accept* a memory retrieval request as complete or *cancel* further work on the request.

Beyond the additional features that are *required* for asynchrony lie features *enabled* by asynchrony. For example, a reasoning task could demand a *best guess* initially and then direct the monitor to continue processing the request, seeking better matching items. If a reasoning task finds new information about a request it could *update* the request with new cues or specifications as needed. These additional features of the retrieval process require that reified retrieval requests contain information beyond that necessary for traditional retrievals — including lists of current candidates, histories of past accepts and rejects, priority levels, sensitivity levels, and so on.

**Operations Required by Asynchrony.** A retrieval monitor must support the operations `post-request`, `update-request`, `get-candidates`, `accept-candidate`, `reject-candidate`, `accept-request`, and `cancel-request`:

- **post-request**

Create a new request for information, including the specifications for successful

retrievals and optionally any cues the requesting task has. If the requesting task does not attach an importance to its request, an default will be assigned.

- **update-request**

Update the cues, specifications and/or importance of this retrieval request.

- **get-candidates**

Get the current set of candidates that the memory retrieval system believes satisfy the specifications of the retrieval request.

- **accept-candidate**

Inform the memory retrieval system that a candidate is indeed a successful retrieval and instruct it to update the request status accordingly.

- **reject-candidate**

Inform the memory retrieval system that a candidate is not acceptable as a result for this request and instruct the memory to update the request status accordingly.

- **accept-request**

Inform the memory retrieval system that this request has been satisfied to the reasoning tasks' satisfaction.

- **cancel-request**

Inform the memory retrieval system that work on this memory retrieval request has not been satisfactory and that no further work on request is desired.

**Extended Operations.** Retrieval monitors may support other features to enable more complex task processing or to support additional learning. For example, when a task is overwhelmed with needs for information relating to a complex task with many subgoals, it may wish to suspend and resume retrieval requests depending on the context of what it is working on. Furthermore, because a memory retrieval system has a great deal of information about the current reasoning and remembering context, the potential exists for reasoning tasks to alert the memory retrieval system that new information has been generated to enable the memory retrieval system to store new items in such a way that they can be more easily retrieved.

The extended operations a retrieval monitor may support include suspend-request, resume-request, and post-storage:

- **suspend-request**

Inform the memory retrieval system to stop active work on this memory retrieval request but to remember its state for continued work in the future.

- **resume-request**

Ask the memory retrieval system to resume work on a past unsatisfied request.

- **post-storage**

Inform the memory system that a new piece of information has been generated, possibly in relation to a past retrieval request. This enables the memory system to annotate the item with information from the current activation and retrieval context, enabling it to be more accurately retrieved in the future.

### **3.3.3. How the Retrieval Monitor Processes Retrieval Requests**

The retrieval monitor is responsible for the entire life history of a retrieval request: within the monitor retrieval requests are created, processed, updated, and ultimately destroyed. Figure 3.4 illustrates the life history of a retrieval request in a context-sensitive asynchronous memory. Retrieval requests are “posted” by reasoning tasks on the basis of information stored in working memory, causing the retrieval monitor to create an explicit declarative representation of a request which it stores in the request queue (1). Working memory data and outstanding requests cue the spread of activation in the experience store (2). The most active items in the experience store become candidate retrievals (3) which are matched against all outstanding retrieval requests in the queue (4). If a good retrieval is found, a retrieval alert is created (5) which notifies reasoning that it should respond.

As this life cycle shows, much architectural support must exist beyond the memory system to enable asynchronous memory retrieval: retrieval requesters that generate requests for information, retrieval accepters which handle alerts, choice mechanisms which determine how to handle new retrievals and integration mechanisms which actually incorporate them into the current reasoning context. Of course, from the perspective of the innards of the memory module, these additional tasks are invisible.

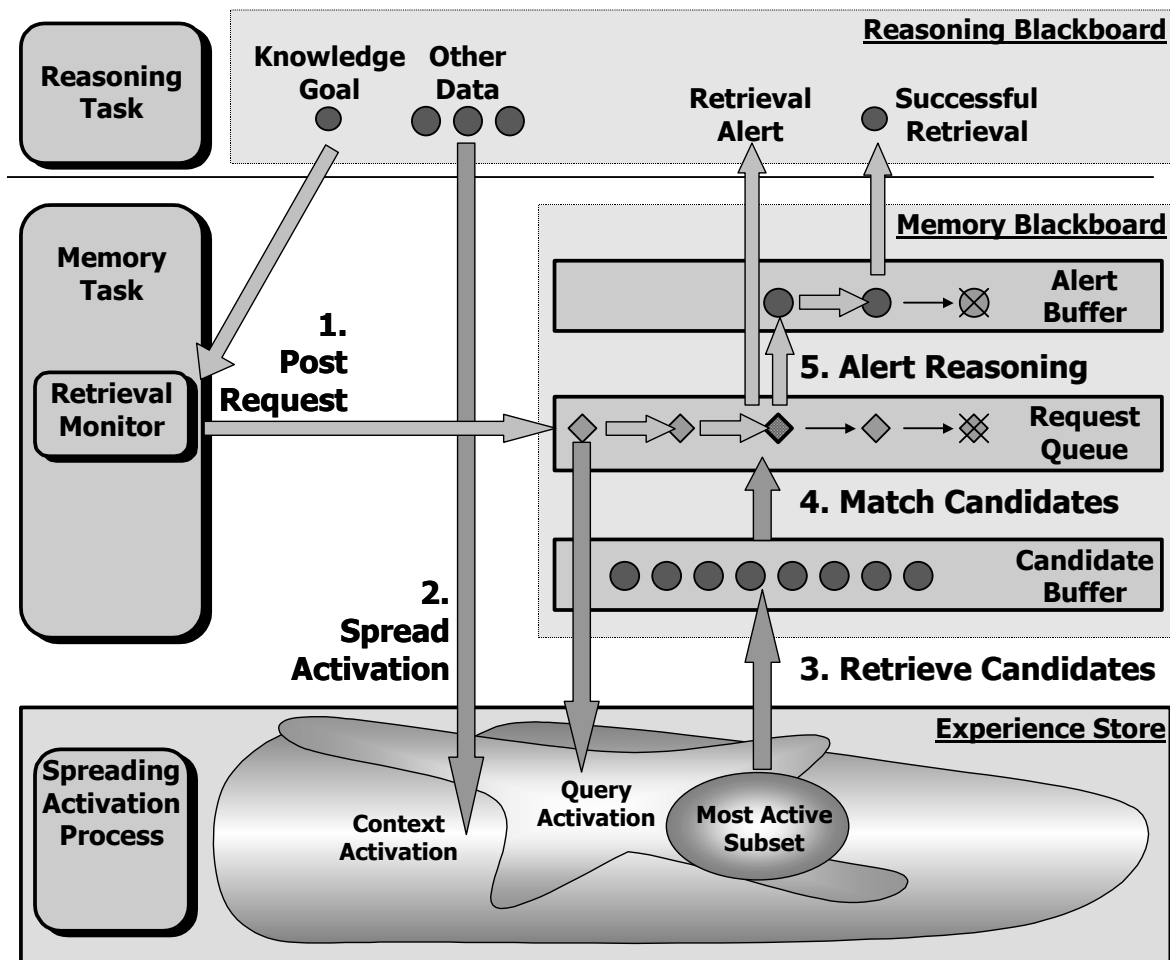


Figure 3.4 The Life History of a Retrieval Request

Not invisible, however, is the selection task. The selection task mediates between the retrieval monitor and the experience store; it consumes the processing power the retrieval monitor grants it and produces the candidates that retrieval monitor desires in return; and in a context-sensitive asynchronous memory, it is the vehicle by which feedback from the task or the environment affects how the memory system's resources are allocated most efficiently to the portions of memory most likely to contain the answers.



The most important part of communicating with a selection task implemented using context-directed spreading activation is providing cues, described in more detail in the next section. Unlike systems such as ACT\* (Anderson 1983) or REMIND (Lange & Wharton 1994), activation in a context-directed spreading activation network does not spread from nodes that are “clamped” at a certain level of activation; instead, activation propagates as discrete quanta of changes, called perturbation, which are added to the network as a result of a cueing process. Therefore, in order for a context-directed spreading activation network to weight its nodes correctly, a request needs more than just a list of cues; it must also initiate the propagation of activation from those cues out into the network.

To limit the costs of the context sensitive search process, activation is not propagated from retrieval requests continually. Instead, activation propagates when requests are originally created, when requests are updated, or when requests are explicitly refreshed. This deliberative propagation of activation is designed to simulate the limited amount of effort a human has to expend on memory retrieval. While of course a reasoner can choose to update all of its outstanding requests each retrieval cycle, because requests are deliberately refreshed a reasoner can choose how to allocate its effort among the requests it has active, weighting the spreading of activation to the requests related to the goals it is processing most actively.

### 3.4. Context Sensitivity

By themselves, reified retrieval requests and a retrieval monitor could make up the core of an asynchronous memory system as long as *some* task existed that produced a set of candidate items on every retrieval cycle. This could be as simple as a sequential search of memory items examining some fixed number on every cycle, as complex as hashed search based on the specifications provided to the memory module, or as rash as random selection of  $n$  items on each retrieval cycle.

However, none of these approaches satisfy the desiderata we identified earlier. Sequential search of an experience store is inefficient, especially when matching operations are expensive. Hashes or indexed searches are more efficient, yet require foreknowledge of the dimensions by which we should hash or index, and are thus less than general. Furthermore, when questions are poorly specified we may not have enough information to trace index links to our destinations or to seed our hash functions. To overcome this problem, we want to find a way to incorporate additional information about the desired answer dynamically, as it is found or generated.

In short, we want a context-sensitive selection task. *Context-directed spreading activation* (CDSA) is a novel memory search algorithm that can serve as a context-sensitive selection task for a context-sensitive asynchronous memory.

CDSA uses the set of currently active items in the memory system to direct and inform the further propagation of changes to activation, on the theory that memory

requests are best served warm — that is, most memory requests can be satisfied with concepts closely related to the concepts that the agent has been thinking about or has encountered in its environment. In psychological terms, context-directed spreading activation is a priming or preactivation process (for an overview, see Klimesch 1994; also see McNamara 1992, McNamara & Diwadkar 1996).

### **3.4.1. Activation, Perturbance, Fanout, and Decay**

Unlike traditional spreading activation approaches, in which activation spreads from source nodes out to target nodes in the knowledge base (e.g., Collins & Loftus 1975, Anderson 1983, Jones 1989, Jones 1993, Jones & Langley 1995), context-directed spreading activation makes an explicit distinction between the *activation* of a node, which determines a node's relevance for retrieval, and the *perturbance* propagating through those nodes, which determines how activation and hence relevance changes. Perturbance is similar to the concept of “zorch” or amount of propagating activation (Hendler 1989, Domeshek 1992). By explicitly separating dynamically propagating perturbation from the more static patterns of node activation, rather than treating them with the same set of equations, we open the possibility for the spreading activation process to become context sensitive. Perturbance propagates and activates nodes; the activation of nodes then in turn affects future propagation of perturbation according to the CDSA algorithm.

**Activation Levels.** In a CDSA network, the *activation* of the nodes in the knowledge base determines the order in which nodes are considered by the context-sensitive retrieval process. Each node has a default weight or base activation which would determine the order in which the node would be considered for retrieval in the absence of any cues or contextual information. In a virgin knowledge base, the default activation of each node might be identical; as a agent learns over time what items are useful candidates for retrieval it could adjust the default weight of a node to influence how often an item was considered as answers to future retrieval requests. The cost of sorting the network based on activation can be controlled by only sorting nodes whose activation differs from their base value by some threshold; this number of nodes is in turn limited by the limits inherent in the CDSA algorithm and in any additional cost control policies, discussed shortly.

**Quanta of Perturbance.** Perturbance is the amount of change in activation propagating through the network. Perturbance can be viewed as quanta of activation which propagate along the links between nodes, depositing activation on a node and then splitting into new, smaller quanta passed along the links of a node to the nodes connected to it. *Cues* are the source of perturbance. A cue consists of a node and a strength, selected either explicitly by the asynchronous memory system or implicitly through the working memory; the result of a cue is the creation of a quanta of perturbance on the target node.

**Fanout of Perturbance.** Quanta of perturbation propagate from node to node along the connections between them — along the knowledge links between concepts in the network. The fanout of a node is the number of links attached to it; the strength of a propagating quanta of activation is divided by fanout to its children according to the CDSA algorithm. With each generation every quanta produces a new set of children, each successive generation growing weaker and weaker until the quanta are “thresholded out” to control spreading activation costs. As an additional limit on perturbation, a perturbation attenuation factor can be applied at each propagation point, causing the strength of perturbation to decay somewhat even if fanout is unity.

**Decay of Activation.** Even when all quanta of perturbation are gone, the activation that they deposited remains on the nodes in the network, biasing them for retrieval processes. Without some process to remove activation, the levels of activation in the network would continue to rise as new cues are added to the network and new perturbation is deposited. To control the amount of time this bias exists, activation of nodes in the network slowly decays over time, until each node returns to its base activation value. This concept of decay and base activation also enables additional cost control processes, discussed shortly.

**Relation Leakage.** Another important factor in the CDSA propagation process is *relation leakage*, an optional addition to the CDSA process in which some amount of perturbation propagating along a link “leaks” up to the parent relation of that link. Relation leakage is an important way that a CDSA network can automatically adjust how

activation flows through the activation network based on the kinds of cues that are being activated. When relation leakage is activated, additional “virtual links” are created for each class of link in the system, with a strength based on a relation leakage parameter.

### **3.4.2. Sources of Perturbance**

While all activation and perturbance in a context-directed spreading activation process is identical at the level of mechanism, there are at least two different sources of perturbance at the level of the memory task: *query perturbance*, which spreads from the specifications of a retrieval request, and context or *priming perturbance*, which spreads from items stored in the system’s working memory. Query perturbance is propagated explicitly whenever a retrieval request is created, based on the knowledge items contained in the matching specification of the request. Priming perturbance is propagated either implicitly through the add/delete hooks in the working memory or explicitly through “hints” or *cues* which are added to the retrieval requests’ specification. Other than their source, query perturbance and context perturbance are identical, exploiting the same context-directing spreading activation process.

### **3.4.3. Types of Context-Directed Spreading Activation**

As noted by Charniak (1983) unconstrained spreading activation or marker passing would activate an entire knowledge base; therefore, most marker passing and spreading activation approaches apply a variety of limits such depth cutoffs that limit the amount of effort expended (e.g., ACT\* (Anderson 1983b), EUREKA (Jones & Langley 1995)

REMIND (Lange & Wharton (1994), KDSA (Wolverton 1994), Sapper (Veale 1995) and WebSCSA (Crestani & Lee 1999)). “Traditional” spreading activation approaches using fanout and propagation thresholds are naturally limited, with fanout ultimately placing limits on how many nodes are reached (Anderson 1983b, Jones 1989, 1993, Jones & Langley 1995).

Context directed spreading activation shares these natural limits of traditional spreading activation but goes beyond them. The key novel contribution of the approach is to alter the way changes in activation spread from node to node based on existing patterns of activation in the knowledge base.<sup>6</sup> The result is that given an appropriate context CDSA can reach targets with a greater “semantic distance” than traditional spreading activation while expending the same amount of retrieval effort. A more detailed theoretical analysis of CDSA and its impact upon semantic distance is given later in this chapter, and empirical results related to this impact are presented in Chapter 7.

---

<sup>6</sup> This approach shares similarities with a proposal by Wolverton & Hayes-Roth to alter the weights of links in a spreading activation network (Wolverton & Hayes-Roth 1993) but this proposal was apparently not implemented (Wolverton 1994, p.32).

Obviously there are a variety of ways by which existing activation could guide the spread of new activation; this research has examined two classes of guided propagation, *primed spreading activation*<sup>7</sup> and *gated spreading activation*:

- **primed spreading activation:**

Perturbance of activation spreads more efficiently to items which are already activated above some threshold value. Primed spreading activation attempts to direct perturbation propagating in the knowledge base to items which the agent has recently seen or which are related to items the agent has recently seen — in short, items which have recently been activated. The result of primed spreading activation is to reduce the effective size of the knowledge base.

- **gated spreading activation:**

Perturbance of activation spreads more efficiently along links mediated by relation nodes which are active. Gated spreading activation attempts to guide changes to activation along types of paths that instantiate the relationships between the agent is paying attention to — the relations that are active. The result of gated spreading activation is to reduce the effective branching factor of the search space.

---

<sup>7</sup> In earlier publications (Francis & Ram 1997) primed spreading activation was referred to as context activation.



### **The Context Directed Spreading Activation Propagation Algorithm**

#### **Step 0. Cueing:**

For each cue (node, strength pair) added to the network:

**Step 0.1:** Create a quanta of perturbation on the cue's target node with a strength equal to the cue's strength (as in Step 1.3)

**Step 0.2:** Imprint the quanta's perturbation on its target node's activation (as in Step 3).

#### **Step 1. Propagation:**

For each quanta of perturbation in the network:

**Step 1.1:** Find the list of target nodes connected to the quanta's host node.

**Step 1.2:** Apportion the quanta's perturbation strength to each target node according to the CDSA Equation (Equation 3.1).

**Step 1.3:** Create a new quanta of perturbation on each target node, with the appropriate level of strength from Step 1.2.

#### **Step 2. Thresholding:**

For each new quanta of perturbation in the network, if the strength of the quanta is less than the propagation threshold eliminate it.

#### **Step 3. Imprinting:**

For each remaining new quanta, add the quanta's perturbation to its host node's activation.

#### **Step 4. Decay:**

For each active node in the network, decay its value towards its base activation value.

**Figure 3.5 Pseudocode for the CDSA Propagation Algorithm**

Both techniques create a “virtual subset” of the knowledge base, focused on the items that the agent has recently encountered linked by the relationships the agent is paying attention to. This virtual subset enables an agent with a large, complex knowledge base to expend its memory search effort more efficiently, focusing on the content related to its current task.

### 3.4.4. The Context-Directed Spreading Activation Algorithm

While the two types of context sensitivity listed above have different effects on the way activation propagates, both can be captured at the same time by a single context-directed spreading activation equation.

In a CDSA network, each node has some level of activation  $a_i$ . Each link between a node  $i$  and  $j$  is mediated by some relation node  $r$ ; the link  $i \rightarrow j/r$  has a strength  $S_{i \rightarrow j}$  and of course nodes  $i$  and  $r$  have their own activation levels. If a CDSA network employs “relation leakage,” for each link from a node from  $i$  to  $j$  mediated by relation  $r$  an additional “virtual link” is created from node  $i$  to  $r$  with a strength attenuated from its parent link by a relation leakage factor:  $S_{i \rightarrow r} = D_{leakage} S_{i \rightarrow j}$ . When used, these virtual links enable a CDSA network to automatically adjust the activation of relations in the network to mirror the kinds relations that connect the concepts that are most heavily used, providing a limited way to automatically reduce the branching factor of the CDSA process.

Perturbance on a node is divided into a set of quanta  $Q_i$ , each quanta  $q_i$  containing some amount of change to activation  $p_{iq}$ . Propagation of spreading activation occurs in cycles, with each cycle representing the atomic amount of time necessary for activation to propagate along links from node to node. As quanta travel from node to node they are kept separate, enabling a cost control policy to “threshold out” quanta when their

Parameter	Description	Typical Value
$P_{base}$	basic level activation spreading to inactive nodes	1.0
$P_{context}$	active node bias for primed spreading activation	0.5
$R_{base}$	basic level propagating along inactive relations	0.2
$R_{gating}$	active relation bias for gated spreading activation	0.8
$S_{default}$	default strength of a link	1.0
$A_{default}$	default (base) activation of a node	1.0
$P_{initial}$	default amount of perturbation of a cue	1.0
$T_{cutoff}$	minimum amount of perturbation for quanta to propagate	0.01 to 0.001
$D_{attenuation}$	damping factor for weakening of propagating quanta	1.0
$D_{damping}$	decay factor for node activation	0.9
$D_{leakage}$	amount of activation that leaks from links to relations	0.0-0.2
$D_{irrelevance}$	decay factor for irrelevant nodes	1.0-0.5

**Figure 3.6 Parameters of the CDSA Algorithm**

activation becomes too small. Each spreading activation cycle is divided into five phases: cueing, propagation, thresholding, imprinting, and decay (Figure 3.5):

**Cueing.** An external process, such as the asynchronous retrieval monitor or the working memory, may “cue” the network by specifying a (node, strength) pair. Before

each propagation cycle begins, for each new cue a quanta of perturbation is created on the cue's target node and the strength of that quanta is imprinted on the target node's activation. These processes are identical to the quanta creation and imprinting processes described next in "Propagation" and "Imprinting".

**Propagation.** On each propagation cycle, each quanta of activation on a node "splits" and propagates along the links connecting the node to other target nodes in the knowledge base. The result is to add new quanta of activation to each target node linked to the source node.

The amount of perturbation propagating is normalized so that the total amount of perturbation in all new quanta is the same as the amount in the original quanta, possibly weighted by a damping factor. The amount of perturbation allocated to each new quanta, however, is influenced by the activations of the relations that mediate those links (gated spreading activation) as well as the activations of the targets (primed spreading activation). Given a quanta  $q$  of perturbation  $p$  on a node  $i$ , the amount of perturbation  $p_{iq}$  that propagates from node  $i$  to node  $j$  along a link mediated by relation node  $r$  is determined by the equation:

$$(3.1) \quad p_{jq}^{t+1} \leftarrow D_{attenuation} \frac{(P_{base} + P_{context} a_j^t)(R_{base} + R_{gating} a_r^t) S_{i \rightarrow j}}{\sum_{\langle r, k | i \Rightarrow k \rangle} (P_{base} + P_{context} a_k^t)(R_{base} + R_{gating} a_r^t) S_{i \rightarrow k}} p_{iq}^t$$

where:

$\mathbf{a}_i$	activation of node $i$ at time $t$
$\mathbf{p}_{iq}$	amount of perturbation $p$ in quanta $q$ of node $i$ at time $t$
$\mathbf{S}_{i \rightarrow j}$	strength of the link between nodes $i$ and $j$
$\mathbf{P}_{\text{base}}$	basic (non-primed) node propagation parameter
$\mathbf{P}_{\text{context}}$	active (primed) node bias parameter
$\mathbf{R}_{\text{base}}$	basic relation propagation parameter
$\mathbf{R}_{\text{gating}}$	active relation bias parameter
$\mathbf{D}_{\text{attenuation}}$	perturbation attenuation factor

**Thresholding.** Once a new candidate quanta has been generated, its strength is compared with the cutoff threshold for spreading activation. If the magnitude of a new quanta of perturbation falls below a certain threshold, that quanta is destroyed rather than being allowed to imprint or propagate further. This helps reduce the amount of effort expended on context directed spreading activation and is discussed in more detail in sections 3.4.5 and 3.5. The set of quanta that propagate are computed by the equation:

$$(3.2) \quad Q_i^{t+1} = \{p^{t+1}_{iq} \mid p^{t+1}_{iq} > T_{\text{cutoff}}\}$$

where:

$\mathbf{p}_{iq}^{t+1}$	amount of perturbation $p$ in candidate quanta $q$ of node $i$ for time $t+1$
$\mathbf{T}_{\text{cutoff}}$	the propagation cutoff threshold
$\mathbf{Q}_{it+1}$	new set of propagating quanta on node $i$ at time $t+1$

**Imprinting.** When an individual quanta  $q$  of perturbation is added to a node  $i$ , either as a result of an external cue or internal propagation, the value  $p_{iq}$  of that perturbation is *imprinted* upon the node, increasing (or decreasing) the node's activation  $a_i$  by  $p_i$ . At the

end of the imprint cycle, the total amount of activation on a node is equal to its previous activation plus the sum of all propagating quanta of perturbation currently residing on the node:

$$(3.3) \quad a_i^{t'} \leftarrow a_i^t + \sum_{q=1}^{|Q_i^{t+1}|} p_{iq}^{t+1}$$

where:

$a_i^t$	original activation $a$ of node $i$ at time $t$
$p_{iq}^{t+1}$	perturbation $p$ of quanta $q$ on node $i$ at time $t+1$
$Q_i^{t+1}$	set of propagating quanta on node $i$ at time $t+1$
$a_i^{t'}$	intermediate activation $a$ of node $i$ at time $t$ including imprinted perturbation

**Decay.** On the final step of each propagation cycle, the amount of activation of each node decays towards its base value, which may be non-zero. The implications of this feature are discussed in section 3.5.

$$(3.4) \quad a_i^{t+1} \leftarrow D_{damping} (a_i^{t'} - b_i) + b_i$$

where:

$a_i^{t'}$	intermediate activation $a$ of node $i$ at time $t$ including imprinted perturbation
$b_i^t$	base activation $a$ of node $i$
$D_{damping}$	the activation damping factor
$a_i^{t+1}$	resulting activation $a$ of node $i$ at time $t+1$ after decay

An optional feature of the decay cycle is *irrelevance decay*. If a node does not match any active retrieval request, it can be viewed as irrelevant and decayed by an additional amount  $D_{irrelevance}$  to allow more relevant nodes to be matched. When irrelevance decay is active, equation 3.4 is modified to look like:

$$(3.4a) \quad a_i^{t+1} \leftarrow D_{irrelevance}^i D_{damping} (a_i^{t'} - b_i) + b_i$$

where:

$D_{irrelevance}^i$	irrelevance damping function for node $i$ : = 1 if node $i$ matches some request = $D_{irrelevance}$ if node $i$ does not match any requests.
$D_{irrelevance}$	the irrelevance decay factor

Irrelevance decay does not affect the core behavior of the CDSA equations and instead affects (and depends on) the interaction of an asynchronous retrieval manager with context-directed spreading activation, enabling a retrieval manager to focus its search on items which match its retrieval specifications.

**Behavior of the Equations.** The behavior of Equation 3.1 (the core CDSA equation) is easy to understand if the various parameters (Figure 3.6) are pushed to limiting values. The bias parameters serve to determine the degree of primed spreading activation. Raising the context bias parameter  $\mathbf{P}_{context}$  for nodes increases the ease with which changes to activation spreads to active nodes; raising the gating bias parameter for relations increases the ease with which changes to activation spreads along links whose relations are active.

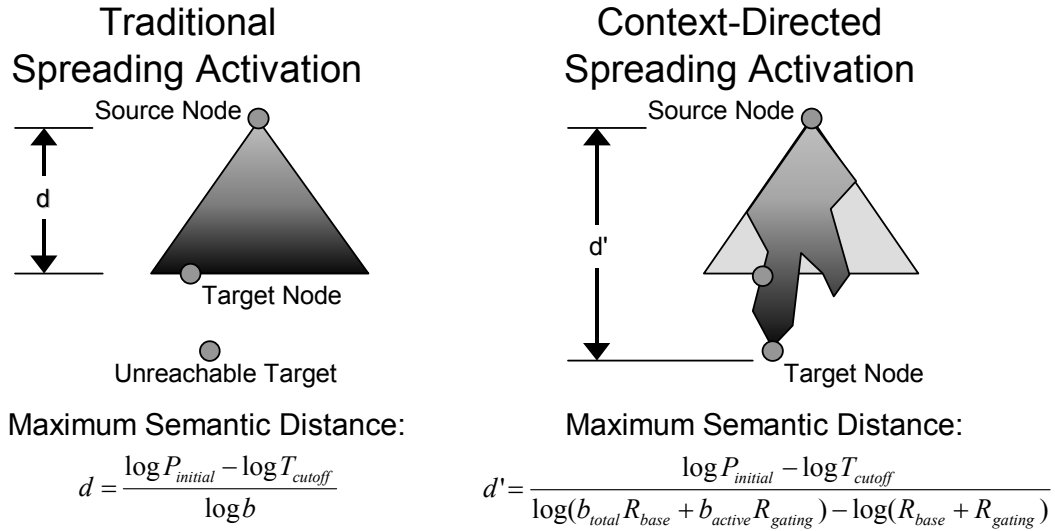
When the bias is set to zero, context-directed spreading activation devolves to traditional unbiased spreading activation with fan-out, such as proposed by Anderson (1983). The base parameters  $\mathbf{P}_{\text{base}}$  and  $\mathbf{R}_{\text{base}}$  determine the properties of this process. If the base parameters are set to zero, essentially the only perturbation that can propagate is context-based: activation can only spread to nodes with some existing degree of activation, and only along links whose relations are active. In practice, intermediate values are chosen for both the base and bias parameters, allowing both context-directed and traditional spreading activation.

### **3.4.5. Semantic Distance and Context-Directed Spreading Activation**

**Self-Limiting Properties of Spreading Activation.** Context-directed spreading activation, like traditional spreading activation, provides a built-in cost control policy. Spreading activation processes that combine fan-out decay of activation with a minimum threshold below which activation cannot propagate are inherently self-limiting: the total number of nodes that can be visited by spreading activation is limited to the maximum activation divided by the threshold value (see Charniak 1983, Jones 1989, 1993, Jones & Langley 1995, Veale 1995).

CDSA does not change this basic principle of spreading activation. Propagating perturbation obeys the same basic laws as traditional spreading activation: there is a fixed upper limit on the size of the set of nodes that can be activated by a single source node. What CDSA does change is *which* nodes constitute that activated set.





**Figure 3.7 Comparing Traditional and Context-Directed Spreading Activation**

**The Effect of Context Guidance in CDSA.** Context guidance alters how changes to activation propagate from a node, with the result that different sets of nodes can become active depending on what other nodes are active in the knowledge base when activation began to propagate (Figure 3.7). Furthermore, this change in propagation changes not only which nodes become active but how much activation they receive, changing the relevance of those nodes and thus their order of retrieval. CDSA thus does not change the amount of effort expended on search from a single node, but instead changes how that effort is expended based on the context and what the impact of that effort is on retrieval order, in the hope that a guided search will be a more effective search.

**Semantic Distance.** Leaving aside the language of hope in favor of measurable terms, CDSA changes the semantic distance reachable by a given amount retrieval effort. The concept of semantic distance was introduced by Wolverton (1994) to measure the

effectiveness of spreading activation approaches at finding information in large knowledge bases. Semantic distance can be simply defined by the number of links between concepts of a query and concepts in a target.<sup>8</sup> The effectiveness of a spreading activation process can be measured by the amount of effort the process must expend to reach a target of a given semantic depth.

For example, an unconstrained spreading activation process must activate a roughly exponential number of nodes to reach targets of a given semantic distance; in a finite knowledge base this curve becomes logistic as the process begins to revisit more and more nodes (Wolverton 1994, p.60). Exponential growth is unacceptable, which is why traditional spreading activation approaches use some kind of cutoff to limit the number of nodes that become activated.<sup>9</sup> The effect of cutoffs on activation is to make certain

---

<sup>8</sup> In Wolverton, concepts were represented as graph structures and semantic distance between two concepts was measured by the average path length between components of the concepts. In the context-sensitive asynchronous memory approach no distinction is made between general network structure and concept-specific structure, and therefore each distinct concept must be represented by a unique signature node. Therefore, for the purposes of this analysis semantic distance is measured from signature node to signature node.

<sup>9</sup> An alternative approach to restricting spreading activation cost can also be found in Wolverton (1994). Knowledge-Directed Spreading Activation (KDSA) exploits “beacons” in the knowledge base to strategically perform a series of constrained spreading activation searches. Presuming that a knowledge base contains many beacons of high quality, KDSA activates a linear number of nodes to reach a given semantic distance and is more efficient than unconstrained spreading activation. In contrast, if beacons are

targets in the knowledge base reachable given a particular amount of retrieval effort, whereas other targets are not.

**Semantic distance and traditional spreading activation.** A fixed amount of initial perturbation in a traditional spreading activation process with fanout and thresholds propagates a limited semantic depth into a knowledge base and activates a limited number of nodes — an exponential function of the semantic depth, but a limited number of nodes nonetheless.

For simplicity of analysis, I will assume that a knowledge base has a regular structure where each node has a fixed fanout  $b$ . An initial amount of perturbation  $P_{initial}$  will be reduced by fanout by a factor of  $1/b$  at each step in the propagation process. Propagation will terminate when the amount of perturbation in each quanta drops below  $T_{cutoff}$ , the propagation cutoff threshold.

$$(3.5) \quad P_{initial}/b^d < T_{cutoff}$$

where:

---

widely spaced and/or are of low quality KDSA can become less efficient than unconstrained constrained spreading activation. Because KDSA essentially applies high-level strategic control to low-level spreading activation processes, it is potentially complementary to the CDSA approach: KDSA's search control component could supply to CDSA contextual information, and in turn CDSA can use that information to ensure that spreading activation effort is spent efficiently.

$P_{initial}$	average initial value of a cue
$b$	the branching factor of the knowledge base
$d$	the cutoff depth for normal spreading activation
$T_{cutoff}$	the propagation cutoff threshold

The semantic distance reachable by spreading activation is simply the number of propagation steps, or  $d$ . Solving 3.5 for  $b^d$ :

$$(3.6) \quad b^d = P_{initial}/T_{cutoff}$$

The maximum semantic distance reachable by traditional spreading activation is therefore:

$$(3.7) \quad d = \frac{\log P_{initial} - \log T_{cutoff}}{\log b}$$

**Impact of CDSA on Semantic Distance.** One of the primary effects of CDSA is to change the relative weighting of links. Assume that this uniformly divides the fanout of the knowledge base into two groups:

- **inactive links:**

Inactive links are links whose parent relations are not active. Each inactive link has some default strength  $R_{base}$  (according to equation 3.1).

- **active links:**

Active links are links whose parent relation are active. Assuming for a moment

that the activation of relations does not change during the spreading activation process; in this case each active link will have an additional strength from gated spreading activation for a total strength of  $R_{base} + R_{gating}$  (according to equation 3.1).

Assume that the proportion of active and inactive links are uniform throughout the knowledge base — that is, assume that the total node fanout  $b_{active}$  can be uniformly divided into two groups,  $b_{active}$  active nodes and  $b_{active}$  inactive nodes. On each step during the propagation process, activation propagating out on each individual link will be normalized by the total activation of all links:

$$(3.8) \quad 1 / (b_{inactive}R_{base} + b_{active}(R_{base} + R_{gating}))$$

where:

$b_{inactive}$	number of links per node with inactive relations
$b_{active}$	number of links per node with active relations
$R_{base}$	base propagation factor for active links
$R_{gating}$	additional propagation on active links

Equation 3.8 reduces to:

$$(3.9) \quad 1 / (b_{total}R_{base} + b_{active}R_{gating})$$

where:

$b_{total}$	total number of links per node (active and inactive)
-------------	--

Because of the nature of fanout decay and cutoff the same number of nodes will be visited,  $P_{initial}/T_{cutoff}$ . What does change are the point at which propagation quiesces. Activation on inactive links decays more rapidly than in traditional spreading activation, leading to an earlier cutoff:

$$(3.10) \quad P_{initial} \left( \frac{R_{base}}{b_{total} R_{base} + b_{active} R_{gating}} \right)^d \leq T_{cutoff}$$

Similarly, proportionally greater amounts of activation are carried along active links, causing propagation along those links to continue for a longer number of cycles:

$$(3.11) \quad P_{initial} \left( \frac{R_{base} + R_{gating}}{b_{total} R_{base} + b_{active} R_{gating}} \right)^d \leq T_{cutoff}$$

Thus, the maximum semantic depth reachable along active links in context-directed spreading activation is:

$$(3.12) \quad d' = \frac{\log P_{initial} - \log T_{cutoff}}{\log(b_{total} R_{base} + b_{active} R_{gating}) - \log(R_{base} + R_{gating})}$$

where:

**d'** the cutoff depth for active links in CDSA

As can be seen from this equation, the semantic distance reachable by context directed spreading activation along active links increases as the strength of relation gating increases (as  $R_{gating}$  increases with respect to  $R_{base}$ ) and as the proportion of active links to inactive links increases (as  $b_{active}$  decreases with respect to  $b_{total}$ ). In the limiting case, when  $R_{base}$  drops to zero, CDSA effectively reduces the branching factor of the knowledge base from  $b_{total}$  to  $b_{active}$ , potentially doubling or tripling the semantic distance reachable by spreading activation in some knowledge bases.

**Implications.** When relations are active in a knowledge base, context-directed spreading activation can take the same initial amount of perturbation  $P_{initial}$  and distribute it preferentially along active links, enabling it to activate nodes that would not otherwise be active (according to Equation 3.12). Furthermore, within the set of nodes that would normally be reached by traditional spreading activation, CDSA can increase the activation of nodes along relevant relations (Equation 3.11) and decrease the activation of nodes along irrelevant relations (Equation 3.10) thus effectively reducing the branching factor of the search space and more efficiently using the relevance evidence provided by the initial perturbation.

In a realistic knowledge base, this simplistic analysis will begin to break down. The fanout will change from node to node in the knowledge base; moreover, not only with the numbers of links from each node change, so will the proportion of active and inactive links. These differences in active links will furthermore not be static, but will change as

Parameter	Description	Typical Value
$R_{max}$	maximum number of active retrievals	3-5
$r_{specmax}$	maximum complexity of retrieval specification	not set
$r_{cuemax}$	maximum number of cues per retrieval	20-30
$C_{footprint}$	maximum number of nodes activated as result of a cue	1,000-10,000
$C_{limit}$	number of candidates matched per request per cycle	20-100

**Figure 3.8 Parameters of a Cost Control Policy**

perturbance propagates through relations, changing not only the active set of relations but also their relative activations on each cycle in the propagation process. Because of the complexity of these interactions, the best analysis of CDSA's effects on a real knowledge base, then, would be results of actual tests of the algorithm on real knowledge bases.

In chapters 7-9, I discuss in some detail empirical studies of CDSA on real knowledge bases for planning and information retrieval. These empirical studies show that CDSA is able to reroute activation productively in realistic knowledge bases, both enabling the retrieval of additional cases over traditional spreading activation in Nicole-MPA and enabling the improved recommendation of useful information in Nicole-IRIA.

### 3.5. Cost Control Policy

While CDSA provides some cost control features, cost control is not simply the province of the context-sensitive selection task or even the context-sensitive



asynchronous memory in isolation. For example, the matching specifications provided with a query could be arbitrarily large, taking a long time to match or producing a large number of query activation sources propagating knowledge into the knowledge base; alternatively, working memory could become overstuffed with items, leading to a large amount of priming activation; or too many requests might be outstanding at once, overwhelming both spreading activation and matching.

A cost control policy should be comprehensive, addressing all of the components of an agent that might affect the performance of the memory (Figure 3.8). Of course, a general-purpose system may be placed in environments in which any particular cost control policy might fail — either by allowing the system to consume too many resources or by limiting the system's resources to the point that it no longer effectively serves the needs of a task.

Keeping this limitation in mind, some of the features which must be addressed by an effective cost control policy for a context-sensitive asynchronous memory include:

- **Limits on effort expended on active retrieval requests ( $R_{\max}$ ):**

Limits are required to prevent the system from being swamped by too many retrieval requests. This can be implemented by upper bounds on the number of active retrieval requests and the amount of effort expended on any one retrieval request (the approach implemented in Nicole system discussed in Chapter 6) or

by a scheduling approach that tracks the total effort of all retrieval requests and allocates resources dynamically.

- **Upper bounds on complexity of retrieval specifications ( $r_{\text{specmax}}$ ):**

Unless some controls are in place, an individual matching specification can be arbitrarily complex to process and can swamp the retrieval costs of an entire system (Tambe et al. 1990). A variety of limits can be placed on matching specifications to prevent the costs of any one individual match from dominating the retrieval effort.

- **Upper bounds on the size of retrieval specifications ( $r_{\text{specmax}}$ ):**

In a system using context-directed spreading activation, limiting the complexity of matching specifications serves a second purpose: because query activation is generated from the knowledge items present in the specifications on a retrieval request, if the request can be arbitrarily large then the amount of activation spreading into the knowledge base, and hence the retrieval effort expended per cycle, can become arbitrarily large.

- **Upper bounds on the number of active cues ( $r_{\text{cuemax}}$ ):**

Similarly, the number of cues used as a source of priming activation must be limited. This can be accomplished by limits to the size of working memory (for implicit priming) or by limits to the number of cues allowed for any one retrieval request (for explicit priming). The implemented Nicole system adopts the second approach.

- **Upper bounds on the effort expended per cue ( $C_{\text{footprint}}$ ,  $C_{\text{active}}$ ):**

The previous two limits do not make any sense unless the amount of search effort expended per cue or specification is similarly limited. As discussed in the previous section, the amount of effort expended for any one source node is limited automatically by the theoretical properties of the context-directed spreading activation process.

- **Upper bounds on the effort expended per retrieval cycle ( $\text{cycle}_{\text{total}}$ ):**

Despite these limits, in an implemented system it is still possible to incur additional costs which could swamp the system — for example, in an extremely large knowledge base a high-level parent node may have millions of instances; the cost of computing the spreading activation connections along the “IS-A” link from that parent to its children even though the fan-out is so high that the propagation threshold will prevent activation from ever spreading out from that node!

Let us look at this last limit as an example. The implemented Nicole system does not provide a mechanism to deal with this potential problem for theoretical reasons — for example, given the right set of parameters on the CDSA equation and the right set of active relations, activation may still propagate through a node with millions of children because the CDSA equation effectively prunes them out — but a commercial-grade system would need to deal effectively with the time cost of performing this computation.

Similarly, a commercial-grade system must effectively deal with all the potential costs of retrieval, both at the theoretical level of “what nodes will we examine in the knowledge base” and at the practical level of “what combination of parameters will cause our garbage collector to choke the system at an inopportune moment?” Designing a cost control policy is thus part theory, part practice, dependent upon a thorough analysis of both the memory system, the software environment in which it is embedded, and the expected profile of retrieval requests the memory is likely to receive.

### **3.5.1. Applying Additional Cost Control to CDSA**

As just discussed, CDSA, like traditional spreading activation, provides a primitive cost control policy, but this policy by itself is not sufficient to fully control all the costs of retrieval.

The CDSA algorithms listed earlier simplify how activation propagates in an actual experience-based agent. As part of an effective cost control policy, the standard CDSA equation must be augmented with a number of additional parameters. First and foremost is the threshold for propagation to limit the amount of activation propagating from a single node. Other parameters include limits on the total number of source nodes or source queries, limits on the total number of propagating nodes, and damping factors to prevent activation from spreading too quickly.

Another potential cost in CDSA is sorting nodes in memory based on their activations to find the most active candidates. Sorting is an  $O(n \log n)$  process in the size of the

knowledge base and this cost may grow as knowledge bases become increasingly large. However, the number of nodes which are active are just the set of nodes which have recently been perturbed; the number of nodes differing in activation from their base values is therefore a simple linear function of the number of active cues at any one time, the number of nodes each cue can activate, the speed with which novel cues appear, and the speed with which activations decay to their base values. If the number of active cues has an upper limit, there will be a similar upper limit on the number of nodes with non-base activations. These nodes can be kept in an explicit active list, added and removed as a result of the imprinting and decay operations of the CDSA process; the active list can then be sorted by itself, placing an upper bound on the cost of finding the most active nodes in the knowledge base.

### **3.5.2. Order Analysis of Context-sensitive Asynchronous Memory**

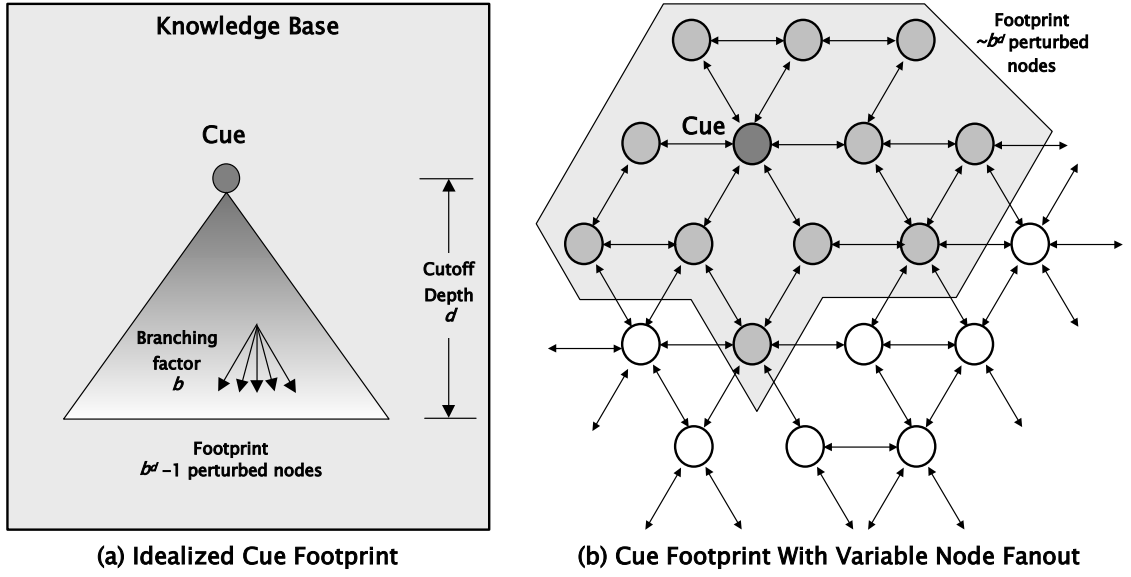
Computational complexity analysis is one method of examining the effectiveness of a cost control policy. A cost control policy should force a reasonable upper bound on the complexity of the processes operating under it. In this section, we will show that a combination of CDSA's innate cost controls with limits on the number and complexity of outstanding retrieval requests forms the core of a reasonable cost control policy.

**Assumptions.** In the model of context-sensitive asynchronous memory just presented, retrieval candidates are returned only as long as nodes in the network are active, nodes become active only when cues spread through the network, and cues spread

only when the network is perturbed. Perturbance is not a constant process; in the model implemented, perturbance is generated only when retrieval candidates are created, updated or refreshed, and optionally when new items are added to working memory. Ignoring the role of working memory for a moment, an complexity analysis of CDSA essentially reduces to how much propagation and retrieval effort is generated by the active set of retrieval requests.

Of course, the profile of retrieval requests differs on a task-by-task basis. For simplicity, we will make a number of additional assumptions. We will assume that retrieval requests are created at a regular rate; furthermore, we will treat updates and refreshes as new requests for the purpose of this analysis.

**Footprint of a Cue.** We will assume the *footprint* of each cue — the number of nodes ultimately activated as a result of perturbance spreading from a cue — is distinct; that is, each cue ends up activating a completely different set of nodes in memory. This assumption is unlikely to occur in any real task but provides a useful worst-case scenario for establishing an upper bound on the amount of effort expended.



**Figure 3.9 Footprint of a cue**

As discussed in the previous section, in a CDSA network propagation of perturbation will be cut off when the initial cue strength  $P_{initial}$  has been attenuated by fanout until it drops below the propagation cutoff threshold  $T_{cutoff}$  (Figure 3.9). Again, I assume a regular knowledge base with branching factor  $b$ . Simplifying out the effects of active relations, the cutoff of propagation will occur at a depth  $d$  where when the number of perturbing nodes attenuates the fanout to less than the cutoff strength. Recalling Equation 3.5 from the previous section, this cutoff occurs at:

$$(3.5) \quad P_{initial}/b^d < T_{cutoff}$$

where:

$P_{initial}$	average initial value of a cue
$b$	the branching factor of the knowledge base
$d$	the cutoff depth

$T_{cutoff}$  the propagation cutoff threshold

Solving for the number of nodes  $b^d$  that are thresholded out yields Equation 3.6:

$$(3.6) \quad b^d = P_{initial} / T_{cutoff}$$

As Figure 3.9(a) illustrates, the total number of nodes  $C_{footprint}$  which have been perturbed are all the nodes from the cue node to depth  $d-1$ , which is also an exponential function of the branching factor:

$$(3.13) \quad C_{footprint} = b^d - 1$$

where:

$C_{footprint}$  count of nodes perturbed by a cue

Solving for  $C_{footprint}$ , the total number of nodes perturbed from an initial cue is:

$$(3.14) \quad C_{footprint} = O(P_{initial} / T_{cutoff})$$

**Complexity of CDSA Propagation.** The total cost of spreading activation out into a footprint is potentially greater than the number of cues activated because the last set of nodes activated may generate a large number of outgoing quanta. Therefore, the total complexity of CDSA propagation from a cue is the number of nodes activated by each cue times the average fanout factor of nodes in the knowledge base:



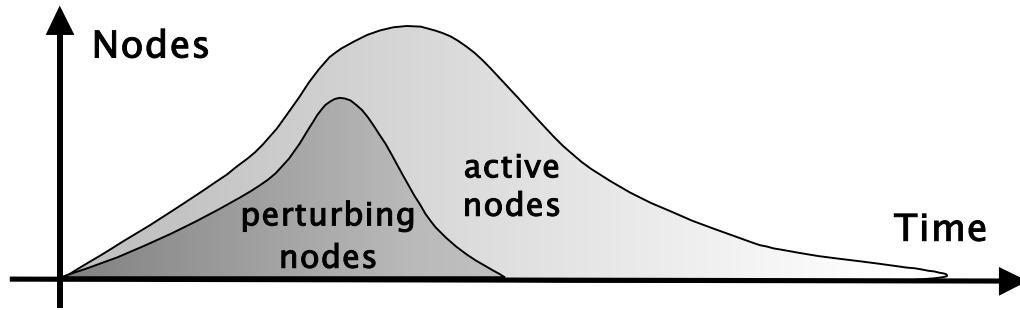


Figure 3.10 Profile of an Activation Wake

$$(3.15) \quad CDSA_{naive} = O(bP_{initial}/T_{cutoff})$$

where:

$CDSA_{naive}$       number of times the CDSA propagation equations are invoked in the naïve algorithm

The branching factor comes from the potential cost of computing the CDSA equation for each connection on each node, which is  $O(b)$  even if all of the quanta that would be generated would be immediately thresholded out. To reduce this cost, an automatic thresholding techniques can be employed to estimate the strength of the average quanta that would be produced as a result of propagation, preventing propagation if the estimated strength of the quanta is below threshold. (This reduces the accuracy of the implementation of CDSA but in practice has little or no effect on the order in which the knowledge base is searched). With this technique in place, the cost of spreading activation from a set of retrieval requests simply devolves to the number of cues times the number of nodes activated by each cue.

$$(3.16) \quad CDSA_{thresholded} = O(P_{initial}/T_{cutoff})$$

where:

**CDSA<sub>thresholded</sub>**      number of times the CDSA equation is invoked  
when automatic thresholding is in place

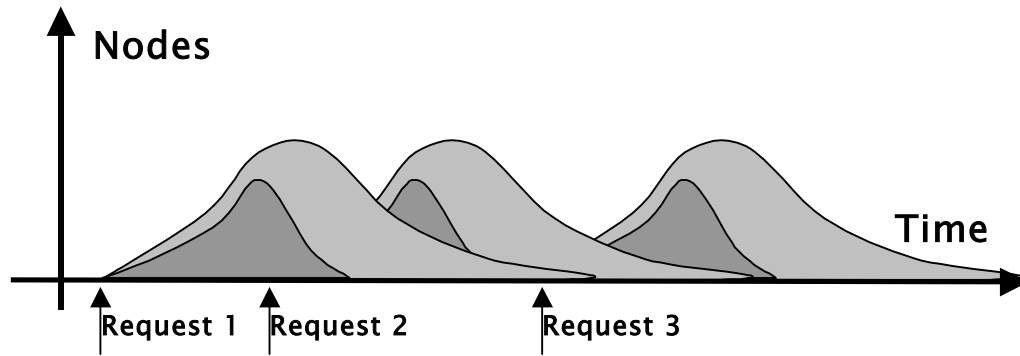
Note that each invocation of CDSA on a node still costs  $O(b)$ , but this cost is now only paid upon nodes with  $O(b)$  successor nodes during the expansion of the footprint, and thus for the purpose of order analysis the CDSA cost “disappears” into the  $O(b^d)$  cost of expanding the footprint itself.

**Imprinting and Decay.** Once a node has been perturbed, activation is imprinted upon it and takes some time to decay. The total cost of imprinting is simply the total number of nodes visited by propagating quanta (Equation 3.13).

Decay is more complex. If decay is computed over the entire knowledge base, the total cost of decay per cycle is a function of the size of the knowledge base:

$$(3.17) \quad D_{naive} = O(N)$$

where:



**Figure 3.11 Multiple Overlapping Retrieval Requests**

$N$	total number of nodes in the knowledge base
$D_{naive}$	total amount of effort spent per cycle decaying nodes

However, we can apply thresholds to decay just as we did to CDSA propagation. If we maintain an active list of nodes differing from their base activation by some threshold, the cost of decay per cycle devolves to the number of nodes visited by propagating quanta:

$$(3.18) \quad D_{thresholded} = O(C_{footprint}) = O(P_{initial}/T_{cutoff})$$

$D_{thresholded}$	total amount of effort spent per cycle decaying nodes on the active list
-------------------	--

However, nodes may take some time to decay. Assuming that the decay system cuts off decay at some activity threshold  $T_{active}$  and that a node begins with an activation  $C_{initial}$ , each node takes  $t_{decay}$  cycles to decay:

$$(3.19) \quad t_{decay} = O(\log T_{active}/C_{initial}D_{damping})$$

where:

$T_{\text{active}}$	minimum difference between base and current activation for a node to be considered “active”
$t_{\text{decay}}$	number of nodes above the $T_{\text{active}}$ threshold at any one time

In practice, the perturbation generated by a cue has a bell curve profile. Perturbation quickly propagates exponentially to more and more nodes and then diminishes as it begins to be thresholded out. A “wake” of activation follows the perturbing nodes, disappearing more slowly as activations decay to their base values (Figure 3.10). Since we cannot specify how much perturbation each node gets without knowing more details of the specifics of the network, for the purposes of our worst case scenario analysis we will assume that each node takes  $t_{\text{decay}}$  cycles to become inactive. Using this estimate, the upper bound on the number of nodes processed in the active list as a result of a single cue is just the footprint of the cue times the number of cycles nodes stay on the active list:

$$(3.20) \quad D_{\text{active}} = O(t_{\text{decay}} C_{\text{footprint}})$$

where:

$\text{Decay}_{\text{active}}$	total number of times nodes are processed in the active list as a result of perturbation from a cue
--------------------------------	---

**Cost of Retrieval Requests.** Thus armed with information about the effort expended on a single cue, we can now examine the cost of individual retrieval requests. A retrieval request will have  $r_{\text{cues}}$  cues. When that request is created, updated, or refreshed, each of

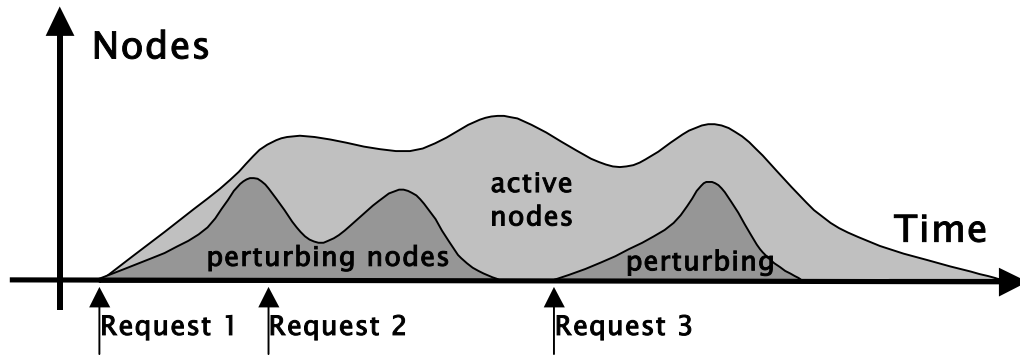


Figure 3.12 Overall Effort spent on Multiple Requests

those cues generates the same amount of effort ( $C_{footprint}$  nodes perturbed and  $C_{active}$  nodes in the active list), meaning the total amount of effort generated as a result of refreshing a request is:

$$(3.21) \quad r_{footprint} = O(r_{cues} C_{footprint})$$

$$(3.22) \quad r_{active} = O(r_{cues} D_{active})$$

where:

$r_{cues}$	number of cues per request
$r_{footprint}$	number of nodes perturbed as a result of refreshing a request
$r_{active}$	number of nodes activated as a result of refreshing a request

However, multiple retrieval requests may be active at the same time (Figure 3.11, 3.12). Assume that retrieval requests are created with a regular frequency  $f_{request}$ . If retrieval requests overlap significantly, the amount of effort spent on each retrieval cycle

contains contributions from the life history of several requests (Figure 3.13). If we average the amount of retrieval effort expended over time, the effort expended per retrieval cycle is just the amount of effort expended per request multiplied by the frequency.

$$(3.23) \quad cycle_{footprint} = O(f_{request} r_{footprint}) = O(f_{request} r_{cues} C_{footprint})$$

$$(3.24) \quad cycle_{active} = O(f_{request} r_{active}) = O(f_{request} r_{cues} t_{decay} C_{footprint})$$

where:

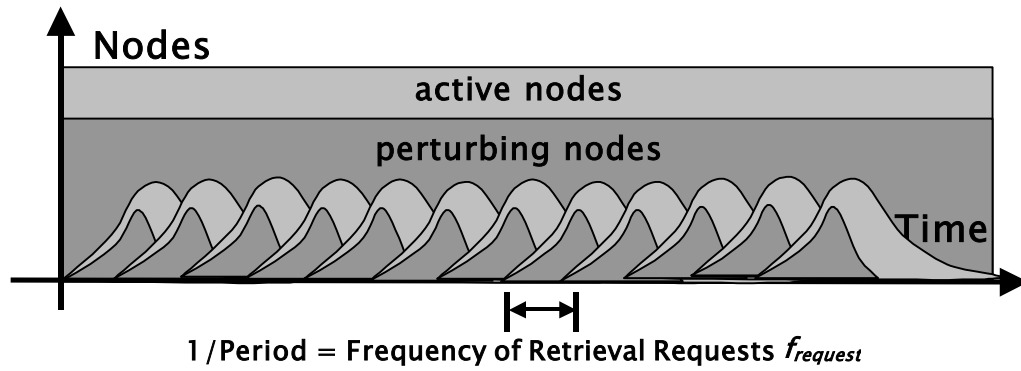
$cycle_{footprint}$	number of nodes perturbed per cycle
$cycle_{active}$	number of nodes on active list per cycle

**Imposing Cost Controls.** As equation 3.23 and 3.24 show, the only important factors in the cost of retrieval per cycle are the number of requests per second and the number of cues per requests — all other factors are constant parameters and do not contribute to the order of complexity of the computation. Suppressing all these constant factors:

$$(3.25) \quad cycle_{total} = O(f_{request} r_{cues})$$

where:

$cycle_{total}$	total amount of perturbation and decay per cycle
-----------------	--



**Figure 3.13 Average Effect of Regular Periodic Retrieval Requests**

Thus, to limit the effort expended per cycle a cost control policy must limit both the maximum number of retrieval request creations and refreshes per cycle and the number of cues per request. Since the maximum frequency of requests would be to refresh all outstanding requests on each cycle, the former limit can be imposed by placing upper bounds on the number of requests per cycle.

$$(3.26) \quad cycle_{max} = O(R_{max}r_{cuemax}) = O(C_{max})$$

<b>cycle<sub>max</sub></b>	maximum total amount of perturbation and decay per cycle
<b>R<sub>max</sub></b>	maximum number of outstanding retrieval requests
<b>r<sub>cuemax</sub></b>	maximum number of cues per request
<b>C<sub>max</sub></b>	maximum number of outstanding cues ( = <b>R<sub>max</sub>r<sub>cuemax</sub></b> )

**Limiting Other Costs.** Spreading activation is not the only cost involved in retrieval. On each cycle, the active list must be sorted and the top candidates compared against the specifications of the outstanding retrieval requests. The cost of sorting is just  $O(n \log n)$  where  $n$  is the number of nodes on the active list:

$$(3.27) \quad O(cycle_{active} \log cycle_{active})$$

This cost is already limited by the portions of the cost control policy that limit the size of the active list.

Some fixed number  $C_{limit}$  of the most active nodes on the list will then be matched with each retrieval request's specifications. The total complexity of matching is thus the number of retrieval requests times the cost of performing each matching operation times the number of candidates to match:

$$(3.28) \quad O(R_{max} r_{specmax} C_{limit})$$

where:

$R_{max}$	maximum number of outstanding retrieval requests
$r_{specmax}$	maximum complexity of matching a retrieval request
$C_{limit}$	limit on the number of candidates returned per cycle

This cost can be controlled by limiting the complexity of each retrieval request's specifications and the number of candidates matched per cycle.

**Implications.** Given a cost control policy placing bounds on the total number of retrievals, cues per retrieval, complexity of specifications for retrieval, and employing fanout thresholds, decay thresholds and automatic cutoffs for spreading activation, all the variable factors in retrieval are controlled in terms of constant parameters which do not



contribute to the order of complexity of retrieval. Factoring these constant parameters out, the cost of each retrieval cycle, including context directed spreading activation, decay, sorting, and matching, is bounded at:

$$(3.29) \quad \textit{cycle}_{controlled} = O(1)$$

where:

**cycle<sub>controlled</sub>** cost of a retrieval cycle under a cost control policy

### 3.6. The Experience Store

As described, a context-sensitive asynchronous memory retrieval system could function as the memory system for a single-task agent, working in parallel with the reasoner and using the features of the task and the environment to attempt to provide information relevant to the current task context. But the context-sensitive features of this memory architecture do more than just find information relevant to a single task; after all, a special-purpose memory retrieval mechanism could be specifically designed to provide efficient memory retrieval for a single purpose.

A context-sensitive memory retrieval system provides a way for a single agent to perform multiple tasks, automatically focusing its retrieval effort on the portion of its knowledge base relevant to the task. This focusing can even occur implicitly, without the conscious effort of the agent: if the right features are available in the environment and the right structure in the knowledge base, a context-sensitive memory can focus its retrieval

effort on information relevant to tasks the agent *could* or *should* be performing in that environment, even if the agent has not yet decided what task it should be performing at that given moment.

In order for the promise of this approach to be realized, the agent must have a knowledge base capable of representing knowledge about a wide variety of tasks, along with the connections of those tasks to the environments in which they occur; the agent must furthermore store all of this information within a unified knowledge store, so that if the agent encounters a novel environment and cannot decide what tasks it should be performing, the memory can make the appropriate connections and provide information to the agent to help it select a course of action. This idea of a general, unified repository for all knowledge in an agent is the essence of an *experience store*, the core concept from which experience-based agency derives its name.

An experience store must satisfy three key properties. First and foremost, of course, the store must permit the operation of the selection task of a context-sensitive asynchronous memory, which in practice means that the store must support context-directed spreading activation. Second, the store needs a rich and expressive knowledge representation capable of supporting the knowledge representation needs of a wide variety of tasks. Finally, the store must support the unified storage and access of multi-task knowledge, enabling access to information in the store to be driven up from the content of the features of the environment and not solely down from the knowledge needs of particular tasks.

Satisfying the first two of these two requirements is a property of the knowledge representation language of the experience store; satisfying the third is a question of a policy of access. Let us examine these issues in turn.

### **3.6.1. Constraints Imposed by Context-Sensitive Asynchronous Memory**

In order to support a context-sensitive asynchronous memory, a knowledge representation requires a variety of features. For example, to enable the iterative, incremental search memory must be divided into separable components which can be examined each in their turn; to ensure that the search of those items can proceed in a way relevant to the current context the memory must have features that associate the contents of its knowledge base with the features of that context; and so on. At the very least, a knowledge representation for a context-sensitive asynchronous memory should include the theoretically significant features found in the experience store implemented as part of this research, including *nodes*, *links*, *bidirectionality* and *grounds*:

- **nodes: units of knowledge:**

For an asynchronous memory to work, the knowledge base must be divided into individual pieces of knowledge, or units. Each node is separate unit, which can be individually retrieved and examined — or passed over and ignored — by the memory process as it searches the memory. In the language of (Maida & Shapiro 1982), nodes in an experience store are primarily intensional objects, representing concepts rather than items in the world.

- **links: associations between knowledge units**

For a context-sensitive memory to work, all connections, whether logical or associative, that exist between each individual piece of knowledge in the knowledge base must be accessible, enabling the memory to make connections from item to item. A link may have an intrinsic weight, or the weight of links may be determined by system-wide parameters.

- **bidirectionality: the symbolic nature of associations**

The entire point of a context-sensitive asynchronous memory is that in a general purpose agent we cannot pre-specify the set of indexes we will need to retrieve an arbitrary piece of knowledge in an unknown future situation. More particularly, when the knowledge base learns that two pieces of knowledge are connected, it cannot determine in advance which piece of knowledge should serve as the cue for the other. Therefore, each link between pieces of knowledge in the knowledge base should be a *bidirectional* link for the purposes of determining what knowledge is related to the current context. In semiotic terms, links in an experience store are not signs, in which one thing refers to another thing, but instead symbols, in which two things refer to — and can evoke — each other.

- **grounds: connection between knowledge units and perceptual data**

To be effective, a context-sensitive asynchronous memory must be able to make connections between the current context and knowledge in the knowledge base. The key information in that context that will enable the memory to find the right

information at the right time might be a piece of knowledge in the agent's reasoning state, but it could just as easily be subtle perceptual cues in the agent's environment. Therefore, knowledge in the experience store needs to ultimately be connected to the features likely to be found in the content; in philosophical parlance, the knowledge in the knowledge base needs to be *grounded* in the perceptual features of the environment (or the internal data structures used by the agent's quick-and-inflexible reasoning modules). In the language of (Maida & Shapiro 1982), nodes in an experience store are primarily extensional connections, connecting intensional concepts with external real-world objects.

Beyond these basic requirements, a memory that employs context-directed spreading activation as its selection task must support additional features: activation levels, link weights, and node-labeled links or “reified relations”:

- **activation levels:**

Each node in the knowledge base must have an “activation level” which represents the memory's current estimate of the relevance of the node to the current context as computed by the CDSA algorithm. Each node has a “base” level of activation, which may be determined by a system-wide defaults or which may be unique to a particular node, promoting or demoting its position on the candidate list. The most active nodes in the system constitute the current set of candidates that the selection task will present to the asynchronous memory retrieval system.

- **link strengths:**

Activation propagates from source nodes out along links to other nodes in the experience store, changing their activation levels and ultimately the set of candidates the selection task presents to the retrieval monitor. This propagating activation, or “zorch”, travels differentially over links based on their intrinsic strength, normalized over the set of all associations attached to a node. Like the base activation level, links may have a default activation or may accumulate increased or decreased strength through learning.

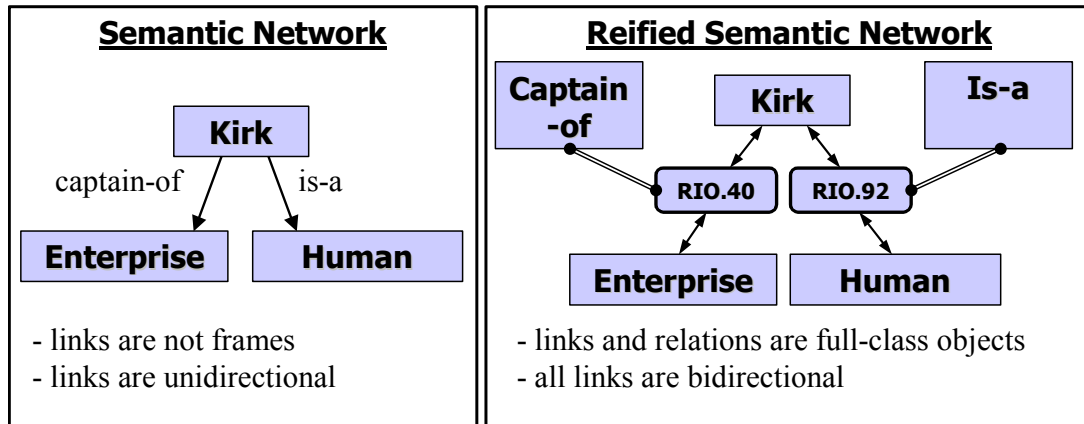
- **node-labeled links:**

Gated spreading activation requires more than just links between nodes: the links must be labeled by other nodes whose activity or inactivity affects the strength of the link and hence the way that activation propagates through the knowledge base. Having each link instantiate a “reified” relation node is one way to achieve this condition.

### **3.6.2. Expressive Power**

When these requirements are combined — nodes, grounds for nodes, links, labels for links which are themselves nodes — they describe a class of knowledge representations which fall firmly within the tradition of research on semantic networks.

**Semantic Networks.** Semantic networks are a graph representation for knowledge in which individual pieces of knowledge are stored on the nodes of the graph and the



**Figure 3.14 Properties of a Reified Semantic Network**

relationships between pieces of knowledge are stored on links between nodes (Figure 3.14a). Semantic networks are an expressive formalism which can represent a wide range of knowledge, including natural language (Quillian 1968, Miller et al. 1993), and are functionally isomorphic to frame systems (for an example of similarities see Lenat & Guha 1990). Many algorithms exist for searching semantic networks; popular ones include spreading activation and marker passing algorithms which propagate symbolic or numerical counters from node to node in the network along the links.

**Reified Relations.** We have already discussed the significance of a unified, grounded, bidirectional semantic network for supporting a context-sensitive asynchronous memory; now let us examine node-labeled links and what kind of representational power they support. To achieve node-labeled links, an experience store can *reify* (represent as explicit knowledge objects) relationships between concepts using *relation nodes*. Relation nodes are first class objects in the experience store which are treated like any other piece of knowledge in the knowledge base and can themselves

participate in arbitrarily complex relationships (Figure 3.14b). This approach, similar to that used in knowledge representation systems such as KL-ONE (Brachman 1985), KODIAK (Wilensky et al. 1988), AQUA (Ram 1989), and CYC (Lenat & Guha 1990), enables not only the context-directed spreading activation algorithm but also complex reasoning about relationships between objects (e.g., Maida & Shapiro 1982).

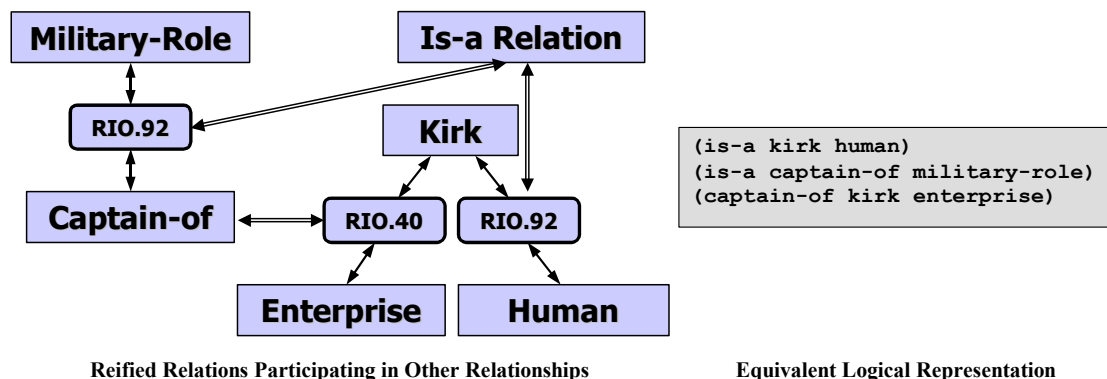
**Mapping to Semantic Networks to Logic.** A semantic network defined in terms of nodes and links can be re-written in a logical formalism, where labels of the links are mapped to logical entities such as predicates, nodes are mapped to constants, and relation instance objects are mapped to propositions. In the example in Figure 3.14, the relations are “is-a” and “captain-of” map to INHERITANCE and CAPTAINCY predicates, and the nodes “kirk”, “enterprise”, and “human” map to constant terms in specific propositions using those predicates. Written as a set of propositional clauses, the semantic network in the figure can be represented as:

**(is-a kirk human)**

**(captain-of kirk enterprise)**

The precise mapping used would depend on the way in which the semantic network was used to represent knowledge, including both the content of the representation and its expression in the structure of nodes and links. Because nodes and relations exist in an inheritance hierarchy, this mapping can be extended to represent functions, predicates, quantifiers and operators as different classes of relations, and to represent constants and variables as different classes of nodes; instances of relations would then naturally map to





**Figure 3.15 Higher-Order Relations in a Reified Semantic Network**

compound terms and propositions. For an example of a similar mapping of a different kind of semantic network network to first order propositional logic, see (Schubert 1991) and for further discussion see (Rich & Knight 1991).

**Reified Relations and Higher Order Logics.** A “traditional” semantic network or frame system, in which the labels on links are not themselves knowledge objects, thus has the expressive power of first-order propositional logic, because a predicate in this formalism can never appear as the subject of another predicate. Adding reified relations introduces additional representational power. Because reified relation nodes are knowledge objects, they can be connected to the rest of the knowledge base by nodes and links in arbitrary ways. For example, the “captain-of” relation in the earlier example might be linked by an is-a link to a more abstract relation, such as “military-role” (Figure 3.15). As a consequence of this promiscuous linking, when a reified semantic network is translated into a logical formalism a predicate of one proposition may be the subject of another proposition. In our example, “**captain-of**” serves both as a predicate of a

proposition in “**(captain-of kirk enterprise)**” and an argument of a proposition in “**(is-a captain-of military-role)**”

Because of this property, a semantic network with reified relations has the expressive power of Nth-order logic: any predicate can serve as the subjects of a higher-order predicate (or even as a subject of itself). While there are outstanding problems in the logic of higher order logic and applications with a logical model incompatible with higher-order logic need not use these features, this kind of higher-order representation has shown itself to be useful for representing complex linguistic concepts (e.g., Maida & Shapiro 1982) and was successfully used to support understanding science fiction stories in the ISAAC system (Moorman 1997).

**Task-Specific Logics.** It is important to note that the knowledge representation for an experience store is most concerned with the structure knowledge and not its specific semantic content or behavior. This holds true both at the level of theory and the level of implementation.

Issues handled by many approaches that focus on pure knowledge representation — such as truth maintenance, constraint languages, languages for constructing propositions, structuring of concepts into conceptual graphs, and so on — are “task specific” as far as the context-sensitive asynchronous memory approach is concerned.

The above analysis of the representational power of grounded reified relations, coupled with actual experience using that representation to build complex knowledge stores in the ISAAC, Nicole-MPA and Nicole-IRIA systems, strongly suggests that it is sufficiently powerful to represent whatever structure of propositions, graphs, or truth maintenance is required for a specific task, as long as that task can be structured to perform whatever additional processing is necessary to ensure that the portions of the experience store it uses conforms to the constraints of that application.

In return, any application task that shoehorns its representation into this framework will be able to fully exploit the retrieval power of a context-sensitive asynchronous memory.

### **3.6.3. Creating a Unified Store of Knowledge**

The point of a unified store of knowledge is to enable one memory process to exploit arbitrary environmental regularities. Given a specific question for an individual task, we can generally devise a separate memory system with a task-specific indexing structure that will enable us to quickly find appropriate answers to the question.<sup>10</sup> When questions become poorly specified, a system performs more than one task, or most particularly when a single system faces multiple, possibly overlapping tasks in a complex, information-poor environment, we no longer have the luxury of routing all questions to

---

<sup>10</sup> For an example of an alternative view of how to structure a multi-task memory system, see (Peterson et al. 1994, Goel et al. 1996).

single-purpose memories. The context-sensitive asynchronous memory approach assumes that a regular environment will contain many features that an agent can learn from to focus its effort on the appropriate parts of its memory, creating the effect of many single-purpose memories as a result of its learning about the structure of knowledge about the tasks it performs and the conditions in the environment in which it encounters those tasks.

The experience store is this unified store of knowledge: a single data structure which stores and provides access to knowledge for multiple tasks. In principle there is no challenge to creating an experience store: simply put all the data in the agent into a single representation which satisfies all of the functional constraints listed earlier. In principle, the designer of an intelligent system can create an experience store using any kind of representation language of sufficient power, simply by playing by all of the rules.

In practice difficulties can arise at the program level. For an experience store to function it must hold all agent knowledge in a single data structure; all connections between pieces of knowledge must be two-way, and moreover must themselves be pieces of knowledge for context-directed spreading activation to work; and for primed spreading activation to work the experience store must provide a way to find any frame that references program-level data using that data as a key. If any of these features are not enforced — if links are created that do not instantiate relationships, if two-way connections are not created between pieces of knowledge, or if a frame references data

which cannot be used to find the frame — then the performance of the memory will degrade, as the results discussed in Chapter 7 show.

For an experience store to be effective, then, it cannot rely on the good behavior of programmers of the system; instead, its interface must enforce the key properties necessary to serve the needs of a context-sensitive asynchronous memory. As already hinted at, these properties include automatic grounding, automatic reification, automatic bidirectionality, and unified access.

- **automatic grounding:**

The first requirement for an experience store is content addressability: we should be able to use any “non-knowledge” content referenced within the network — perceptual data, program-level numeric or string data, or handles to opaque data structures generated quick-and-inflexible reasoning modules — to find the parts of the network that refer to them. This content-addressable property could be achieved through extensive knowledge engineering, but can be achieved more directly by “automatic grounding”: the application programming interface of the experience store should allow connections to be made between knowledge and data, but should then wrap that data in “placeholders,” or knowledge objects which encapsulate external data, to ensure that the knowledge is grounded in the real-world situations in which it will occur. Placeholders are full-fledged knowledge objects, indexed by the data that they encapsulate, that stand in for that data in semantic relationships with other knowledge within the experience

store. Automatic creation of placeholders, along with indexing of placeholders by data, ensures that the knowledge content experience store is accessible using features likely to be found in the environment.

- **automatic reification:**

Just as the experience store must automatically ensure that all data referred to by knowledge is encapsulated within knowledge, the experience store must ensure that all relationships between knowledge are encapsulated within relationships. Ontological engineering and knowledge representation are difficult tasks; if the API of a knowledge representation does not explicitly prohibit it, it is all too easy to create slots on frames or links between nodes which do not instantiate any existing relationship. One way to enforce this is not to allow the creation of a links between two nodes that does not instantiate an existing relation; another, recommended by this research, is to create relationships between nodes on demand when links are created.

- **automatic bidirectionality:**

Just as it is easy to omit the creation of a relation node for every slot if the knowledge representation does not demand it, it is difficult to ensure that every link between two pieces of knowledge is automatically maintained in both directions. A knowledge representation for an experience store should maintain bidirectionality automatically, either by enforcing two-way links in a semantic

network or by creating inverse slots on frames referred to in the slots of other frames.

- **unified access:**

Finally, all of these automatic operations must occur across the whole knowledge base as a unit. It does no good to create placeholders, reified relations, and bidirectional links on the fly if the knowledge base is partitioned in such a way that creating a placeholder or relation in one part of the knowledge base does not affect the content-addressability or available relation sets in another. This does not suggest that artificial relationships be created between knowledge that is not related; instead, it simply notes that for context-sensitive retrieval to be effective across a multiple task knowledge base, when a conceptual relationship or piece of data appears in two tasks then the experience store should automatically maintain the connections between that relationship or placeholder and the knowledge of those two tasks.

### **3.7. Getting Answers with Context-Sensitive Asynchronous Memory**

Given a context-sensitive asynchronous memory, and an experience store which supports and enforces all the features that this memory needs to be effective, how can we use the combination of these two to effectively get useful answers to difficult questions?

A context-sensitive asynchronous memory process is best used as part of a process in which an agent explores its environment, its task and its memory iteratively and interactively, using feedback in one area to make progress in another. Specific instantiations of this approach will be reviewed in the discussion of the Nicole-MPA planning and Nicole-IRIA information retrieval systems in Chapters 8 and 9, but a brief sketch of the context-sensitive asynchronous memory approach to finding answers to difficult questions involves seven overlapping stages — seeding, analyzing, questioning, evaluating, performing, updating, and learning:

- **seed the experience store:**

The context-sensitive asynchronous memory cycle usually begins with an experience store that has some “seed” of knowledge already stored in the experience store. This kernel of knowledge, seeded into the knowledge base by past experience or design, will potentially provide answers to the agent’s questions if the questions can be specified well enough or enough resources can be devoted to the search. For example, in a planning system the experience store could be seeded with a library of planning cases, either explicitly by the designer or accumulated by the system itself over time. In an information retrieval system the experience store could be seeded with an initial set of document pointers, again explicitly by the provider of the system or accumulated from past retrieval sessions. Alternatively, the experience store could be seeded with information which would facilitate retrieval: for example, an information retrieval system’s



experience store could be seeded with a thesaurus or content theory to improve retrieval of relevant documents.

- **analyze environment and task:**

The agent then analyzes its environment and the task(s) it is presented with to prepare for performing its task. This corresponds to the case-based reasoning stage of situation assessment (Owens 1989, Kolodner 1993), in which the agent elaborate its description of the situation prior to search of the case library. In experience-based agency this elaborated description is used to prime the context-sensitive memory process. Here, the experience store and context-sensitive memory work together: the unified experience store attempts to provide to the context search system “hooks” into its knowledge that will enable the memory system to exploit environmental cues to find the relevant set of knowledge even before the agent asks its questions. For example, planning cases stored in an experience store should expose the content of their initial and goal conditions to make context-sensitive retrieval based on environmental cues possible; document pointers should be labeled with the index terms or concepts which are likely to be found in queries.

- **ask questions, watch for answers:**

If the agent does not have the information “on hand” to solve the problem it can try a more exhaustive search of its experience store for potential solutions by initiating a retrieval request with the asynchronous memory system. For example,

the planning system can continue to search for cases; the information retrieval can continue to search for relevant document pointers. The first cases retrieved may not be suitable for application to this individual plan; the first documents retrieved may not accurately reflect the user's information needs.

- **perform task, examine environment:**

What the asynchronous memory here provides the agent is the ability to continue to work without waiting for a response to its questions: it can instead continue to elaborate the description or perform its task while the asynchronous memory searches for relevant information. As it does so, the context-sensitive portion of the memory guides the asynchronous search towards potentially relevant parts of the experience store. Further processing of a plan may reveal new conditions or constraints which can cue the retrieval of relevant plans; additional processing of retrieved documents, queries, or user responses to retrieved documents and queries can provide cues to suggest other documents.

- **evaluate answers, incorporate useful information:**

As answers are returned, the agent needs to evaluate their usefulness and relevance with respect to the current state of its processing. A spontaneously returned answer from the asynchronous memory may be useful and relevant, meeting the information needs of the task in a timely way; alternatively the answer may be useful but no longer relevant, or relevant but not correct or useful.

- **update questions, continue processing task and environment:**

Based on this relevance judgement, the agent may refine the questions it is asking to further guide the memory, or otherwise update its specification of its information needs (e.g., Owens 1989). In parallel with this, of course, the context-sensitive memory can continue to take advantage of additional information generated by the agent's reasoning or events in the environment to guide search. For example, elaboration of the problem may change either the cues or the specifications for the planning cases that may need to be retrieved; as users process initial retrieved documents they may generate new keywords which can further refine the system's understanding of their information need and improve their retrieval.

- **get more answers:**

The ultimate goal of using asynchronous and context-sensitive memory is that the result of additional effort spent on retrieval and additional data provided by contextual feedback will be additional answers which more accurately or usefully answer the agent's questions. Hopefully, cues generated during planning will aid the memory to find the most on-point prior planning cases; the cues generated from processing documents and users inspecting documents will remind the system of the most on-point document pointers in the store.

- **repeat until satisfied:**

However, the agent may not be initially satisfied, or may want a comprehensive

search of its memory, or may simply be difficult to please, so asynchronous, context-sensitive search of memory should continue until the agent signals that it is done. While major portions of a planning problem remain open, it may be profitable to continue to search for cases; as long as a user continues browsing retrieved documents it may be profitable to continue to search the experience store for relevant document pointers.

- **update knowledge base (continuous):**

When the agent signals a request has successfully or unsuccessfully completed, the asynchronous memory should learn from this experience, updating the weights and structure of the knowledge base to improve future retrievals. In reality, learning and updating happen continuously in a system with an experience store: as the agent performs any reasoning task which generates new information or modifies old information, the experience store automatically generates or updates the connections between relevant pieces of knowledge, enabling future context-sensitive asynchronous memory searches to more readily take advantage of whatever contextual information can be generated from a task or found in the environment. Any time an experience-based agent learns a new planning solution or processes the content of a new document, the experience store is automatically updated with the information it needs to seed the next round of memory retrieval.

As can be guessed from this description, taking full advantage of a context-sensitive asynchronous memory places constraints on an agent: a reasoning task which treats a

context-sensitive asynchronous memory as a pure subroutine call — posting a request for information and then freezing while it awaits a response — cheats itself of the processing cycles it could be exploiting to solve its problem and cheats the memory system of the contextual feedback it might use to guide its search.

### **3.8. Applicability of the Approach**

The preceding section discussed *how* to exploit a context-sensitive asynchronous memory system. A more important question is *when* to exploit a context-sensitive asynchronous memory system — under what conditions is the approach expected to provide benefits? Answering this question requires examining the assumptions of the approach in more detail.

One of the central assumptions of the context-sensitive asynchronous memory approach, if not *the* central assumption, is that an agent can exploit the regularity of its environment to improve its memory retrieval performance. More specifically, in a regularity-rich environment, an agent that accumulates experience can improve its ability to recall relevant experiences by exploiting *context* — information in its current situation not explicitly specified as part of its information needs. For example, in a planner attempting to retrieve cases based on a problem description, context could be derived from information generated during an attempt to find a solution to the problem; in an information retrieval system attempting to find resources relevant to a query, context could be derived from user browsing behavior.

This claim is derived from the more general principle that using context, as defined here, can inform and improve action. The context principle can be observed in the priming effect in human memory retrieval (e.g., Quillian 1962, 1966, 1967, Collins & Loftus 1975, Anderson 1983a, 1983b, Klimesch 1994; also see Greene 1992) but applies just as strongly to task and user interface of many common computer-related tasks (for discussion, see Nickerson 1977, Rissland 1984, Bolt 1985). Computer systems that exploit context are being developed and tested in a variety of applications (e.g., Kolodner & Penberthy 1990, Simina 1999; also see the Contextual Computing Group (<http://www.gvu.gatech.edu/ccg/>), the Context Aware Computing Group (<http://www.media.mit.edu/context/>) and so on).

The key novel element that context-sensitive asynchronous memory provides is an automatic task-independent mechanism for exploiting this knowledge for memory retrieval. However, the context-directed spreading activation algorithm, and by extension the context-sensitive asynchronous memory approach, require both an “appropriate” environment and an “appropriate” context to improve upon more traditional memory retrieval approaches. To place meat upon these bones we must unpack more explicitly what these environments, contexts and approaches are — in other words, we must specify more precisely the approaches context-directed spreading activation is competing with, the circumstances under which traditional approaches fail to perform as well as desired, and the environmental structure and contextual information that context-directed spreading activation needs it to improve upon its competition.

### **3.8.1. Traditional Spreading Activation and its Limits**

Context-directed spreading activation competes most directly with noncontextual spreading activation approaches. In fact, in the absence of context, context-directed spreading activation degenerates to a simple spreading activation approach.

Spreading activation is a weak method, relying on history, not logic. In other words, spreading activation sensitive not to the logical properties of the items in the knowledge base but instead to the structure of the knowledge base itself, which is as much a property of the historical accidents by which the knowledge base became to be populated as it is of the logical structure of the items themselves.

When the knowledge base is full of “irrelevant distractors” — items which match the cues of a query but fail to match its specification for knowledge — activation is as likely to spread to items which have similar structure and connections to relevant items in the knowledge base yet which do not match the specifications of the query.

For example, if the goal of retrieval is to retrieve a case relevant to a particular problem in a planning domain, adding large numbers of cases in other, irrelevant planning domains should be expected to degrade the system’s ability to retrieve relevant cases. Empirical tests, discussed in later chapters, verified this: adding new planning domains with large numbers of cases degraded or eliminated the system’s ability to retrieve relevant planning cases.

### **3.8.2. How Context-Directed Spreading Activation Uses Context**

The context-sensitive approach uses a supplied context — additional information not part of a query, either culled automatically from the environment or explicitly provided by the reasoning task — to alter search in a heuristic fashion. Context appears in the form of nodes already active in the knowledge base — primes, in psychological terms — alter how future activation spreads in a priming-based weighted propagation process.

As discussed earlier, there are two facets to context-directed spreading activation: primed spreading activation, in which activation spreads more efficiently to items which are already activated above some threshold value, and gated spreading activation, in which activation spreads more efficiently along links mediated by relation nodes which are active. Primed spreading activation makes it easier to (re)consider concepts which we have recently thought of; gated spreading activation makes it easier to channel activation along routes relevant to the kinds of relationships between information items that we have been considering. So context, as far as the mechanisms of context-directed spreading activation are concerned, consists of primes --- active nodes --- which change how activation spreads.

### **3.8.3. Types of Contextual Information**

A wide variety of things can be used as primes: objects in the current situation, high-level concepts in the current situation, potential targets of the search, or relationships we expect might hold between objects, concepts, and targets.



For example, in the planning domain an object might be a location or an actual physical object mentioned in the problem description or seen in the environment. A concept might include things like a plan or a class of objects which can be logically derived from the structure of the problem. A target might be a planning case recently discussed. Relationships might include the class of links between a case and the initial conditions of the problem it solves or between the case and its goals. Of course, a context can be good or bad: an object seen might not appear in any relevant case; a concept thought of may not sufficiently discriminate existing cases; a recently seen case might not be relevant; and a relationship thought of might be relevant to a different domain.

In the most extreme case, the set of “priming” concepts encountered in the environment or the reasoning trace might not be relevant to the current task, problem or history of stored cases. For example, if a fire drill sounded in the middle of an epistemology class, students’ recent experiences would not necessarily be helpful in recalling relevant cases.

#### **3.8.4. Available Sources of Context**

We now return to the principle that in a sufficiently regular environment an agent can exploit the information available to it in the current situation to improve its retrieval performance. What information can we expect to find in the current situation which can be exploited in this way?

We cannot count on a kindly oracle floating nearby to conveniently drop the name of a relevant case just before a problem arrives. Similarly, the general concepts of a domain are a part of every case in the problem domain and (in general) do not usefully discriminate among them, nor are they useful retrieval targets in and of themselves given that they are derivable from the structure of the problem.

However, two classes of information are immediately and always available to an agent: the objects and conditions of the world in which the agent finds itself (or which it finds within the description of the problem it has been given) and the relationships relevant to carrying out its task.

The first two sources of context are obvious: objects and conditions are simply the features of the world that the agent observes in its environment or hypothesizes during the course of its reasoning. The third source of context, the relationships relevant to carrying out a problem, is inherent in most reasoning tasks but rarely made explicit. As a reasoning task processes objects or conditions related to its task, it must specify the components of those knowledge items it wishes to inspect, as well as their connections to other knowledge items it wishes to process. In an experience store, where all knowledge is represented through nodes and their connections, connections between knowledge items take the form of links, and specifications of those connections by reasoning tasks take the form of reified relations. Taken together, a knowledge item and a reified relationship specify links between knowledge items which can be used to access other parts of the knowledge base necessary for a reasoner performing a task. For example,

finding the initial conditions of a plan in an experience store requires not only having a concept for the plan, it also requires knowing the reified relationship which connects plans to plan initial conditions. Finding the title words of a document requires not only having a document pointer concept, it also requires knowing the reified relationship which connects words and documents. These reified relationships themselves are concepts in the experience store, and can serve as cues for activation just like plans, initial conditions, documents and words can.

Even if an agent does not ask a question, the combination of activating the objects that we see (a natural consequence of processing information about the world) and activating the relationships relevant to a problem (a reasonable consequence of attempting to solve a problem in a domain) will cause activation to flow from those activated objects towards cases in the problem domain which are potentially relevant, in theory making it easier to retrieve those objects when the agent does ask a question. As discussed earlier, examples of tasks in which this kind of problem analysis goes on include situation assessment in case-based reasoning (Kolodner 1993) and elaboration in textual processing (Lange & Wharton 1994).

### **3.8.5. Profile of Environments In Which CDSA Will Provide Benefits**

CDSA should provide benefits over traditional spreading activation when the cues that are available to act as context focus spreading activation between the questions that an agent asks to the answers that it desires. This analysis indicates that this is most likely

to occur in environments in which objects and conditions in the world are usefully connected to potential solutions stored in the knowledge base by relationships that can be derived from the structure of the task, and in which the cues found in the environment are correlated in some way with the solution.

In other words, CDSA is most appropriate when the process of assessing the task (considering and thus activating concepts relating to its logical structure) can route activation from the content of questions towards sets of potential solutions via gated spreading activation, and when cues in the environment will preactivate likely solutions via primed spreading activation. Some examples of these tasks include:

- **information retrieval:**

Applications such as web search or document retrieval are particularly appropriate for the context-sensitive asynchronous memory approach. The process of analyzing a question will uncover relationships between elements of the question and elements of likely targets, such as an authorial relationship between a name and a document or a summary relationship between a word and a document. In conditions in which questions are asked within an environment, or a set of questions are asked in the context of a single information need, elements of the environment can preactivate potential answers to a question before the question is asked. Chapter 8 discusses the Nicole-IRIA system which applies context-sensitive asynchronous memory to information retrieval.

- **planning:**

In many planning domains, easily observable features of the problem figure prominently in the solution. As mentioned earlier, in domains such as logistics, travel planning, and meal planning features observed in the environment influence not only applicable operators but classes of solutions and thus correlate with classes of solutions. Chapter 7 discusses the Nicole-MPA system which applies the context-sensitive asynchronous memory approach to planning.

- **language understanding:**

Language understanding is a task that can be performed on many levels. Without entering into the debate of what “really counts” as “language understanding,” let us simply note that for simple tasks like limited-domain dialogue systems “language understanding” can be done with simple pattern matching (e.g., Bobrow 1968, Raphael 1968, Weizenbaum 1966) or relatively constrained parsing (Winograd 1972) whereas for complex tasks like real-world story understanding “language understanding” can be a rich process drawing on many different domains of knowledge (Moorman 1997). The further up the hierarchy of complexity one goes, the more opportunities there are in real texts to find out new information, revise old models, and change the existing understanding of the problem. Language understanding is therefore an ideal task for the context-sensitive asynchronous memory approach. In fact, the ISAAC system cited above (Moorman 1997) used the Nicole system as its memory retrieval module; however, because the focus of ISAAC was creative reading rather than memory

retrieval the designer made an explicit choice not to exploit the full potential of the context-sensitive asynchronous memory approach (Moorman 1998, personal communication).

- **diagnosis:**

One of the features of diagnosis is that the immediately available features in the environment — a patient's gross symptoms or the immediately observable features of a bug — may not directly indicate a solution. A severe frontal headache can indicate a stroke, a tumor, the onset of an acute migraine attack, or an overzealous consumption of an ice cream cone. However, another important feature of diagnosis is that there are information gathering actions which can be performed that yield features which are more indicative of actual conditions. This process of iterative evidence gathering is a strong indicator for the context-sensitive asynchronous memory approach.

Other domains which are potentially applicable are design, story understanding, and aiding systems such as those discussed in the introductory medical diagnostic aid discussed in Chapter 1. Domains which might be less applicable or not applicable include problems where solutions cannot be easily reused, such as constraint satisfaction problems, or where evidence in the environment does not correlate well with solutions, such as cryptography.

### 3.8.6. Some Limitations

The context-sensitive asynchronous memory approach has its limitations. Its memory retrieval algorithms can readily support a large knowledge base containing many different bodies of knowledge, even overlapping knowledge; in the presence of appropriate contextual information the context directed spreading activation algorithm can effectively focus memory search on the appropriate subset of the knowledge base. A similar situation exists in large knowledge bases for single tasks in which adequate structure exists to distinguish items from each other. However, single-task knowledge bases containing very large sets of poorly discriminated items (where “very large” will be defined shortly) pose a more difficult challenge.

**The Massive Fanout Problem.** Context-directed spreading activation guides search of memory by spreading activation from features observed in the environment or in the reasoning context to potential answers to memory retrieval requests. It limits retrieval effort through fanout and thresholding; as quanta of perturbation propagate from concept to concept they decrease in strength until they fall below threshold, when they stop. As the number of items connected to a feature (along a single relation) grows, the fanout from observable features also increases; when that number becomes sufficiently large, the cost control policy inherent in the CDSA algorithm will threshold out activation along those links. Essentially, the CDSA algorithm treats high-fanout nodes as “stopwords” (see discussion in Sparck Jones & Willet 1997) which do not usefully discriminate items in the knowledge base along that particular relation.

However, in a single-task knowledge base containing many poorly discriminated items — essentially, redundant answers to a question which share many of the same features — many potentially observable features in the environment will be connected to many potential answers. As the number of observable features which have sufficiently high fanout increase, the CDSA algorithm becomes less and less effective at guiding search to relevant items in the knowledge base. In the worst case scenario, when the number of poorly discriminated items becomes “very large” then all of the potentially observable features have high fanouts and the CDSA algorithm will be unable to find *any* answers in the knowledge base — even though there may be literally millions of answers to the question!

**A Dynamic Memory Solution.** The behavior of CDSA under this extreme condition is precisely what we would expect given CDSA’s cognitive model. CDSA is based on a spreading activation approach to semantic memory — essentially, based on models of the human memory retrieval process. Humans do not have good memories for tens or hundreds of thousands of poorly discriminated items; a human may remember specific details about each of a hundred or even a thousand novels read over a lifetime, but are less likely to recall specific details about the tens of thousands of morning newscasts that they have heard over their lifetimes.

Yet humans do remember specific details about distinguished headlines — “Iraq Invades Kuwait,” “Dow Falls 500 Points,” “Evidence of Life on Mars” — and generic information about classes of news stories and news stories in general — “President Signs



Bill into Law,” “Stock Market Rises,” “Home Team Wins.” It has been argued (Schank 1982) that this human capability is based on dynamic memory: the active and ongoing organization of memory to learn general concepts of classes of events and remember distinctive details about specific events.

One way to overcome CDSA’s limits with large, poorly discriminated data sets would be to implement an experience store based on a dynamic memory approach such as used in the CYRUS system (Kolodner 1983). CYRUS used a technique called redundant discrimination nets (Kolodner 1983, 1993) to organize a knowledge base of events, simultaneously learning higher-level concepts about classes of events in the knowledge base and providing a rich structure to enable those events to be retrieved and reconstructed based on partial information. This approach is complementary to spreading activation approaches, providing a structure over which spreading activation can operate (Kolodner 1983 pp. 229-230).

A dynamic memory version of an experience store would automatically structure and organize knowledge as new items were added, building *memory organization packets* or MOPs which would simultaneously summarize many similar nodes and limit the effective fanout of any given feature node. This richer structure is indexed and labeled by features in the environment, enabling a CDSA-based memory search system to use these features to guide spreading activation to relevant items or relevant summarizing MOPs.

**Other Solutions.** A number of other technical solutions exist to this problem. One would be to alter the properties of the CDSA algorithm; alternatives include adding new kinds of context directed spreading activation, using the available information in new ways to focus activation along certain links for high fanout nodes, or to introduce randomness, stochastically allowing limited quanta of activation to “tunnel” past a fanout barrier. Another solution would be to add new layers to the search process, harvesting information from a larger database prior to applying the CDSA algorithm. A similar solution would be to rely more heavily on primed spreading activation, preactivating portions of the knowledge base to reduce search.

**Other Problems.** Because CDSA limits the amount of memory search performed, it has the potential to fail when the semantic distance between the source and target is too great. As the semantic distance of relevant targets increases, the strength of spreading perturbation quanta decreases, ultimately leading to threshold cutoff. While the CDSA approach is superior to traditional spreading activation at dealing with this problem when adequate contextual information exists (as discussed in more detail in Chapter 7) if the semantic distance is too great even CDSA will ultimately fail. To cope, an experience-based agent system may need an additional “strategic” search component which directs search through the search space, such as the “beacon” search used in the KDSA approach (Wolverton 1994).

### 3.9. Conclusion

This chapter presented context-sensitive asynchronous memory, a general solution to the problem of getting useful answers from large knowledge bases under resource constraints. Context-sensitive asynchronous memory is based on the general idea of using feedback from task performance to guide an ongoing memory search process.

To make this idea work, a context-sensitive asynchronous memory is composed of several components that work together, including an asynchronous memory retrieval system that represents retrieval requests as explicit data structures, a context-sensitive memory search system based on a context-directed spreading activation algorithm that reduces the effective size and branching factor of the knowledge base, a cost control policy that limits the amount of effort expended on search and matching, and an experience store whose representational structure simultaneously supports the requirements of asynchronous search and context-sensitive retrieval while enabling the storage of knowledge for a variety of tasks.

A context-sensitive asynchronous memory is most effectively used as part of a process in which memory retrievals are asked and search proceeds in parallel with the performance of a task, and in which the task actively inspects the results memory returns and actively generates new cues to guide further memory search. This approach attempts to exploit regularities in the environment, using feedback from the task to activate related

and potentially relevant items and to bring them to the attention of reasoning and memory.

In the next chapter, we will discuss in more detail the requirements a context-sensitive asynchronous memory places on a reasoner and how we can design reasoning tasks to meet these requirements.

# CHAPTER IV.

## INTEGRATION MECHANISMS

---

*Coping with the impact of context-sensitive asynchronous memory on reasoning*

Context-sensitive asynchronous memory is different from traditional approaches to memory retrieval in a number of ways. When a context-sensitive asynchronous memory is asked a question it generally does not provide an immediate response; instead, answers may be arbitrarily delayed as memory search progresses. Furthermore, answers are provided through an alert/response system rather than a call/return model, and for best results the memory system should be provided context while it searches and feedback about the items that it has found so far. Context-sensitive asynchronous memory is most effectively used as part of a process in which reasoning both expects retrieval to be interleaved with task performance and in which reasoning is expected to actively participate by providing cues and inspecting results.

For these reasons, “traditional” reasoning systems may not be able to reap the full benefits of the context-sensitive asynchronous memory approach without modification. “Traditional” reasoning tasks, designed to function around a simple call-return model, cannot use context-sensitive asynchronous memory directly — for example, posting a request returns a retrieval request handle and not an immediate answer. Therefore, traditional reasoning tasks will need some kind of support framework that simulates the

functional profile they require from a memory retrieval system. Similarly, reasoning tasks that can work with an alert/response system may not be designed in a way that effectively provides context-sensitive asynchronous memory with the feedback it needs; these systems will also need to be augmented. In short, reasoning tasks must satisfy certain constraints to work with a context-sensitive asynchronous memory effectively: the ability to consider knowledge when it becomes available, decide whether to use that knowledge, and to integrate the knowledge into the current processing state.

It is important to note at this point that these guidelines for constructing reasoners are technically a part of the overall experience-based agent approach, just as the context-sensitive asynchronous memory approach itself is a part of the experience-based agent approach. However, just as context-sensitive asynchronous memory can be coherently pulled out of the experience-based agent approach and applied to other domains (and, as a further parallel, as CDSA can be pulled out of the context-sensitive asynchronous memory approach and applied to other spreading activation systems), here we pull these guidelines for constructing integration mechanisms out into a separate unit to examine them on their own footing.

This chapter details the impact of context-sensitive asynchronous memory on reasoning processes. It begins by discussing the constraints context-sensitive asynchronous memory imposes on reasoning tasks and how these constraints are motivated just as much by the environments in which context-sensitive asynchronous memory is most suited as they are by the context-sensitive asynchronous memory

approach itself. The chapter continues by unpacking the demands that the context-sensitive asynchronous memory approach places upon reasoning tasks and by providing guidelines and examples of how these demands can be satisfied. Finally, the chapter concludes by putting the pieces together, outlining how a reasoning task that satisfies the constraints can exploit the benefits of the context-sensitive asynchronous memory approach and pointing the way for putting context-sensitive asynchronous memory and an appropriate reasoning system together into a complete experience-based agent architecture.

## **4.1. Demands of Context-Sensitive Asynchronous Memory**

In order for an agent performing a task to take full advantage of a context-sensitive asynchronous memory, it must have certain characteristics.

First, the task itself must have a structure that is useful for guiding retrieval based on contextual information. As discussed in the previous chapter, the knowledge that an agent accumulates during the performance of a task must be related or associated in some consistent way with features of the task or environmental context in which that knowledge can be applied. Consider a planning task in which an agent solves problems in some environment. Useful contextual structure for this task means that the features observable in the environment surrounding a problem — initial conditions, goal events, contextual information — correlate with classes of plans that are applicable or related to solving the problem. For example, in the domain of logistics planning, the presence of a

dolly, a pickup truck or a freight train correlates well with kinds of plans that can be applied. The alternative is that there are no observable features which correlate well with applicable plans; an example might be a cryptographic domain in which the encrypted text has been deliberately designed to obscure cues to the mathematical operators that can be applied to it. Tasks without useful contextual structure will find it difficult to fully exploit context-sensitive asynchronous memory because they will not provide the memory with the cues it needs to guide its search.

Second, the agent's reasoning methods must interoperate with a context-sensitive asynchronous memory. In order for an agent to benefit from the context-sensitive asynchronous memory approach, its reasoning methods must be configured in such a way that they can work with the distinctive properties of a context-sensitive asynchronous memory system. This includes things as simple as generating requests for information that the memory can then process and things as complex as evaluating the usefulness of an asynchronously returned retrievals with respect to the current reasoning context. Returning to the planning task, in order to take advantage of a context-sensitive asynchronous memory the reasoner must generate explicit declarative requests for information and must be able to examine retrieved plans with respect to the current reasoning context. For example, when faced with the problem of shipping a load of bananas, the planner needs to generate an explicit request for plans and give that to the memory; then, if the memory returns a plan for shipping lettuce via refrigerated trailers, the planner must be able to evaluate the plan with respect to the reasoning it has already



done and adapt the plan accordingly (e.g., by increasing the temperature of the trailers so the bananas are not damaged by the cold). In other words, the input and output of reasoning must be formatted in such a way that it provides what context-sensitive asynchronous memory needs.

Finally, the agent's control methods must be able to take advantage of a context-sensitive asynchronous memory. When memory can return information spontaneously and asynchronously, the control of reasoning must be able to change to accept and incorporate information as it is generated. Rather than attempting to accumulate all relevant knowledge prior to beginning reasoning, planning what knowledge to use, and then beginning the reasoning process, an agent must instead consider knowledge when it becomes available, decide what knowledge to use, and decide how to integrate it into its current processing — all in the context of the dynamic action, reasoning and retrieval state of the entire agent functioning in a real-world domain. Returning to our example, the agent planning to ship our bananas could begin to calculate the number of pallets and the estimated loading time and at the same time consult the dispatcher to see what shipment methods are available; hopefully, this conjunction of cues would lead to the retrieval of better plans than any one cue would alone. In order for an agent to fully exploit a context-sensitive asynchronous memory, control of the agent as a whole must be configured to deliberately exploit the asynchronous, context-sensitive nature of the memory retrieval process. The next chapter discusses in more detail how this can be accomplished as part of an overall experience-based agent architecture.

## 4.2. Demands of Dynamic, Unpredictable Worlds

The requirements context-sensitive asynchronous memory places upon reasoners are just as much a consequence of the environments for which context-sensitive asynchronous memories are designed as it is of context-sensitive asynchronous memory itself. Agents built around context-sensitive asynchronous memories — experience-based agents — are designed to operate in dynamic, unpredictable worlds with limited information; these environments place their own demands on the reasoning process.

Retrieval itself is motivated by a desire to gain the benefits of experience: an agent operating in a complex but regular real-world environment faced with difficult problems and tight bounds on the time and computation available to solve those problems can reuse its past knowledge to help it survive. The benefits of reuse have been demonstrated in a variety of systems, including CASEY (Koton 1989), ROUTER (Goel et al. 1994) and SPA (Hanks & Weld 1992, 1995) although some researchers question the efficiency with which case-based systems can learn (e.g., Griffiths & Bridge 1995). This is the challenge of reuse: taking advantage of past knowledge efficiently and effectively.

While the past provides benefits, retrieving experience comes with a time cost. In a dynamic world, we cannot guarantee that the cues and specifications that reasoning provides to memory will be sufficient to allow retrieval of the best experiences before some initial action or reasoning is demanded. To meet the pressing need for action an agent may not be able to wait for retrieval to complete, and in a traditional system would

need to interrupt the retrieval process. An agent built around a context-sensitive asynchronous memory avoids this problem by allowing the memory process to return a “best guess” initially while continuing to search memory in parallel with any reasoning, acting, or sensing operations being performed by the agent.

Even if time is not an issue, it may not be possible to gather all the knowledge needed to solve a problem at the beginning of problem solving. For example, imagine the problem facing a woodworking hobbyist building a new bookshelf. It is not clear in advance whether or not the agent needs to buy new sandpaper, and hence unclear whether the agent should recall past experiences of buying sandpaper at a hardware store. This uncertainty arises out of several concerns: the uncertainty of the world state (how much sandpaper does the agent have?), uncertainty in the effectiveness of agent actions (how much wood will a piece of sandpaper sand?) and the potential of exogenous events that can invalidate parts of the plan (if a friend drops and scars a piece of the furniture, will the agent have enough sandpaper to remove the scar, or will he need to buy more?). But it can also arise out of *the plan itself*: until the agent has decided on a design for the piece and how much wood will be involved, it is unclear precisely how much sandpaper is needed (or even if sandpaper will be needed at all), and hence unclear whether or not a plan should be retrieved. If some amount of sandpaper is on hand, the goal of acquiring sandpaper may not even arise until late in the planning process, when it has become clear that the amount on hand is insufficient.

Beyond these problems lies another: in a complex environment no single past experience may completely address the needs of the current situation. While the world is partially regular, it is not perfectly regular; often, the information necessary to solve a current problem is spread out over several past experiences. Examples of systems and proposals which have exploited multiple experiences to solve problems include MEDIATOR (Kolodner & Simpson 1988), Barletta & Mark (1988), SBR (Turner 1989), CELIA (Redmond 1990, Redmond 1992), JULIA (Hinrichs 1992), ROUTER (Goel et al. 1994) and NIMRANI (Watson & Perera 1997) to name just a few.

If the reasoner had all the past experiences presented to it, it could identify the relevant portions and attempt to combine them. However, even presuming an agent had the capability to correctly specify all that knowledge it needed before tackling a problem, and presuming that the costs of collecting all potentially relevant knowledge did not outweigh the benefits,<sup>11</sup> an agent may not have the luxury to wait around and collate all the past knowledge it needs before it begins reasoning. The reason is that even if an agent is not using an asynchronous memory retrieval system, in a real environment important information may nonetheless be delivered to the agent asynchronously by an exogenous event. Consider again the example of a NASA mission controller trying to solve a problem aboard a spacecraft. A key piece of information for resolving a problem

---

<sup>11</sup> For an discussion of the potential drawbacks of storing or retrieving too much information, see Goel et al. 1994, which varied the storage strategies of an case-based route planning system to illustrate the conditions under which learning too much information led to a performance penalty.

may be delivered “asynchronously” from the environment — a crucial reading is taken on board the spacecraft, a schematic is found after an exhaustive search, or the vacationing designer of a part is finally tracked down and calls in with critical advice.

Regardless of the source of a “spontaneously retrieved” piece of knowledge or past experience, when that knowledge does arrive the reasoner must not only dynamically compute what parts of that experience are relevant to processing, but also use that knowledge to change what it is doing. Dynamically combining the best portions of past experiences or knowledge items into the current reasoning state as those experiences are retrieved is key to exploiting spontaneous retrieval. This capability is called *integrative reasoning*.

In sum, reasoning tasks in dynamic unpredictable environments require the ability to combine multiple experiences, clip out their irrelevant subparts, and splice them together into a complete solution for the problem at hand. This integration of experience both enables and can be driven by the spontaneous retrieval of relevant experiences by the context-sensitive asynchronous memory system. In the next two sections, we will outline a general strategy for efficiently processing spontaneous retrievals, then unpack these requirements on reasoning more explicitly, setting the stage for our discussion of how these requirements can be satisfied.

### **4.3. Efficient Handling of Spontaneous Retrievals**

Incorporating an arbitrary piece of knowledge into an arbitrary reasoning state is a potentially computationally expensive process. Arbitrary amounts of reasoning may need to be performed to determine the relevance of the piece of knowledge to the current reasoning state and to then perform the reasoning operations which would integrate it into that state. For example, large amounts of reasoning may be required to incorporate a new piece of evidence into a scientific theory; major parts of the theory may need to be revised or discarded or unusual features of the evidence may need thorough analysis to be explained away.

In some circumstances, such as scientific discovery, it may be acceptable to perform this kind of intensive relevance determination and integration processing on a given piece of knowledge. However, it is not practical to apply deep reflection to each passing fancy that flits through our head; nor is it practical for a computer system based on a context-sensitive asynchronous memory to exhaustively examine each candidate retrieval for relevance to the current reasoning state.

Instead, the context-sensitive asynchronous memory approach to handling asynchronously returned spontaneous retrievals limits the amount of effort expended in considering retrieval by making quick decisions as early as possible in the process. This retrieval handling process is a cooperative four-stage evaluation and winnowing effort conducted by memory retrieval and reasoning processes in concert, evaluating and

eliminating irrelevant experiences as soon as possible to reduce the effort expended on expensive processes such as matching or relevance determination. The stages of this process include fast memory search, memory matching, retrieval evaluation, and relevance determination:

- **fast memory search:**

Fast, knowledge-poor selection of items in the knowledge base that are possible matches to the retrieval specifications. Requires no matching or comparison, but instead lookup of knowledge based on indices distilled from retrieval specifications and contextual cues.

- **memory matching:**

Slower selection of items out of the fast search set which satisfy the retrieval request. Requires inexpensive matching of individual knowledge items against a fixed set of retrieval specifications.

- **retrieval evaluation:**

Evaluating successfully matched items in the context of the current reasoning state. May involve more detailed matching or arbitrary amounts of reasoning and analysis to determine whether the retrieval is in fact suitable for use.

- **relevance determination:**

Given a knowledge item which is relevant, relevance determination examines its detailed structure and extracts the subportions which are relevant to the current reasoning state.

Fast evaluation and expensive relevance determination in reasoning are decomposed for the same reason fast search and expensive matching processes are decomposed in memory. It is certainly possible for a reasoner to inspect every item that the memory retrieves at every reasoning step for potential relevance, just as a memory system could match every item in a knowledge base against the retrieval specifications. However, just like specification matching, relevance determination (applying the test of a rule, or finding the component of an experience that teaches a relevant lesson) is potentially a computationally expensive process, depending on the particulars of the reasoning task; therefore, determining relevance for every retrieved candidate would also be computationally expensive.

For this reason, the context-sensitive asynchronous memory approach to handling asynchronous retrievals explicitly separates fast retrieval evaluation from relevance determination. Retrieval evaluation enables a reasoner to quickly examine the retrieval candidates returned by memory, winnowing out unsuitable candidates using fast, estimated metrics and task-specific retrieval criteria and thus eliminating the need to perform relevance analysis upon them.

#### **4.4. Constructing Reasoning Systems That Satisfy the Demands**

Context-sensitive asynchronous memories, dynamic environments, and efficient asynchronous information processing all place demands on reasoning tasks. In addition



to a functional profile that responds to the peculiar input/output style of context-sensitive asynchronous memory, reasoners need the ability to combine experiences for maximum benefit while minimizing the effort of processing asynchronously retrieved information.

Returning to our desiderata for memory systems listed in chapters 1 and 2, we can now see that certain desiderata upon memory — a memory system should exploit task/environmental information (Desideratum 6), should be potentially interleavable with reasoning (Desideratum 8), and should provide guidelines for reasoning integration (Desideratum 9) — are just as much constraints on the reasoning tasks that interface with memory as they are constraints upon memory itself.

Reasoning systems for context-sensitive asynchronous memory must work with an independent memory task, provide feedback to that task, and efficiently accept and profitably integrate retrievals as they are found. Unpacking these requirements, to take advantage of a context-sensitive asynchronous memory a reasoning task needs to:

- **Work with asynchronous memory:**
  - represent all potentially useful knowledge in the experience store
  - pass requests for information to the asynchronous memory
  - work in parallel or interleaved with the memory process
- **Provide feedback to context-sensitive memory:**
  - provide feedback about the task and environment to the memory

- optionally, update the specifications of requests for information
- **Accept spontaneous retrievals:**
  - **interruptible:** provide handlers which accept asynchronous memory retrievals
  - **deliberative:** efficiently evaluate the fitness of asynchronous retrievals
  - **integrative:** incorporate beneficial retrievals into current processing

Some of these capabilities, such as operating in parallel with other tasks and providing feedback to the memory, do not need to be implemented afresh for every reasoning task; in the next chapter we will consider what kind of architectural support can be provided for interfacing reasoning tasks with a context-sensitive asynchronous memory. However, while architectural support is useful, it is not sufficient: to meet these desiderata modification must usually be made *within* a reasoning task as well.

Traditional reasoning algorithms have well-defined inputs, modify their inputs in some way, and emit the resulting data through well-defined outputs. While reasoning algorithms may iterate over well-defined steps in the process of modifying that data, those steps are “private” and do not necessarily translate into any coherent partial state which would be meaningful to the outside world. For example, a reasoning system’s intermediate states may not be logically correct or coherent prior to the application of truth maintenance routines; similarly, interrupting a neural network or classifier system

early during training may leave it with internal states that are incoherent state, not useful for performing any task.

Therefore, “in the middle of processing a task” has no well-defined meaning in the traditional input-processing-output model, and thus retrieving information “in the middle of processing a task” is simply useless if the task has no way to affect its outputs based on that information. A reasoner needs additional “entry points” to accept asynchronous retrievals, along with “integration mechanisms” which can incorporate that information into current processing. And since the point of asynchronous retrieval is to improve the system’s performance, the reasoner furthermore needs to evaluate the retrievals that it does receive to make sure that incorporating that information will actually provide some benefit.

Each of these three general requirements places can be unpacked into more specific constraints on reasoning tasks. The next three sections expand upon these requirements and explain how reasoners must be modified to satisfy them.

#### **4.4.1. Working with an Asynchronous Memory**

Before a reasoner can integrate an asynchronously retrieved piece of information, it must have initiated a request that that information responds to; in order for a reasoner’s requests to be satisfied some information relevant to the reasoner must be stored in memory. In other words, for a reasoner to exploit an asynchronous memory it must first use it.

- **reasoners must represent all potentially useful knowledge in the experience store:**

A memory is only as good as its content; if no reasoning task ever stores information in the experience store then the memory system will never retrieve anything useful. Each piece of knowledge that a reasoning task would like to retrieve (or would like to make available for other tasks to retrieve) must be stored or referenced by the experience store. If those pieces of knowledge are difficult to represent in a semantic network, they can be stored as “opaque” pieces of knowledge, wrapped by the grounding system of the experience store and “tagged” with labels useful for the retrieval of that knowledge.

- **reasoners must pass requests for information to the asynchronous memory:**

Next, when a reasoning task has a need for information it must communicate that to the asynchronous memory, which will encapsulate that request in a knowledge goal or retrieval request. Ideally, a reasoning task should maintain its own list of the outstanding requests it has and should have a strategy for managing retrieval. This strategy would dictate when to create a request, when to refresh or update it, when to accept or reject a candidate, and when to accept or reject the request overall.

- **reasoners must work in parallel or interleaved with the memory process:**

Once a request for information has been posted, the reasoning task must not “block” — that is, must not pause waiting for a result. While this synchronous

reasoning style is a traditional mode of interaction between reasoning modules and data sources, with its use one of the major benefits of asynchronous retrieval is lost. To fully take advantage of asynchronous retrieval, a reasoner must not only continue working on the problem, it must be structured in such a way that the memory task can search at the same time. There are a number of ways of achieving this goal: the reasoning method and memory system could run in parallel, the reasoner could be a subtask of a controlling task responsible for interleaving its execution with the memory, or the reasoner itself could act as a controller the memory subtask, allocating the memory a certain amount of processing time. In Chapter 7, we will discuss ways to provide architectural support to make it easier for reasoners to meet this requirement.

- **provide the memory feedback about the quality of retrieved results:**

While providing feedback to the memory about the quality of its work is not required to exploit a context-sensitive asynchronous memory, this feedback is potentially useful in two ways. First, if initial retrievals for a request are poor, the memory system can attempt to improve its performance on subsequent retrievals. Second, the memory system can learn across retrievals over time — for example, by changing the default weights of retrieved items to ensure that likely candidates are returned and unlikely ones are ignored, or by learning new associations between items in the knowledge base.

#### 4.4.2. Providing Feedback to Context-sensitive Memory

Just as there are things that an agent requires to take full advantage of asynchronous memory, so there are things that an agent must provide in order to exploit context-sensitivity. First and foremost, the reasoner can provide feedback; second, the reasoner can provide updated specifications.

- **reasoners must provide feedback about the task and environment to the memory:**

Without feedback, a context-sensitive memory system cannot guide or improve the search of the experience store. As the reasoner generates additional information about the task, or encounters additional information about the environment, it should feed that information to the memory in the hope that this additional information will improve the quality, quantity or timeliness of retrieval. Feedback can be explicit or implicit; the reasoner may deliberately and explicitly feed information to the memory system, or the memory may monitor the reasoner's actions unobtrusively to guide its search. In Chapter 7, we will discuss ways to provide support for implicit and explicit feedback at the architectural level.

- **reasoners may update the specifications of requests for information:**

The reasoner may learn additional information about its request for information, as a product either of further reasoning about a task, of encountering additional information in the environment, or of evaluating the goodness of initial candidate

retrievals.<sup>12</sup> While informing the memory about a change in the requirements is not necessary to take advantage of context-sensitive retrieval, it can improve the system's performance — either by changing the context for memory search or by changing the specification of what should be retrieved. The result of this update might be to weed out candidates previously let in, or to let in what had been previously screened out, or to simply change the set of candidates considered; regardless, the goal of any updated specification is to improve memory retrieval.

#### 4.4.3. Responding to Spontaneous Retrievals

Presuming a reasoning method can interoperate with a running asynchronous memory and is providing useful feedback to its context-sensitive mechanisms, what does it then need to do when spontaneous retrievals actually arrive? At least three capabilities are key:

- **reasoners must be interruptible:**

Interruptibility is the capability change the control flow of a reasoning process in response to a signal from the memory system — essentially, the capability to

---

<sup>12</sup> For example, a variety of systems such as have demonstrated the need strategic control of memory processes which includes updating and refining how memory is queried; these include CYRUS (Kolodner 1983), MINSTREL (Turner 1992, 1994), KDSA (Wolverton 1994) and ISAAC (Moorman 1997) to name just a few. A variety of other systems have demonstrated the need to take new information into account when performing retrieval, including REMIND (Lange & Wharton 1994) and ALEC (Simina 1999) to name just a few.

react to spontaneous retrievals. Interruptibility is the first and perhaps most crucial constraint imposed upon reasoning tasks by context-sensitive asynchronous memory: if a reasoning task cannot respond when asynchronous memory retrieval occurs then asynchronous memory retrieval is worthless. When memory finds a new candidate asynchronously and that data is brought to the attention of reasoning, the control flow of reasoning must be able to change appropriately.

- **reasoners must be deliberative:**

Deliberation is the capability to react to spontaneous retrieval only when retrieval provides something worthy to react to. A reasoning process must evaluate the fitness of an asynchronous retrieval with respect to its relevance to the current reasoning state. Ideally the disruption of the reasoning process in response to a retrieval should be limited to a minimum unless we have some indication that the item which has been asynchronously retrieved will provide some benefit if incorporated into reasoning. This requires both evaluating the goodness of a result from memory search and directives for action based on the evaluation of candidate retrieval.

- **reasoners must be integrative:**

Integrative reasoning is the dynamic incorporation of new information as it arrives. A reasoning method needs mechanisms to identify the relevant portions of useful past experiences and to incorporate them into the current reasoning



state. Once an agent has reacted to the interruption of a spontaneous retrieval and has come to a deliberate decision that the retrieval is useful, it must then actually act on the retrieval by incorporating it into its current reasoning state.

These processes are the task-specific counterparts of the matching process within an asynchronous memory: just as the asynchronous memory performs a quick-and-dirty, domain-independent match of active candidates against specifications and then evaluates the quality of the match with respect to the ongoing retrieval state, so must the reasoning task apply its slower, domain-specific criteria to each spontaneous retrieval, evaluate its relevance with respect to the ongoing reasoning state, and, if and only if appropriate, activate any facilities it has to process the candidate further.

## **4.5. Integration Mechanisms**

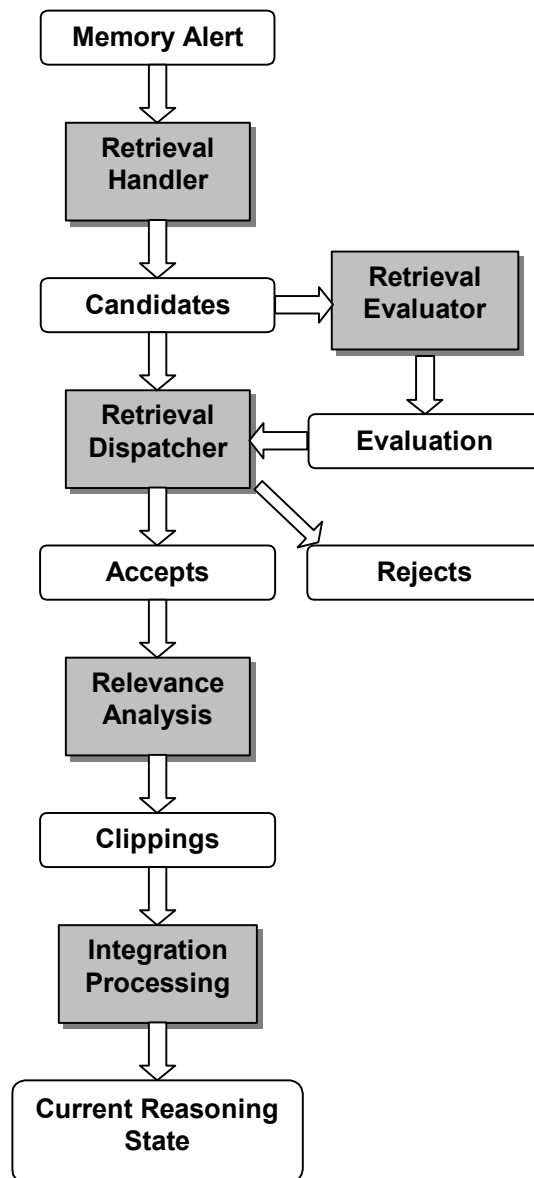
We are now in a position to place all of these desiderata, constraints and proposed solutions together into a solution. Clearly, achieving interruptibility, deliberation and integration could be nontrivial, depending on the reasoning method and the task it performs; a traditional reasoning method built around synchronous memory retrieval and an uninterrupted reasoning process might need to be drastically altered or augmented to satisfy these constraints. While the context-sensitive asynchronous memory approach cannot solve these problems for every reasoning process, it can provide an outline for how these problems can be solved in general.

A reasoning process can be made interruptible, deliberative and integrative through the use of an *integration mechanism*: a coordinated suite of subtasks within a reasoning process that accepts retrievals when memory finds them, determines the quality of the retrieved items, decides whether it is worthwhile to integrate them, and, if the retrieved items are worthwhile, extracts the relevant portion of a retrieved item and integrates it into current processing. Optionally, integration mechanisms may also be responsible for providing feedback to the memory system, either in the form of updated cues or additional specifications.

Specific subtasks within integration mechanisms, such as extracting the portion of a retrieved item relevant to the reasoner's current processing state, are necessarily task-specific. However, an integration mechanism is a general *design pattern* which applies across a variety of tasks (Figure 4.1). The components of this pattern include:

- **retrieval handlers:**

Retrieval handlers are components which respond when memory “posts an alert” that it has found an answer. Retrieval handlers are reasoning's component of the asynchronous memory alert system discussed earlier.



**Figure 4.1 Outline of an Integration Mechanism**

- **retrieval evaluator:**

Retrieval evaluators are responsible for evaluating the candidate retrieval according to task-specific metrics, quickly and inexpensively determining whether a candidate is worthy for further processing.

- **retrieval dispatcher:**

Retrieval dispatchers for executing the appropriate action based on task-specific criteria and the results of the evaluation; this may include immediately discarding a candidate, queuing a candidate for further consideration, or interrupting current reasoning to immediately act upon a candidate.

- **relevance analyzer:**

If the dispatcher determines that a candidate should be incorporated into the current reasoning state, the relevance analyzer then performs a more detailed analysis to determine what portion(s) of the candidate can be used in the current scenario.

- **integration processor:**

Finally, the integration processor performs the task of incorporating the prior experience or other memory retrieval into the current reasoning state, using the guidance of the relevance analyzer to determine what to incorporate.

For an individual task the distinctions between these modules may blur: for example, in some tasks retrieval evaluation and relevance analysis may be one and the same; in others relevance analysis and integration processing may be combined.

In the following sections, we will discuss the various components of an integration mechanism, explain how they contribute to an reasoning module's ability to take advantage of spontaneous retrievals, and provide examples of how these components are

instantiated in the sample tasks Nicole-MPA, a case-based planner that adapts multiple plans, and Nicole-IRIA, an information retrieval system that uses context to recommend useful information.

#### **4.5.1. Retrieval Handlers**

To be interruptible, a reasoning process must be structured in such a way that when it asks a question it does not block or pause waiting for a response. If it does, the reasoning process will recreate synchronous memory system within an asynchronous one. Therefore, the reasoning process must first continue in parallel or interleaved with memory processing, but this is not enough; the reasoning process must also respond when an answer is found. Retrieval handlers are the elements of the integration mechanism design pattern that make a response possible.

A retrieval handler is a component of an integration mechanism that responds when memory “posts an alert” that it has found an answer, fulfilling the reasoning task’s part of the asynchronous memory alert system discussed earlier. A retrieval handler inspects working memory for the alerts posted by the retrieval monitor and responds when one is found. This can be done by active polling of working memory or by a callback invoked when an alert is posted. Regardless of the specific mechanism, the handler is responsible for getting the candidate from working memory and feeding it into the next step in the integration pipeline, the retrieval evaluator.

Two different mechanisms for retrieval handlers can be found in Nicole-IRIA and Nicole-MPA. The retrieval handler used in the Nicole-IRIA web search and recommendation application is simple: each time Nicole-IRIA presents or reorganizes a set of results, its presentation algorithm explicitly polls working memory for memory retrieval alerts. If it finds an alert, it then inspects that alert for relevance and attempts integration, a process discussed in the following sections. Nicole-MPA's retrieval handler is more complex, exploiting the properties of its experience-based agent architecture to perform the equivalent of a callback. When Nicole-MPA initiates a retrieval request, it spawns a recurring subtask whose precondition is a memory retrieval request; that subtask then impasses until a memory retrieval request is found. When the task executes, it inspects the alert for relevance as discussed in the following section.

#### **4.5.2. The Retrieval Evaluator**

A retrieval evaluator's task is simple: gauge the quality of a retrieval. It needs an evaluation function that computes the quality of retrieval according to some metric; both the evaluation metric and evaluation function are specific to the reasoning task. For example, a reasoning task seeking rules to apply to knowledge state needs only a boolean metric: either a retrieval candidate is a rule, or it isn't. A case-based reasoning task, in contrast, seeks experiences, of which there are many types; this type of task needs a more discriminating metric which estimates whether the experience is relevant to the current problem. Retrieval evaluation plays a crucial role in the overall retrieval process,

enabling a system to rapidly consider and discard irrelevant retrievals without expending the full effort for relevance detection.

Both Nicole-MPA and Nicole-IRIA employ a similar set of heuristics for relevance determination. At the most basic level, there may be multiple reasoning tasks and multiple retrieval requests for each task; a reasoning task must first match a retrieval alert to one or more of the retrieval requests that it has posted. Then, the retrieval candidate proposed by the alert must be matched against the knowledge goal that spawned the request for knowledge. If the reasoning task has carefully specified the matching specifications it provided to memory retrieval, the memory system's matching process may have already winnowed out the most poorly matching retrievals; however, since the memory retrieval process is deliberately designed to be fast and approximate an alternative strategy is to accept the memory system's best guess and then to apply task-specific tests. In Nicole-MPA, this matching consists of checking the domain and initial conditions of a retrieved candidate plan against the current problem; in Nicole-IRIA this matching consists of checking the words in the retrieved candidate web page against the search criteria provided by the user.

#### **4.5.3. The Retrieval Dispatcher**

Once the reasoner has evaluated a retrieval, it must then decide how to act. The retrieval dispatcher is the component of an integration mechanism that acts on the results of the retrieval evaluation. Naïvely, the dispatcher should discard retrieved candidates

below a certain quality; unfortunately, for an asynchronous, approximate reasoner the problem is a little more complicated. There are three primary questions an integration mechanism must face:

- When should the system begin reasoning after querying memory?
- When should it interrupt or alter what it is doing to incorporate an answer?
- When should it stop looking for new answers (either in success, or failure)?

Furthermore, the reasoning task must give appropriate guidance to the asynchronous memory system: it should communicate acceptance or rejection of the retrieval candidates it is given, and, when the reasoning task is ready to stop looking, should communicate when to terminate a retrieval. These decisions are handled by the retrieval dispatcher, which uses the retrieval evaluator's judgements about the candidates caught by the retrieval handler to guide both the execution of the task and the continued search of the memory.

As an example, consider case-based planning. A case-based planner can apply complex relevance determination which could be used to determine the most on-point case or cases, but this process may be expensive. If a case-based planner's memory presents it with a plethora of potential cases, a fast winnowing stage can eliminate unsuitable candidates before relevance determination begins. Ideally, during winnowing the planner should select the best-matching set of cases as the basis for further reasoning. However, if memory retrieves only a single case, the planner has little choice: it must



either accept the case it has been given, or reject it as of being too irrelevant to be of any use.<sup>13</sup> If the memory is a context-sensitive asynchronous system, there is another option: wait and see if more information can be retrieved.

Ideally, a reasoner should begin reasoning immediately based on first principles, accept only the highest quality retrievals for integration, and then terminate the memory retrieval request when we have a sufficient amount of information or when the reasoning process successfully terminates, whichever comes first. A second option would be to wait for an approximate first case, begin adapting it and then update the adaptation process as better retrievals are found. Failing that, after a suitable interval, the reasoner might take a suboptimal case and try to adapt it. If none of these options are viable, the reasoner may try to reason using first principles, or may simply give up.

There is a tension between the amount, quality and timeliness of information retrieved. For each of these variables there are three thresholds: the minimum threshold on retrievals (to proceed if we have no other choice), a sufficient threshold (to proceed but keep looking) and a maximum threshold (to stop looking altogether). These three

---

<sup>13</sup> Actually two additional options exist: to refine the question, or to reformulate the problem. The memory retrieval request can be strategically refined (e.g., Wolverton 1994 or Kolodner 1983) in the hope that a better retrieval can be found, or the problem itself can be reformulated to use a different reasoning task (e.g., analogical reasoning or reasoning from first principles for which knowledge is available (Moorman 1997). Both of these approaches are beyond the scope of this research.

criteria applied to these three variables yield nine thresholds  $Q_{minimum}$ ,  $Q_{commence}$ ,  $Q_{success}$ ,  $N_{minimum}$ ,  $N_{commence}$ ,  $N_{success}$ ,  $T_{minimum}$ ,  $T_{commence}$ ,  $T_{failure}$ , (Figure 4.2).

These thresholds are by their nature specific to each reasoning task. A rule-based reasoning task may proceed as soon as it finds any rule. A case-based reasoner without a first principles component may not begin reasoning until some case is found.

Nicole-MPA was a platform for testing a variety of first-principles, adaptive and integrative reasoning systems, and was tested with a variety of empirically set threshold settings.

The thresholds on quality of retrieval  $Q_{minimum}$  were set to exclude any candidate which could not be incorporated into the adaptation process ( $Q_{minimum} = 0.5$ , a threshold which, by convention in Nicole, signals a matching value “good enough” for retrieval;  $Q_{commence}$  was also set to 0.5). The structure of Nicole-MPA’s knowledge base made it unlikely that a candidate would be found in less than two retrieval cycles, so  $T_{minimum}$  was set to two; it also made it unlikely that any new information would be found after thirty cycles, so  $T_{failure}$  was set to 30. The number of retrievals sufficient to begin reasoning depended on whether Nicole-MPA was deployed in adaptive, integrative or first-principles mode; adaptive retrieval sought at least one retrieval ( $N_{minimum} = 1$ ,  $N_{commence} = 1$ ), whereas integrative retrieval attempted to collect, but did not require, multiple retrievals before the reasoning process was allowed to proceed ( $N_{minimum} = 1$ ,  $N_{commence} = 2$ ).

Parameter	Description	Typical Value
$Q_{minimum}$	Minimum threshold on the quality of a retrieval (if time limits are exceeded).	0.5
$Q_{commence}$	Sufficiency threshold on quality to begin reasoning	0.5
$Q_{success}$	Maximum threshold on quality at which we can stop looking (e.g., a perfectly matching case with a valid solution)	$\infty$
$N_{minimum}$	Minimum amount or number of retrievals to begin reasoning (if time limits are exceeded and quality threshold not met).	1
$N_{commence}$	Sufficient number of retrievals to begin reasoning	2
$N_{success}$	Maximum number of retrievals to terminate retrieval process	$\infty$
$T_{minimum}$	Minimum time to wait before beginning to reason if we have sufficient information (this may be zero)	2
$T_{commence}$	Sufficient time to wait if we have information above the minimum thresholds on quantity and quality	2
$T_{failure}$	Maximum time to wait if we have no information above minimum thresholds on quantity and quality	30

**Figure 4.2 Parameters of an Integration Mechanism**

No upper limit was set on the number of cases or quality of cases that could be retrieved ( $Q_{success} = \infty$ ,  $N_{success} = \infty$ ); Nicole-MPA continued to seek cases throughout its problem solving cycle.

The problem is much simpler in Nicole-IRIA, where fast response to user queries is key. Quality candidates are integrated into the result set as soon as they are found; if the memory system finds no results it informs the user immediately and gives the user the opportunity to try a new search. This is equivalent to settings of  $Q_{\text{minimum}} = 0.5$ ,  $Q_{\text{commence}} = 0.5$ ,  $Q_{\text{success}} = \infty$ ,  $N_{\text{minimum}} = 0$ ,  $N_{\text{commence}} = 0$ ,  $N_{\text{success}} = \infty$ ,  $T_{\text{minimum}} = 0$ ,  $T_{\text{commence}} = 5$ ,  $T_{\text{failure}} = \infty$ .

#### 4.5.4. Relevance Determination

After a reasoning process has noticed an spontaneous retrieval and decided that it is interesting enough to consider incorporating into current processing, the next stage is to determine what part, if any, of the retrieved experience is relevant to the current situation.

In some domains, each retrieved piece of knowledge is effectively an atomic unit; in these domains identifying relevance degenerates to filtering items against search criteria. While the evaluation of a retrieved piece of information with respect to the retrieval criteria can be quite complex (especially if the criteria incorporate multiple or fuzzy descriptions of matching items and/or feedback from the user) once the criteria have been established the entire item automatically becomes relevant. An example of such a domain is web search; in Nicole-IRIA once a web page has been found to meet the retrieval criteria it can simply be added to the retrieval set.

For other domains, the problem is not so simple. One example is document clipping and summarization: where a traditional web search application is expected to return

references to complete web pages, a summarization system might perform in a more complex analysis to attempt to identify the paragraphs or subsections which are relevant to the search criterion, rather than returning the whole document. More generally, relevance determination becomes more complex for any reasoner which remembers and manipulates complex objects which can be divided into many different components, each of which may be may be relevant to the current reasoning state. Examples include solution plans, object descriptions, arguments, designs, texts or stories, and so forth.

With the exception of knowledge that encodes its own conditions of applicability, such as rules, or knowledge which is effectively atomic, such as a fixed-size feature vector for an instance-based learning system, determining which part of a piece of past knowledge that is to be reused is relevant to a particular reasoning scenario cannot be done in when that knowledge is stored; it must be done when the knowledge is applied to the reasoning state in question. But when knowledge is asynchronously retrieved, the reasoning state is a moving target, complicating the problem of relevance determination.

As we mentioned earlier, a reasoning system may encounter multiple, overlapping problems and tasks as well as single difficult tasks which have many subcomponents. Furthermore, the reasoner may not have the luxury to wait for memory to return an exactly-on-point solution (presuming one exists). In any of these circumstances, a past experience similar to the current situation may contain only pieces relevant to the current problem, pieces which need to be identified and decomposed before they can be applied to the current situation. Under some circumstances these components can be extracted at

storage time (Redmond 1990, Veloso 1995) but the costs of this practice can come with a penalty when too many components are extracted in advance (for an example, see the discussion in Goel et al. 1994).

Furthermore, since the reasoner may have modified or refined the problem between the time that a query was posted and the time that memory found an answer, the initial assumptions about the specific type of and content of knowledge needed may have been invalidated. Relevance determination must therefore take into account not only the degree of fit of a retrieved item to the specifications of the query but also its degree of fit to the needs of the current reasoning state.

The capability of determining the relevant parts of a retrieved item with respect to the current reasoning state, rather than with respect to the state under which the retrieval request was made is called “dynamic clipping”. The major steps involved in dynamic clipping include:

- **Formulate Knowledge Goals:**

Reasoning tasks must be able to specify the knowledge needs of the current reasoning state (Ram & Hunter 1992). Presumably the reasoner had some way of computing its initial information needs when it first made a retrieval request; however, many reasoning tasks generate an initial request for information from the problem statement and not from a reasoning state; therefore, computing

information needs dynamically from a reasoning state may require additional machinery.

- **Identify Relevant Components:**

Reasoning tasks must be able to determine what parts of a retrieved item are relevant to those knowledge needs. This process may be more complex than comparing whole plans to unsolved problems because the reasoner may be in an intermediate reasoning state with additional complexity not found at the level of whole problems and whole solutions.

- **Extract Relevant Components:**

Reasoning tasks must be able to extract the relevant portions of a retrieved item in a form which will be useful for integration into the current reasoning state.

These operations should be low-cost: ideally, determining knowledge needs, determining relevance, and extracting relevant elements should all be linear or polynomial-time operations. The result of these processes should be a prepared piece of knowledge which can be quickly and efficiently reused through incorporation on into the current reasoning context.

An example of complex relevance determination for dynamic clipping occurs in Nicole-MPA. A retrieved case is unlikely to be an exact match for the current planning state; instead, only a subset of the goals and initial conditions of the plan are likely to be a match for the current set of open conditions and goals. As a result, the plan may

contain structure which is irrelevant to the current reasoning state; for example, a retrieved plan for a travel planning problem may contain steps related to actions such as choosing a destination, booking the travel, finding a hotel, and packing; however, one or more of these actions may not be relevant to the current reasoning state. To deal with this problem, Nicole-MPA “clips” a retrieved case to fit the current partial plan using an extension of Hanks & Weld’s SPA fitting algorithm (Hanks & Weld 1995).

This algorithm begins by formulating an intermediate goal statement which represents the needs of the current reasoning state in the form of a knowledge goal. The algorithm then attempts to find a maximal fit between the intermediate goal state and a retrieved plan. As part of this process, the algorithm identifies and removes structure from that plan not relevant to the current reasoning state. The result is a “clipping” or fitted partial plan ready to be merged into the current reasoning state. The most computationally expensive part of this process is determining the maximal match, which is exponential in time in the number of goals of the plan; clipping out the irrelevant parts is linear in the size of the plan. For more details on Nicole-MPA’s dynamic clipping algorithms, see Chapter 10.

#### **4.5.5. Integration Processor**

Once a relevant portion of past experience has been found, it must then be exploited. For domains in which answers consist of sets of atomic components, incorporation into current processing is straightforward: add the unit to the current set of found items. This



is the strategy used in Nicole-IRIA: as new web pages are found, they are added to the list of retrieved results.

For domains with more complex solutions, integrative reasoning is not so simple. Given both a current reasoning state and a segment of an experience which is relevant to that reasoning state, the task is to modify the current reasoning state, by merging the experience or otherwise altering the reasoning state to incorporate the information embedded within the experience.

Like the process of relevance determination, the process of merging should be low-cost, performed quickly and efficiently enough so that the cost of merging is outweighed by the benefits is provided by the added knowledge. If the cost of extracting a relevant piece of an experience and merging it is greater than the benefit it would provide if incorporated into the current reasoning state, integration is not worthwhile. This suggests but does not necessarily mandate the use of polynomial-time algorithms — reasoning processes themselves are often exponential-time and the reuse of past knowledge can provide vast speedups — but it does mandate that whatever algorithms are used should consume far less resources than the expected gains.

One example of an integration processor is Nicole-MPA's multi-plan adaptation algorithm. Nicole-MPA's plan splicing algorithm takes the current planning state (a partially completed plan) a prepared clipping (a pre-existing case with irrelevant parts spliced out) and uses the clipping to guide the addition of new steps to the plan that

mirror the structure of the old plan. This process is linear in the number of steps in the plan, and together with the dynamic clipping algorithm has been shown to merge plans with less cost than the benefits provided by the merged plan. For more details on Nicole-MPA's dynamic clipping algorithms, see Chapter 10.

## **4.6. Putting the Components Together**

An intelligent agent which combined the ideal version of all of these components would look and behave differently from many existing reasoning systems.

All of the reasoning modules in such an agent would use the same knowledge representation, long-term store, and working memory. As it encountered problems in its environment activation would spread from the working memory into the long term store to guide retrieval, enabling the memory system to estimate what kind of task and problem the agent was encountering and to return reasonable results even given very vague questions. If the memory was not able to immediately give reasoning processes within the agent the information they needed, reasoning would continue to work, generating more cues to guide and inform memory's search. If memory did find additional information, reasoning modules would quickly evaluate it and either incorporate it into the reasoning state or reject it, providing feedback to memory to further improve its retrieval performance. Systems based on this model would be able to freely exploit information from the environment or reasoning and be able to deal with both tight resource constraints and abundant resources.

This approach to constructing intelligent agents, in which the agent architecture is built to support and exploit the properties of context-sensitive asynchronous memory, is experience-based agency. The next chapter discusses the experience-based agent architecture in more detail, presenting the architecture and identifying where it is possible to provide general architectural support for the context-sensitive asynchronous memory approach.

# CHAPTER V.

## EXPERIENCE-BASED AGENCY

---

*Architectural support for context-sensitive asynchronous memory*

Where context-sensitive asynchronous memory tells how to get information from memory, and integration mechanisms explain how to incorporate it into reasoning, experience based agency explains how to put context-sensitive asynchronous memory and integration mechanisms together into a complete package — a package that uses context to get information for use in context, with the goal of building up information, or experience, that can improve how information can be reused in the future.

Given a context-sensitive asynchronous memory system, we could construct a completely new agent around it from scratch each time we were faced with a new task that we needed to apply the context-sensitive asynchronous memory approach to. Each time we did so, we would need to solve problems such as controlling independent memory and reasoning processes and integrating spontaneous retrievals into the current reasoning state; as a result many of the components of these agents would share the same features.

Some of the constraints context-sensitive asynchronous memory places on reasoners require task-specific solutions. For example, generating specifications for retrieval, evaluating the goodness of a particular candidate retrieval with respect to the current

reasoning state, or actually incorporating a candidate into the current reasoning state are all tasks which require specific knowledge of the features of the task and the implementation of the reasoning mechanism.

Other constraints lend themselves to more general solutions; in particular, working in parallel with an independent memory process and providing feedback to that process are tasks which require little knowledge of the specifics of the reasoning task and which have the potential to be solved in a similar way across a variety of reasoning tasks. Ultimately, control, working memory and some reasoning components for each new context-sensitive asynchronous memory agent would be similar, if not identical.

An alternative to constructing new agents from scratch is to design an architecture which encompasses the components that would be shared across various agents based on context-sensitive asynchronous memories, and then to use that architecture as a toolkit for constructing systems based on the context-sensitive asynchronous memory approach.

Experience-based agency is precisely such an architecture. The experience-based agent architecture specifies mechanisms for task control, task communication and reasoning structure which support and exploit the context-sensitive asynchronous memory approach.

## 5.1. Principles of Experience-Based Agency

An experience-based agent is an intelligent system, or agent, built around an autonomous memory process which continually and incrementally searches for knowledge in a context-sensitive fashion, guided by the agent's reasoning and environmental context. Retrieval in an experience-based agent can be anytime, responding on demand to reasoner requests, or proactive, alerting reasoners autonomously when better answers are found. To cope with these autonomous retrievals, the reasoning modules of an experience based agent must be able to accept retrievals as they arrive, evaluate them for fitness, and integrate relevant parts into the current reasoning state. To enable reasoning and memory to interoperate, and to enable context-sensitive search, reasoning modules and memory retrieval are embedded in an overall architectural framework which provides comprehensive knowledge representation, storage, communications and control facilities.

The experience-based agent approach supports and extends the context-sensitive asynchronous memory approach, augmenting context-sensitive asynchronous memory with additional principles which enable the benefits of context-sensitive asynchronous memory to be realized in an application. Reviewing our list from Chapter 1, these core principles of the experience-based agent approach include:

- **memory retrieval is asynchronous:**

memory retrieval operates in parallel with other processes within an agent, searching the knowledge store continually and incrementally, enabling it to

respond to retrieval requests either in an anytime fashion, returning the “best guess” found so far, or in an spontaneous fashion, retrieving answers as they are found.

- **memory search is context-sensitive:**

memory search unobtrusively or explicitly monitors activity in reasoning tasks or the environment and modifies its search of the knowledge store in an attempt to focus search on the most relevant items.

- **memory search is cost-controlled:**

memory search and retrieval consume a limited amount of resources during any given time slice, allowing reasoning tasks to execute; however, memory can continue to search if given more time and resources in an attempt to improve quantity or quality of retrieval

- **knowledge storage is unified:**

memory search operates over a single store of knowledge, which serves as a repository for all the knowledge that reasoning tasks in the system might attempt to retrieve from memory, ensuring that the search for answers to a vague question can be dynamically focused on appropriate answers when feedback becomes available

- **reasoning is communicative:**

reasoning modules communicate to memory search not only their needs for information but also additional information about their reasoning state and the

quality of retrieved items, enabling context-sensitive memory to focus on appropriate knowledge

- **reasoning is integrative:**

reasoning modules have the capability to accept information as it is retrieved, using integration and control techniques to filter out irrelevant retrieved items and incorporate relevant ones, ensuring that spontaneously retrieved items can profitably contribute to task performance.

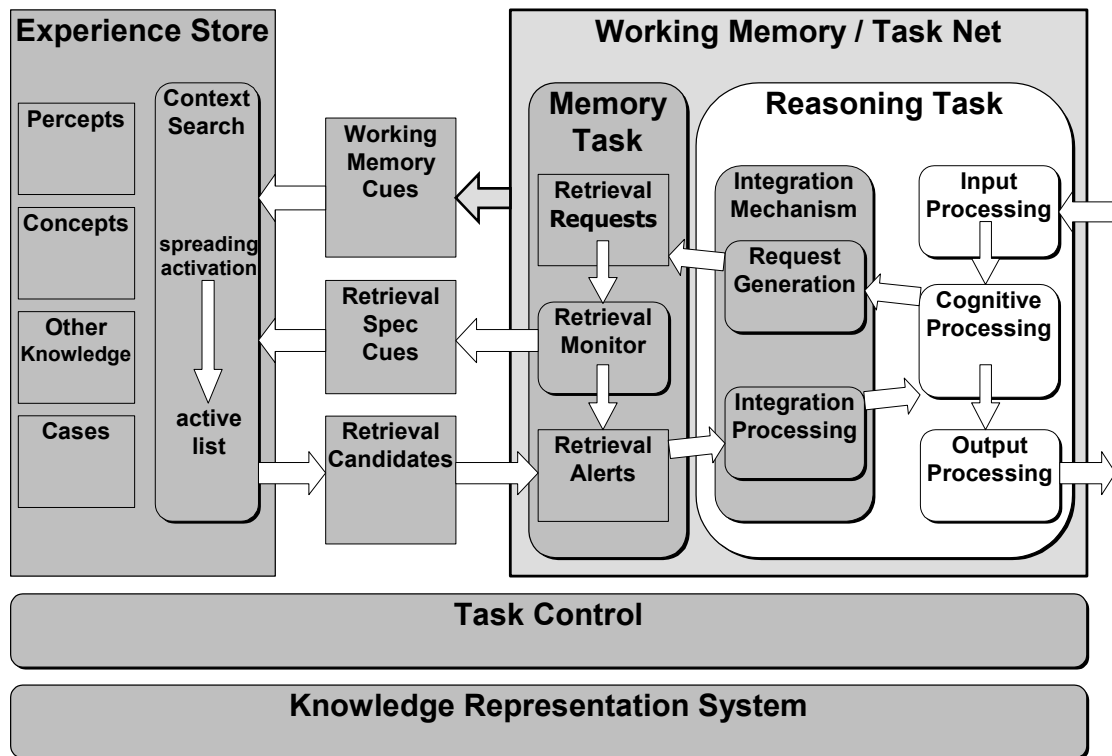
- **communication and control are globally provided:**

the architecture provides facilities to support the control of processes operating in parallel through a global task controller and support communication between memory and reasoning through a global working memory

## 5.2. Architecture of an Experience-Based Agent

To provide these capabilities, an experience-based agent has a set of functional components, including the memory system itself, constraints on reasoning tasks, and the overall architecture within which memory and reasoning operate. Figure 5.1 illustrates these components and their connections: components shown in grey are part of the experience-based agent architecture (or, in the case integration mechanisms, part of the specification) whereas components shown in white are specific to a task. Each of these components has important properties in its own right which enable it to fill its role:





**Figure 5.1 Detailed Architecture of an Experience-Based Agent**

- **The Context-sensitive Asynchronous Memory System:**

The most theoretically important and novel component of the experience-based agent architecture is the context-sensitive asynchronous memory system. Searching in parallel with but guided by reasoning tasks, context-sensitive asynchronous memory both gives experience-based agency its powers of retrieval and demands sacrifices on how it is organized. To briefly review, a context-sensitive asynchronous memory's three most important functional capacities are asynchronous memory search, anytime memory matching, and context-sensitive memory search:

- ***asynchronous memory search:***

An asynchronous memory searches its store of knowledge incrementally while other reasoning tasks proceed, and spontaneously alerts them when answers are found. Asynchronous memory search helps an experience-based agent cope with limited search resources or large knowledge bases, enabling the agent to continue working on tasks while memory searches secure in the knowledge that memory is able to take advantage of additional information generated as it does so.

- ***anytime memory matching:***

An anytime memory can return a “best guess” item based on an weighted match of the items it has examined so far. If a retrieval is requested and a suitable item has been found it should be returned in constant time; alternatively an anytime matching system can retrieve the first suitable item it finds in a “shoot first” fashion while the asynchronous memory continues to search for a better answer. Anytime memory is necessary for processes that need an initial seed of knowledge to begin reasoning and is useful to processes that have algorithms to refine a query based on an initial retrieval. Anytime matching is particularly useful when a knowledge base is large and contains many distractors (irrelevant or weakly relevant pieces of information) which would be difficult to exhaustively match and rank.

- ***context-sensitive memory search:***

Context-sensitive memory alters search of a knowledge base based on the reasoning context. Any piece of information or relationship between pieces of information can serve as a reasoning context, but in this architecture context is primarily derived from a trace of environmental input and reasoning activity. Context-sensitive search is needed to improve the speed and precision of search by focusing the memory system's asynchronous search on the most relevant portion of the knowledge base. This also enables a single memory system to efficiently search a knowledge base used for several tasks, focusing the system's search dynamically on the portions of the knowledge base relevant to the task at hand.

- **The Reasoning System**

For a reasoning task to use spontaneous retrievals to improve its task performance, it must have a set of processes and/or properties which guarantee that the reasoning task can work with an asynchronous memory, provide feedback to that memory, and integrate accept spontaneous retrievals:

- ***work with asynchronous memory:***

Reasoning tasks need to work with the asynchronous memory system, using its store of knowledge to hold important information, routing questions through the memory retrieval system and working in parallel with the ongoing memory retrieval process.

- ***provide feedback to memory:***

The reasoner needs to provide feedback to the asynchronous memory system, both implicitly through use of working memory data structures, explicitly through providing cues to memory, and explicitly through its acceptance and rejection of candidate retrievals.

- ***integrate spontaneous retrievals:***

A reasoner needs to respond appropriately when new information is provided, whether from memory or the environment; this involves at a minimum interruptibility, relevance extraction and reasoning integration. Interruptibility permits a reasoner to actually use spontaneous retrievals when the memory provides them. Dynamic relevance extraction enables a reasoner to extract the portions of a retrieved experience relevant to its current problem solving context. Dynamic reasoning integration enables a reasoner to integrate the relevant portions of an experience into the current reasoning context. Without it, the reasoner cannot take advantage of the spontaneous retrievals even if they have relevant knowledge.

- **The Architectural Infrastructure**

Finally, both the reasoner and memory need to be able to not only work in parallel but also talk to each other. Communication is particularly important because the memory is designed to profit from inspecting the trace of the reasoner's activity. We propose a general architectural framework which serves as a *lingua franca* for

knowledge representation and task communication and which allows this memory system to be instantiated in several different tasks or in a single system for multiple tasks. The features of the architecture include a control system, a working memory, and a global knowledge base called an experience store:

- ***experience store:***

The experience store is a unified knowledge base which represents all agent knowledge in a uniform representational format. This allows a unified memory system to represent knowledge for multiple tasks within a single knowledge base, and provides a rich connection between knowledge items that enables the memory system to take advantage of context.

- ***working memory:***

The working memory provides a standard mechanism for multiple tasks to actively communicate with each other, and for tasks to inspect each other's reasoning traces. The working memory provides the primary source of context for the context-sensitive memory and thus is the central mechanism for feedback between task performance and memory search.

- ***control system:***

Finally, the architecture has a control system which enables multiple tasks to operate in parallel and to communicate with each other. This allows the interleaving of reasoning and memory processes necessary for asynchronous memory search.

Because experience-based agency does not commit to any particular reasoning method or problem solving strategy it is more properly termed a specification for agent design than a complete cognitive architecture for general intelligence. Figure 5.1 illustrates this distinction: the processes and data structures in grey are fully specified by the theory, whereas the details of the reasoning task depend on the design of the agent or system.

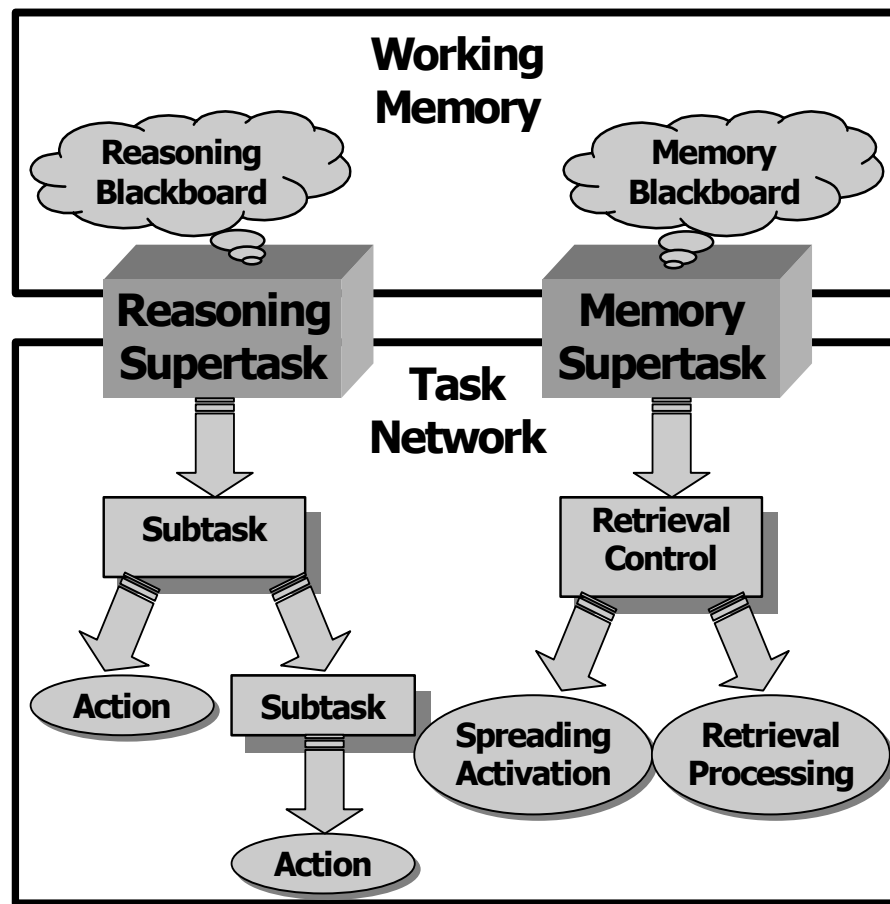
### **5.3. Distinctive Features of Experience-Based Agency**

Many of the features of experience-based agency are inherited from the context-sensitive asynchronous memory approach and will not be discussed further here. Other features of the architecture are more properly constraints applied to reasoning tasks and were discussed in detail in the previous chapter. The most distinctive feature of experience-based agency — the properties that set it apart from other approaches — is that it unifies working memory and task control in a way that supports context-sensitive asynchronous memory.

Many agents can be divided into separate reasoning modules, each of which represents knowledge and performs processing and each of which needs to communicate with and be coordinated with its counterparts. The tools an experience-based agent provides for reasoning tasks to represent information and communicate with each other are a unified working memory and a task controller that uses that memory to guide its behavior. By essentially forcing reasoning tasks to use the working memory as a shared

resource, an experience-based agent makes it possible for a context-sensitive asynchronous memory system to monitor the current reasoning state. By breaking tasks into a hierarchy with an explicit, declarative control structure and small units of execution, an experience based agent makes it possible to interleave or parallelize reasoning tasks and the memory process under the guidance of the task controller.

Experience-based agency achieves this unification by combining the representation of a reasoning task's working memory space and task control architecture within a single data structure called a *supertask*. The term "supertask" was defined by Moorman (1997) to refer to constellations of reasoning modules and communication pathways within an agent designed to achieve a single high-level task, such as sentence processing or story structure understanding. The experience-based agency model modifies this idea slightly, using the term "supertask" to refer to high-level objects, corresponding to major cognitive functions such as memory, reasoning and perception, which structure the system's working memory and spawn specific subtasks to achieve those cognitive functions. A supertask is a top-level task within an agent, operating in parallel with other top-level tasks within an agent, and can be decomposed into a hierarchical set of working memory data structures and a hierarchical tree of tasks, methods and subtasks.



**Figure 5.2 Supertasks Define Working Memory and Task Structure**

From the perspective of memory, supertasks resemble blackboard definitions (Erman et al. 1980, Hayes-Roth & Hayes-Roth, 1979, Nii 1986); from the perspective of the task controller supertasks resemble Reactive Action Packages (RAPs; Firby, 1989), and hierarchical task decomposition architectures such as task-method-knowledge (TMK) systems (e.g., Goel & Murdock 1996). Figure 5.2 illustrates supertasks' dual role as both memory and processing structures.<sup>14</sup>

---

<sup>14</sup> The combination of working memory and control also occurs in other systems such as ALEC (Simina 1999). As discussed in Chapter 2, ALEC's enterprise-directed reasoning model focuses on long-term agent control and is thus complementary to the experience-based agent approach's focus on exploiting



Both the working memory system and task control system of an experience-based agent require some structure to achieve their functional roles. The working memory must be expressive, organized and inspectable to enable reasoning tasks to communicate while allowing memory to unobtrusively inspect their progress; the task control system must support complex reasoning modules operating in both deliberative and reactive fashion to support a wide range of tasks, and must support powerful parallelism and synchronization operators to enable memory and reasoning to interoperate.

In the following sections, we will discuss working memory and task control in more detail and briefly discuss how these techniques can be implemented, using the existing Nicole system as a reference where appropriate.

### **5.3.1. Working Memory**

The goal of a working memory is twofold. First, it should act as a communications mechanism, enabling global communication between all for all tasks in the agent; second, it should act as a monitoring mechanism, effectively “wiretapping” internal agent communications to ensure that the content of reasoning operations is visible to the memory to provide guidance to the context-sensitive search system.

---

context for short-term retrieval. ALEC’s working memory and task structures would easily enable it to exploit the benefits of a context-sensitive asynchronous memory, or to be integrated into an overall experience-based agent approach.

Making a general-purpose working memory depends on, and is made easier by, a general knowledge representation mechanism; however, the requirements for both knowledge representation and working memory for a general cognitive architecture are different from the requirements for memory retrieval in isolation.

A memory retrieval system like context-sensitive asynchronous memory may benefit from a general knowledge representation, but it does not require one. Once the requirements of a task have been fully identified and a content theory for the knowledge needed to perform the task has been outlined, a special-purpose memory retrieval system could be developed that used the context-sensitive asynchronous memory algorithms but a task-specific representation. For example, a specialized CDSA network could be constructed with just the links and relations necessary to store planning cases, in much the same way that PRODIGY/ANALOGY uses specialized redundant discrimination nets to retrieve cases (Veloso 1994); as another example, a dedicated semantic network could be constructed with just the necessary links for a specific kind of document (four different variants include Cohen & Kjeldsen 1987, Salton & Buckley 1988, Crestani & Lee 1999, and Pirolli et al. 1996).

In contrast, a working memory for an experience based agent does not have this luxury because it is designed to support the temporary data needs of a range of tasks and to enable those tasks to communicate by sharing and inspecting each other's data. A working memory for an experience-based agent thus requires and depends on generality: extending to not only the kind of information that can be stored in the working memory

(requiring a general knowledge representation) but also how that information can be accessed (requiring an access language) and how that information can be structured (requiring a working memory definition language).

Therefore, a working memory needs three properties to provide adequate architectural support for context-sensitive asynchronous memory:

- **Expressive:**

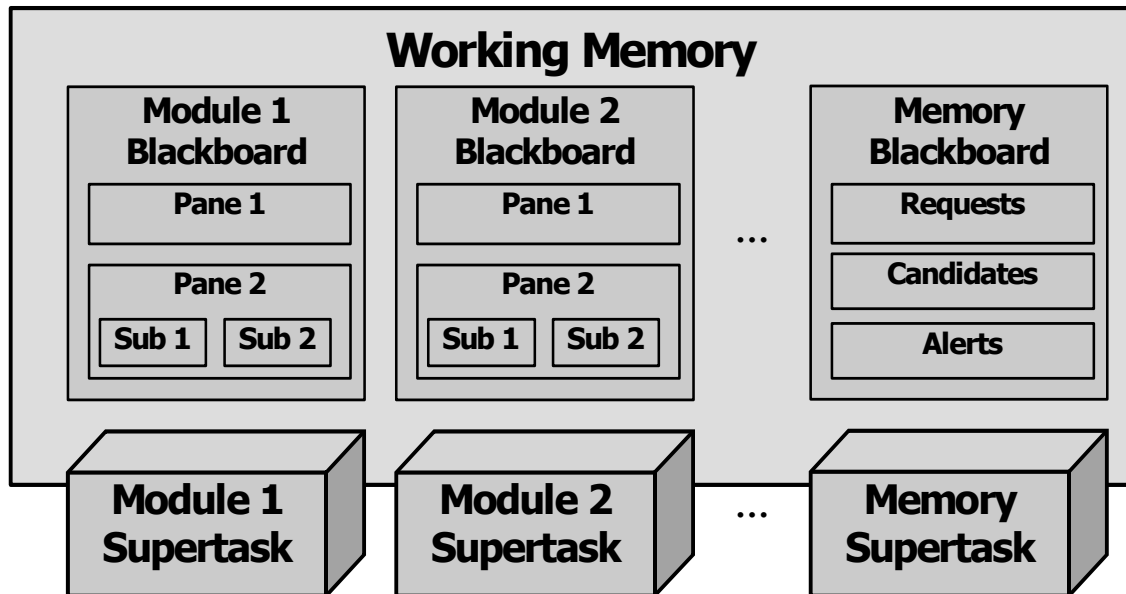
A working memory for an experience-based agent must be capable of storing any information that can appear in the experience store. This includes knowledge for all kinds of tasks — from plans and goals to documents and terms.

- **Organized:**

A working memory for an experience-based agent must be capable of storing and retrieving separately the knowledge used by each task or subtask. Data in the working memory should be organized so that a planning task can distinguish the history of plans it has considered from the current planning frontier, or so that an information retrieval system can distinguish accepted, rejected or retrieved documents from each other, not to mention distinguishing user queries or browsing activity.

- **Inspectable:**

Any knowledge stored in a working memory for an experience-based agent must be visible to all tasks in the system. In particular, in order to support retrieval alerts reasoning task should be able to inspect the alert panes of the memory



**Figure 5.3 Structure of the Main System Blackboard**

blackboard, and to support autonomous cueing memory retrieval should be able to unobtrusively inspect the contents of working memory.

Supplying these properties requires a simple functional interface. An expressive working memory must be able to support the addition of items from the experience store to the working memory, as well as the deletion of those items when they are no longer needed. An organized working memory for multiple tasks should segregate its data into separate, hierarchical storage units, based on a specification of the tasks in the system and the types of data manipulated by each task. As part of both being organized and inspectable the working memory should furthermore provide a language to access and manipulate that content based on that specification. Inspectability also requires that adding or removing items to working memory can trigger actions, such as alerting reasoning tasks or spreading cues.

One potential implementation of a working memory that meets these functional requirements is a blackboard (Newell 1962, Erman et al. 1980). A blackboard is a hierarchical, frame-like data structure divided into planes and panes organized around the types of tasks and knowledge within an agent.<sup>15</sup> Blackboard architectures also have the advantage that they have been successfully used to model human psychological data (e.g., Hayes-Roth & Hayes-Roth 1979) and as implementations for complex artificial intelligence tasks (see the overview in Nii 1986; a more recent example is Hayes-Roth 1995). Furthermore, blackboards can also provide properties theoretically useful to the CDSA memory retrieval algorithm, such as the ability to limit the number of items stored in a blackboard panes as a way to limit the number of retrieval requests or cues.

It is not difficult for a blackboard system to satisfy the constraints sufficient to implement working memory for an experience-based agent. Blackboard planes and panes should be constructed to hold any type of information that can be stored in the knowledge base, potentially by constructing blackboard planes and panes in the same representation language. The blackboard should be partitionable into divisions specific to each individual task, perhaps by allowing tasks to define their own blackboard panes (either dynamically or task compilation time). The blackboard's operations should be open, allowing tasks to communicate with each other by writing across panes or

---

<sup>15</sup> Blackboards also traditionally contain a control structure called an *agenda* which decides which of various *knowledge sources* can write to the blackboard at any given time. This function is subsumed by the task controller in the experience-based agent architecture.

inspecting the panes of another task. Optionally, the blackboard itself may inspect items which are added to it, actively communicating changes to listening tasks such as the memory system.

The working memory of the Nicole system uses exactly this blackboard-based approach. Blackboards in Nicole are constructed using Nicole's native CRYSTAL knowledge representation. There is an omnipresent main system blackboard (Figure 5.3) whose panes and planes are created dynamically as part of the instantiation of privileged top-level tasks called *supertasks*, described in the next section. Blackboard panes are specific to each task but all tasks in the system can inspect or modify any existing blackboard pane. Finally, the blackboard inspects knowledge items as they are added or removed, optionally signalling the context-sensitive asynchronous memory system to generate cues.

### **5.3.2. Task Processing**

The design of a task controller and the types of task networks that it supports determine the types of reasoning the system could perform. While it is possible to program a wide variety of behaviors in an extremely simple task language — indeed, possible to do “anything in anything” if the base “anything” is Turing-complete — complex interleaving of memory and reasoning are difficult in an impoverished task language which does not support features such as parallelism and synchronization and reactive task execution, and complex reasoning methods are difficult to implement in a

system which does not have deliberative task decomposition. Without these features, building an experience-based agent's memory requests, retrieval processing, and integration mechanisms must be handled through highly explicit, reasoning-task dependent code, eliminating the benefits of using a general agent architecture.

To provide adequate general support for constructing systems which interleave memory and reasoning tasks, a task control system must be Turing-complete, support parallelism and synchronization, and enable both deliberate and reactive processing:

- **Completeness:**

Because it is not possible to enumerate in advance all the ways a reasoning task might interface with a context-sensitive memory system, a task control system should support as wide a range of task organizations as possible. Ideally, it should be Turing-complete within the time and space bounds provided it.

- **Multiple Tasks (Parallelism):**

The architecture should support parallel (or tightly interleaved) execution of multiple tasks (such as memory and reasoning).

- **Task Synchronization:**

The controller should support a variety of mechanisms to enable tasks to coordinate their activities, including blocking (synchronous) and non-blocking (asynchronous) task coordination primitives.

- **Reactivity:**

The control of the architecture must support fast response to urgent environmental demands, either through architecturally-guaranteed real time constraints or through the scheduling of quick and inflexible (Q&I) modules (Pollock 1995).

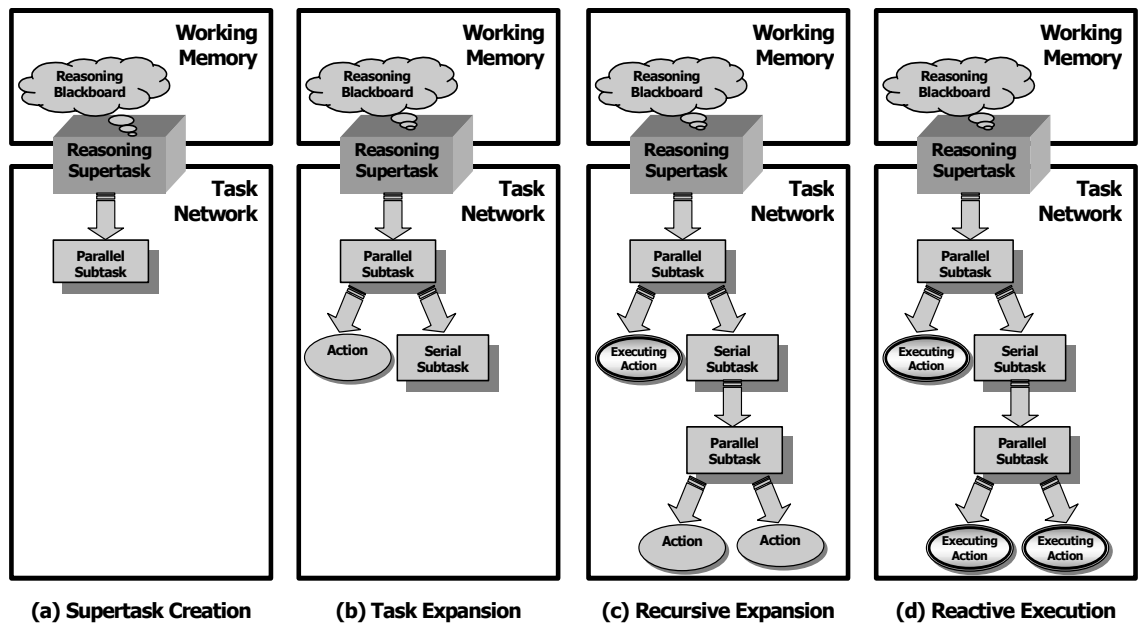
- **Deliberation (Control Flow):**

The architecture should support arbitrarily complex and fine-grained decisions on what tasks to execute and in what order, up to the limit of execution of basic actions or Q&I modules.

These qualities can be achieved using a task control architecture employing *parallel recursive reactive task decomposition* in concert with a global working memory. Parallel recursive reactive task decomposition deliberately expands a set of hierarchical top-level task *descriptions* into a dynamically constructed reactive task *controller*, composed of executing atomic task modules arranged in a hierarchical control tree. Both the decomposition of the task descriptions and the reactive execution of the controller are based on the current contents of working memory, and may modify the contents of working memory as a result of their execution. Furthermore, both the hierarchical structure of the task and the working memory data structures it uses are defined by the same root data structure, the supertask definition for the task.

This model shares a common heritage with systems such as RAPs (Firby 1989), TMK models (Goel & Murdock 1996) and generic tasks (Chandrasekaran 1989). But it also





**Figure 5.4 Decomposing a Supertask in Nicole**

shares some properties with systems ACT\* (Anderson 1983) in that much task processing occurs through the operation of productions and with systems like Soar (Newell 1990) in that task processing steps can impasse if appropriate knowledge is not available.

**Parallel Recursive Reactive Task Decomposition.** In parallel recursive reactive task decomposition, the root of the task system consists of several supertasks (Figure 5.2) created at system initialization time, each defining a working memory structure and a root task (Figure 5.4a). All supertasks are processed in parallel by the **processSupertasks** method in Figure 5.5. An implementation of this system may opt to simulate parallellism through a timeshared or interleaving process.

#### Parallel Recursive Reactive Task Decomposition Algorithm

```
processSupertasks():  
  For each Supertask s do in parallel  
    processTask(s.rootTask)
```

**Figure 5.5 Top-Level Parallel Recursive Task Decomposition Algorithm**

#### Recursive Task Processing Algorithm

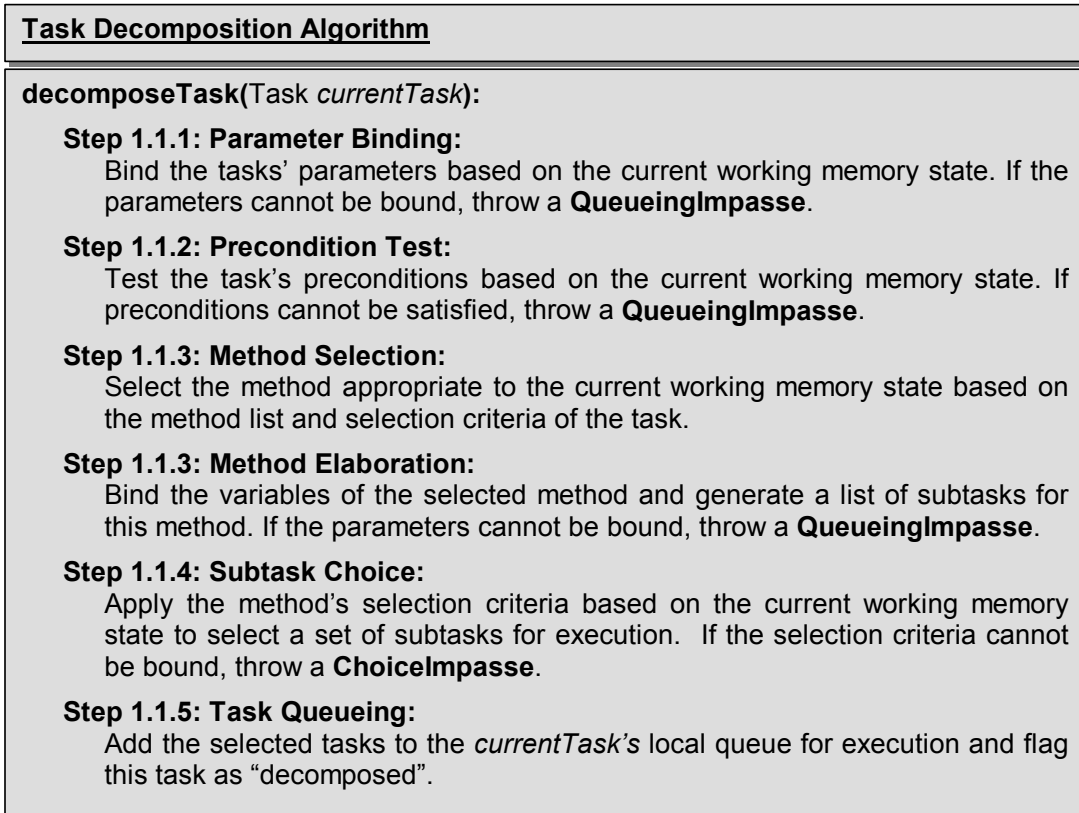
```
processTask(Task currentTask):  
  Step 1.2: Precondition Test:  
    If the preconditions of the currentTask are not satisfied then  
      throwQueuingImpasse(currentTask).  
  Step 1.1: New Task:  
    Else if the currentTask is new then decomposeTask(currentTask)  
  Step 1.2: Queued Task:  
    Else if the currentTask is not impasssed then dispatchTask (currentTask)  
  Step 1.3: Termination Test:  
    If canTerminate(currentTask) is satisfied then remove it from it's parent task's  
    task queue.
```

**Figure 5.6 Recursive Task Processing Algorithm**

Beneath the supertasks, execution becomes a more complex mixture of recursive decomposition and reactive execution carried out by the **processTask** method in Figure 5.6. Beginning with the supertasks and recursively thereafter, each task in the system is decomposed based on the current contents of the working memory. This process is recursive because each tasks can be *compound tasks*, containing a *task queue* with further levels of subtasks (the boxes in Figures 5.2 and 5.4), or *atomic tasks*, forming a unit of reactive execution (the ovals in Figure 5.2 and Figure 5.4). Compound tasks can be either serial, supporting the execution of one subtask in their task queue at a time, or parallel, supporting multiple subtasks in the task queue executing at once (Figure 5.4b and c).

On each cycle of the parallel recursive reactive task decomposition algorithm, all supertasks are processed in parallel by processing the root tasks of each supertask (Figure 5.5). If a task has not yet been decomposed (Figure 5.6), it is handed to the decomposition algorithm for expansion into subtasks (for compound tasks) or instantiation (for atomic tasks) (Figure 5.7). Once a task has been decomposed, the task is “dispatched” for further processing depending on its type and its parallelism and synchronization configuration (Figure 5.8). This dispatching is where the recursive processing of subtasks in the tree occurs. Ultimately, however, the task tree bottoms out in atomic tasks which, once instantiated and elaborated, are executed immediately (and potentially reactively) based on the current state of the working memory blackboard (Figure 5.4d). Atomic tasks execute until their termination conditions are met or their parent task terminates (Figure 5.9).

A running supertask is thus a recursively, iteratively decomposed tree of subtasks, bottoming out in a set of atomic tasks which execute reactively. Task decomposition, recursive processing and reactive execution all occur simultaneously; note that the atomic task shown in Figure 5.4b begins executing in Figure 5.4c, a full cycle before the compound tasks expanded in Figure 5.4c begin executing in Figure 5.4d.



**Figure 5.7 Task Decomposition Algorithm**

Recursive task decomposition can furthermore support whatever degree of complexity necessary to support the real-world tasks that the agent is performing, including deliberate decision making, coordination and synchronization between parallel executing task branches, and task expansion and selection based on computed functions. Figure 5.4 illustrates this process: tasks (boxes) are decomposed until they are unpacked into primitive actions (the round circles) which can be immediately applied on each action cycle (the “atomic” case in the task dispatch function found in Figure 5.7).

#### **Parallel Task Processing Algorithm**

```
dispatchTask(Task currentTask):  
  case (currentTask.type)  
    Case 1: Atomic Task:  
      If the currentTask is atomic then execute(currentTask.codelet)  
    Case 2: Parallel Task:  
      If the currentTask is parallelized then  
        for each in the currentTask.taskQueue do in parallel  
          processTask(currentTask)  
    Case 3: Serial Task:  
      Else if the currentTask is new is serialized then  
        select the next task in the local task queue  
        do processTask(currentTask.nextTask)
```

**Figure 5.8 Parallel Task Processing**

**Task Decomposition.** To build this tree, the root task must be decomposed into subtasks and ultimately atomic tasks. There are five steps to task decomposition in this architecture: method selection, method elaboration, subtask choice, task queueing, and task application.

Once a task has been queued for execution, a method must be selected to actually execute a task. A method, which may specify a complex network of subtasks acting serially or in parallel, must be elaborated to propose a set of subtasks for execution. Those tasks are evaluated and, depending on the task network structure, one or more are chosen for execution. If the chosen subtask's parameters can be bound and its preconditions satisfied, the subtask will be queued for execution, and finally will be applied — recursively decomposed if it is a complex task, or executed immediately if it is atomic.

### Parallel Synchronization/Termination Algorithms

```
canTerminate(Task currentTask):  
  case (currentTask.synchronization)  
    Case 1: Anytime Termination:  
    return terminationConditionSatisfied(currentTask)  
      or postconditionSatisfied(currentTask);  
    Case 2: Blocking Termination (Run to Completion):  
    case (currentTask.type)  
      Case 1: Atomic Task:  
      return postconditionSatisfied(currentTask);  
      Case 2: Parallel Task:  
      If each subtask in currentTask.taskQueue canTerminate(subtask) then  
      return true.  
      Case 3: Serial Task:  
      If currentTask.taskQueue is empty then return true.
```

**Figure 5.9 Parallel Task Synchronization and Termination**

Figure 5.10 illustrates the task decomposition process from the perspective of a task waiting in the execution queue. First, its parameters are bound and precondition satisfied, then, a method is selected. That method is then elaborated to select a set of candidate subtasks, one or more of which may be chosen for execution.

**Support for Parallel Task Execution.** The task control system contains a variety of operations designed to allow a mix of subtasks operating in series or parallel to coordinate their operation. This language includes defining types of tasks, defining the parallel operation of those tasks, how they synchronize with each other, and how they can iteratively repeat.

- **Compound and Atomic Tasks:**

Tasks must come in at least two flavors: **atomic** tasks which contain executable

code and **compound** tasks which contain sets of subtasks. Optionally, a task language may contain other kinds of tasks specialized for particular functions.

- **Compound Task Parallelism:**

Compound tasks must come in at least two flavors: serial tasks whose subtasks are executed in a specific order and parallel tasks whose subtasks are executed in parallel or effectively interleaved. Optionally, a task language may support other kinds of parallelism, such as **OR** tasks which are considered to execute if any one of their subtasks can complete in any order and **AND** tasks which wait until all subtasks are completed in any order.

- **Parallel task synchronization:**

Parallel tasks require at least two kinds of synchronization flags: **anytime**, in which the parent task is complete as soon as its termination conditions are satisfied, or **blocking** or **run to completion**, in which the parent task is not considered to be complete until its subtasks are complete. Another useful flag is **race**, in which the parent task is considered to be complete as soon as any of its subtasks completes, regardless of the completion state of other subtasks.

- **Task Iteration:**

Tasks may run **once**, terminating when their subtasks are complete or when their termination conditions are satisfied, or may be **repeat** tasks which repeatedly reinstantiate their subtasks once their subtasks have completed.

- **Task Control:**

Finally, tasks require a rich set of control operations to enable them to be coordinated. In addition to a task's **preconditions**, **parameters**, and **termination** conditions, tasks may also contain **abort** conditions which force the termination of a task even if its conditions are not satisfied or may contain **loop** tests to control iteration. Conditionals are written in the task conditional language discussed in the next subsection.

This language for task definition enables the construction of complex tasks with many subcomponents which may operate in series or parallel or iteration, which may coordinate with each other to ensure that multi-part computations are completed correctly, and which may abruptly wait, resume, or terminate depending on the state of some other computation to ensure that information is responded to appropriately as it arrives.

**Task Condition Language.** Just as a context-sensitive asynchronous memory system requires a language for matching candidate items, so does a task controller require a language for finding information, binding parameters, and testing conditions. The basis of this language are content tests of specific blackboard panes, organized and composed by logical and imperative operators:

- **blackboard specification:**

Similar to the memory matching language, the working memory provides a



language to specify individual blackboard panes within the hierarchy using lists of blackboard labels beginning from the main system blackboard, or by specifying a blackboard object explicitly.

- **content tests:**

Once a blackboard pane has been specified, its content can be tested to determine whether it satisfies various criteria. This can include whether the content of a pane **equals** some predefined value or other variable, whether a pane **contains** a particular item, or whether the pane **contains** any items at all.

- **parameter binding:**

A similar set of tests can be used to extract content from blackboard panes and bind them to variables for use in future tests or as input to an atomic task's executable module.

- **logical and compositional operators:**

These content tests of blackboard panes can be composed together using a variety of logical operations, including the canonical **and**, **or**, **not** and utility operations such as **if** and **when**.

- **imperative operators:**

Conditional tests can also include special operators designed for control flow, such as **always** and **never** (used most often in precondition and termination tests), **once** (used for tasks that should run only one time) and **wait** (used for explicit task synchronization).

In addition to the language of conditional operations, there is a similar language for writing information to blackboard panes, including set, add, and clear operations. For more details on how a task condition language might be implemented, see the examples in Chapter 6.

## **5.4. Further Support for Memory and Reasoning Integration**

Beyond basic features such as support for task to memory feedback and memory/task interleaving the possibility exists for further integration of reasoning and memory. For example, two classes of impasses in the task system — queuing impasses and choice impasses — provide opportunities to automate asynchronous memory retrieval.

### **5.4.1. Impasse-Driven Retrieval Request Generation**

When no productions or atomic tasks exists to implement a task processing step, the task system can impasse. In particular, when a method specifies that a subtask must run to completion and that subtask's preconditions cannot be satisfied, the result is a *queuing impasse* (shown as the top impasse in Figure 5.10). One way to resolve a queuing impasse is to retrieve an item from memory; traditionally, this has been done in the implementation with reasoning-task specific code.

However, a task's preconditions and parameters can specify both the type and structure it needs, as well as where in the working memory that knowledge should appear. It just so happens that the specification of the type of memory needed for a task is

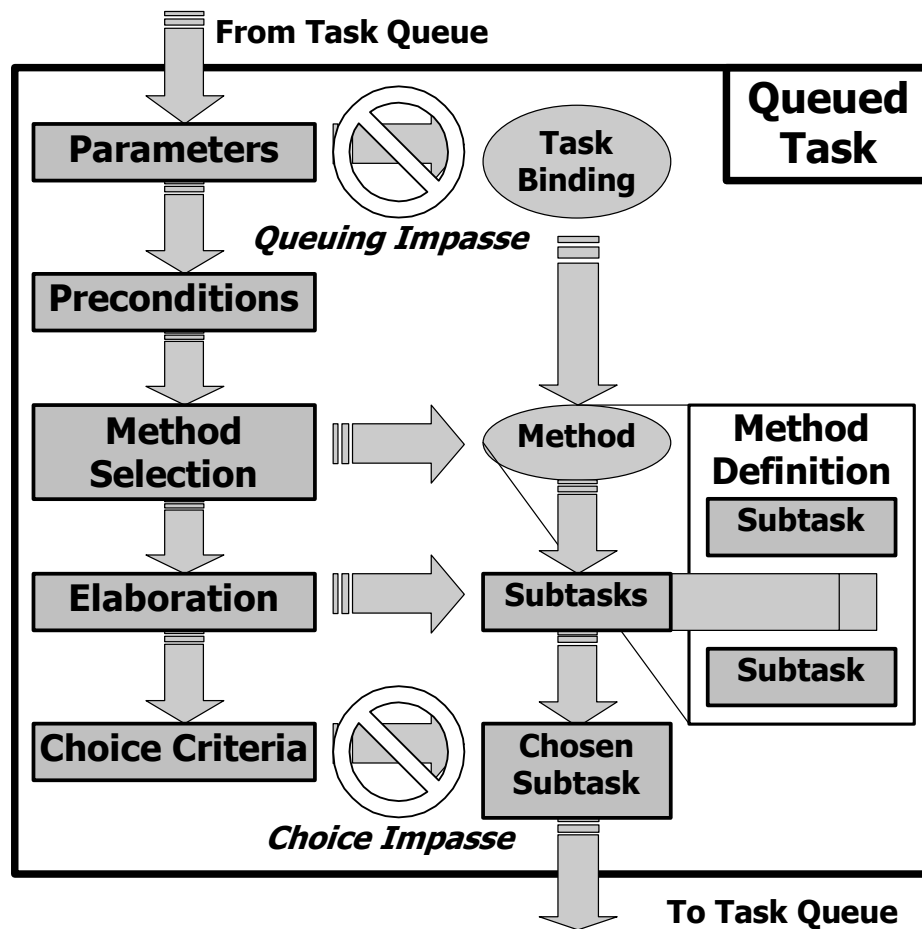


Figure 5.10 Potential Impasses in Task Decomposition.

almost identical to the core of a retrieval request to the memory system (although memory retrieval specifications can be further elaborated). Thus, when a queuing impasse occurs, a memory retrieval request can be automatically spawned and processed in parallel; once an item has been retrieved spontaneously, it can be posted to working memory, allowing the task to proceed.

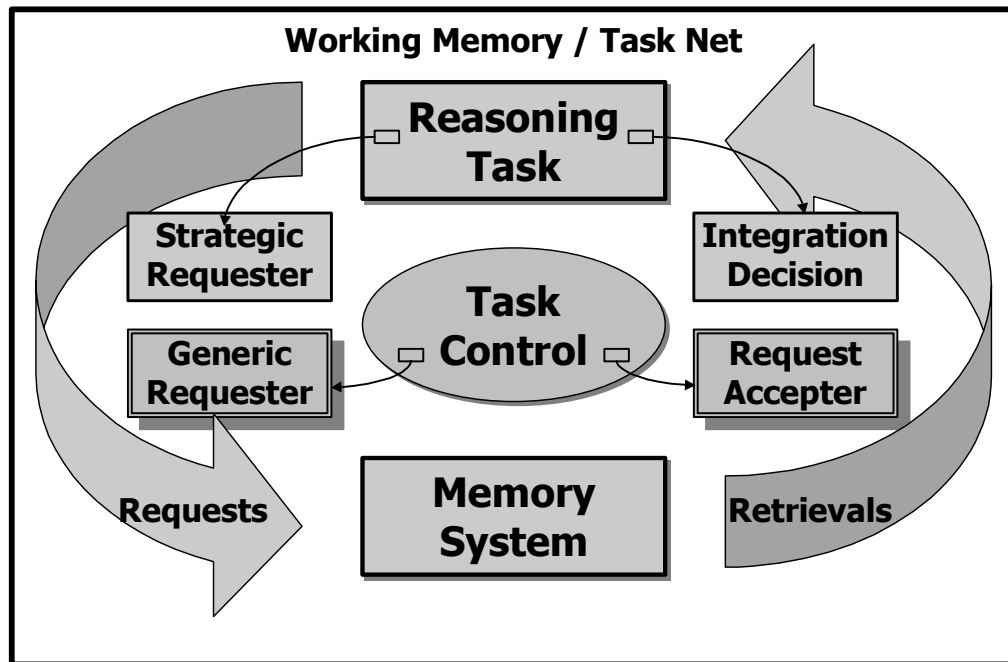
While this method does not encompass high-level strategic memory requests, it does provide an automatic way to detect and satisfy a reasoning task's needs for information from long-term memory, something that previously had to be done by hand. Impasse-driven retrieval request generation is thus a weak method for knowledge retrieval — a

general method for performing a task which formerly had to be accomplished through knowledge-intensive problem solving methods. Figure 5.11 illustrates how this method elaborates our original picture of asynchronous memory retrieval by explicitly separating strategic and “normal” retrieval and by the addition of generic request mechanisms; note how the generic requester and the request acceptor are now subtasks of the task control system, rather than the reasoning task.

This raises another question: once this information is retrieved, how can we be sure it will not disrupt the rest of the reasoning process — which may have been elaborating and executing other tasks in parallel?

#### **5.4.2. Impasse-Driven Integration Mechanism Invocation**

The solution to this dilemma again lies in how a task method is specified. Just as a method can specify that a task must run to completion, it can also specify that out of several alternative subtasks, only one may be chosen. When no information exists to choose a subtask, we have a *choice impasse* (shown as the bottom impasse in Figure 5.10).



**Figure 5.11. Impasse-Driven Retrieval Request Generation**

Integration mechanisms encapsulate (implicitly or explicitly) three types of knowledge: how to prepare raw retrievals from memory to make them suitable for a particular reasoning context, how to actually merge the prepared retrievals into the current reasoning context, and evaluation metrics on when this retrieval is fruitful. While a great deal of preparation can be done in parallel to ongoing reasoning tasks, merging cannot; by definition merging represents a departure from the course of ongoing reasoning. This represents a natural choice point, and unless some information is available to discriminate between these choices a choice impasse will arise.

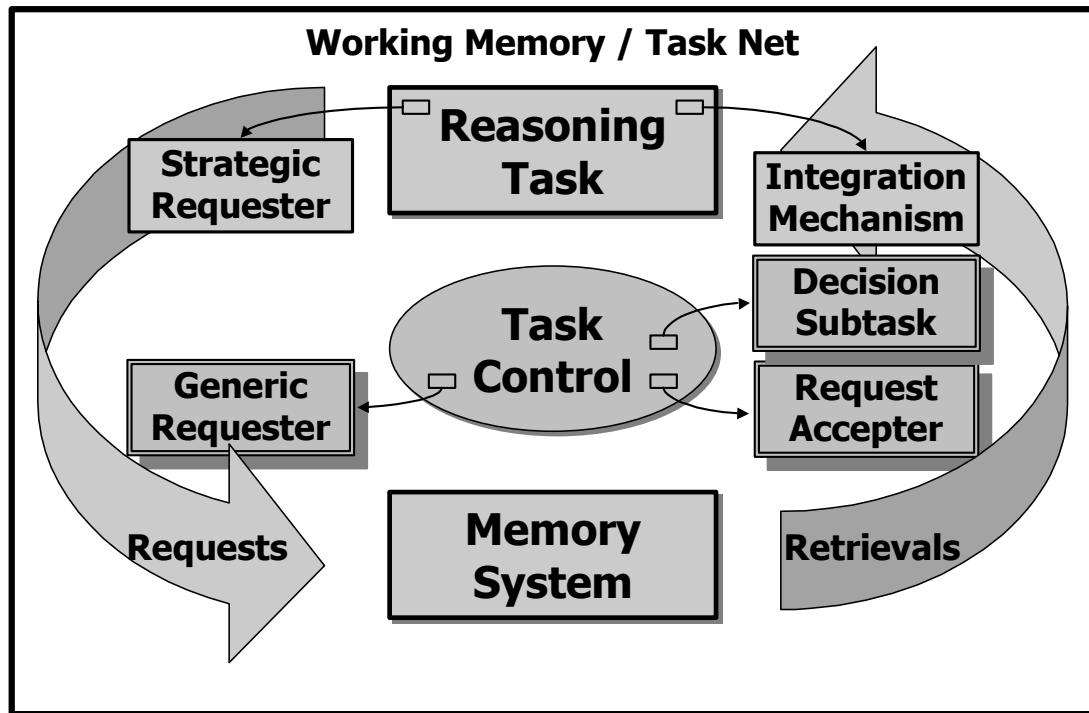
Note that since merging subtasks are by definition optional, no queuing impasses will arise if their preconditions cannot be satisfied. Only if some other task satisfies their preconditions for them — such as a preparation task, which can run to completion in

parallel and can hence generate memory retrievals — will they be candidates for queuing, and only then will a choice impasse arise. When the choice impasse does arise, the evaluation subtask of the integration mechanism can be invoked, allowing the system to decide whether to continue reasoning or to attempt to merge.

Because integration mechanisms are intimately tied to the reasoning processes, no completely general method can be devised to automatically perform integrations or to evaluate their utility. However, using choice impasses as the vehicle to orchestrate integration mechanisms gives us a clear account of the content needed in an integration mechanism, a structure for the storage of that content, and a uniform mechanism for its execution. Figure 5.12 illustrates this in action: even though a choice mechanism may be provided by a reasoning task, it is invoked automatically as a subtask of the task controller when a choice impasse occurs.

### **5.4.3. Significance of Impasse-Driven Memory Processes**

Using explicit impasses to automatically generate memory retrieval requests and to organize the implementation of integration mechanisms makes the task of constructing an experience-based agent easier, but why should the designers of intelligent agents care about these methods, especially if they aren't using an explicit instantiation of the experience-based agency theory? The answer is twofold: first, this method teaches a general lesson about intelligent agents in particular, and second, this method makes an



**Figure 5.12 Impasse-Driven Integration Decisions**

agent built on the experience-based agency theory in particular a more viable choice for an implementation.

First, spontaneous memory retrievals coupled with integration mechanisms are powerful tools to simultaneously deal with both novel information from the environment and to enlist the an agent's own reasoning processes (through the working memory trace) in the task of effectively exploiting the agent's past experiences. But making asynchronous integration work requires a powerful controller and a methodology for choosing when to retrieve and when to integrate. Even if the actual implementation of a system diverges wildly from the type of task decomposition used in an experience-based agent, a task-method-knowledge analysis (TMK) of the reasoning process within an

agent (Goel & Murdock 1996) can provide pointers of where to retrieve, where to prepare, where to merge, and where to choose to integrate.

Second, with the addition of this method, designers building systems explicitly based on experience-based agency principles no longer need to develop retrieval request generators, retrieval acceptors, integrators and choice routines in an ad-hoc fashion. The control and execution of these components can be more directly controlled by the architecture through the interaction of memory and task control (and to a lesser extent by the interaction of task control and specific reasoning mechanisms).

## **5.5. General Architectural Issues**

Clearly, achieving generality in an architecture designed for real-world components is not limited to ensuring that the knowledge representation language can express higher-order relations or that the task-control mechanism is Turing-complete. Additional issues, such as scaleup to large knowledge bases or across many task domains, guarantees on reactivity or performance, and integration with realistic sensors and effectors, must also be dealt with. This research has attempted to address issues dealing with scaling to large knowledge bases and to multiple task domains, but does not deal with the latter issues.

Developing artificial intelligence systems with guarantees on reaction time and performance is a complex area of artificial intelligence with many open issues beyond the scope of this research (e.g., Mouaddib & Zilberstein 1998, Musliner 2000). For example,



building a real-time system requires dealing with constraints of real-world tasks, low-level processing and efficiency issues, and even the reliability of the underlying software (Chen et al. 1995) furthermore, developing guarantees on performance requires an in-depth formal analysis sufficient to make provable logical statements about a system (Craig 1991).

Integrating with real or even simulated sensors and effectors raises its own issues. Both robotic systems and simulation environments are complex and impose many constraints on how systems may interface with them. Issues such as noisy sensors, unreliable effectors, command and control languages, and again real-time processing constraints must be dealt with to build an agent which can successfully operate using real sensors and effectors (McKerrow 1991, Jones & Flynn 1993) and similar complexity arises in the kinds of simulation environments of most use for evaluating the context-sensitive asynchronous memory and experience-based agent approaches — for example, one exploratory test using the PHOENIX simulator (Cohen et al. 1989) showed that the bulk of the effort in porting the experience-based agent architecture to the simulator would be designing an agent to perform the task, not testing the particulars of the theory.

While answering questions about sensor integration and performance guarantees are important, they raise new problems which are at best orthogonal to the problem of the problem of efficiently finding good answers to bad questions in a general fashion, and at worst major unsolved problems in artificial intelligence deserving of focused

dissertations in their own right. For these reasons, these issues are beyond the scope of this dissertation.

## **5.6. Benefits of the Experience-Based Agent Approach**

What good is the experience-based agent approach over context-sensitive asynchronous memory itself?

### **5.6.1. What Experience-Based Agency Provides**

Experience based agency is intended to be a general method of information access, applicable to and efficient enough to serve the information needs of a wide range of tasks. To do so, it provides architectural support for applying context-sensitive asynchronous memory to be applied to a variety of different reasoning tasks, and provides a framework to structure those tasks to respond appropriately. Furthermore, the experience-based agent approach is designed to be a unified method of information retrieval, capable of not only being applied to several reasoning tasks in isolation but potentially applicable to several reasoning tasks operating at the same time within the same agent. To achieve these functions, this approach utilizes a variety of strategies to control cost and exploit contextual information, which in concert enable it to scale to large or multifunction knowledge bases or to adjust its search to suit current information needs.

### **5.6.2. What Experience-Based Agency Enables**

The ultimate benefit of the experience-based agent approach is not to find a better way to solve just one sub-problem (memory retrieval) which has been solved before, but instead to provide a more general, flexible solution to memory retrieval which enables systems that require memory to solve their problems better or to solve them in novel ways.

Therefore, the experience-based agent approach is designed to enable new, collaborative strategies for interaction between reasoning and memory, including "proactive" memories which monitor reasoning tasks, "opportunistic" memories which exploit environmental cues, "integrative" reasoners which exploit proactive and opportunistic retrieval, and "parsimonious" memories which naturally adjust their effort based on available resources. This approach has already been used to construct a prototype integrative planning system called Nicole-MPA, which in certain circumstances can achieve better performance than the original planning system upon which it was based (as discussed in the case study on Nicole-MPA in Chapter 8). This enables the construction of more flexible, complex intelligent systems, which we claim could potentially solve more difficult problems, solve problems faster, or perform tasks which were previously unsolvable — systems which would push the state of the art closer to full cognitive agents, at least as far as memory is concerned.

Because of its ability to return a best guess, an experience-based agent has many similarities to the interactive “shoot-first” case-based reasoning systems advocated by Riesbeck (1993). Given a problem input by the user, a shoot-first system attempts to quickly retrieve an approximate set of cases and/or propose a sketchy solution, and then uses feedback from the user to redefine the problem, refine the search, or adapt the solution. Both experience-based agents and shoot-first systems require similar memory capabilities; however, the primary “user” of an experience-based agent’s quick retrievals is the agent itself.

### **5.6.3. Exploiting Experience-Based Agency**

Building an agent that exploits the potential of experience-based agency begins with the experience store. All the knowledge that an agent might want to retrieve for any of the reasoning tasks it performs should be stored in the experience store; to enable it to be retrieved it should be richly interconnected and grounded in the kinds of sensory data and concepts the agent is actually likely to encounter in solving its problems.

The next step is the context-sensitive nature of memory. As the agent solves its problems it should be designed to make information relevant to its current task state available to the memory to guide its search. This context can be made available either overtly through explicit cues or covertly through working memory activation, but it should contain not only objects and concepts in the world that the agent observes but also the relationships relevant to the task the agent is performing.

Then, the agent should exploit the asynchronous nature of an experience-based agent's memory. This can take the form of demanding fast anytime retrievals, updating retrieval specifications, or exploiting asynchronous retrievals as they arrive. As new information is retrieved and provided by the memory the agent should use its integration mechanisms to dynamically incorporate as much of that information into the reasoning state as possible, and should provide feedback to the memory about how well it is doing to enable the memory to improve its performance.

Finally, the task of exploiting experience-based agency returns to the experience store. Like a case-based reasoner, the results of an agent's problem solving should be stored in the experience store to improve future retrieval. But an experience-based agent should do more than just store cases: with every action the agent takes should add to the experience store, building knowledge of the connections between the agent's environments and the solutions to the problems it is having into the structure of the experience store to aid future retrieval. Storing not just solutions to problems but the content of reasoning traces and how that reasoning connects to environmental features enables the retrieval of experiences in relevant circumstances later through the context-sensitive asynchronous memory process; for a complex reasoner which performs many tasks, it further enables an experience to retrieve past knowledge in many different circumstances in which it might be useful.

This process — representing knowledge in the experience store, using context to find it, using asynchrony and integration mechanisms to exploit it, and then storing the results

of that exploitation back into the store — is the heart of the experience-based agent approach to reasoning.

#### **5.6.4. Living up to Experience-Based Agency's Potential**

The experience-based agent systems described in the case studies, Nicole-MPA and Nicole-IRIA, begin to realize some of this potential. Nicole-MPA shows how to practically implement a system which can integrate multiple experiences, and Nicole-IRIA shows how feedback can be used to painlessly improve and to organize the presentation of information to a user. However, these systems only tap part of the potential of the experience-based agent approach; future research should continue to apply the approach to more and more complex tasks to both provide additional benefits and to illustrate the limits of the theory.

For example, a full integration of the EBA theory into a story understanding system, a process begun in the ISAAC system but not yet complete, could provide important tests of context-sensitive retrieval based on feedback from a rich reasoning trace. As another example, the *enterprise-directed reasoning* model of invention used in the ALEC system (Simina & Kolodner 1995, Simina 1999) models agents with multiple open goals or enterprises, a current work context, and an external environment that can provide new information at any time. Integrating the EBA and EDR models could flesh out both theories, providing to experience-based agency a model of high-level goals and reasoning and to enterprise-directed reasoning a uniform memory retrieval model. A combination

system could also serve as a further testbed for context-sensitive memory retrieval driven by both internal and external events. Finally, complex design and problem solving tasks which deal with changing constraints generated by external events, such as the JULIA system (Hinrichs 1992), could provide a rich platform to test the benefits of experience-based agency's memory driven reasoning integration.

## **5.7. Summary**

This chapter presented experience-based agency, a general framework for memory retrieval and integrative reasoning which enables the context-sensitive asynchronous memory approach to be applied to a variety of tasks.

The experience-based agent approach provides pervasive architectural support for context-sensitive asynchronous memory. It begins by coupling context-sensitive asynchronous memory with a semantic network experience store, and adds to this a working memory system, task control system, and task/memory communications language which enable the construction of complex reasoning tasks which can work with and provide feedback to memory.

In addition, the experience-based agent framework includes a specification of how reasoning tasks should be constructed to satisfy the demands of the approach, along with a specification of the conditions under which the approach will provide benefits.

Together, these elements of experience-based agency specify all the parts of a complete reasoning system based on a context-sensitive asynchronous memory except the core task-specific parts of the reasoning system itself. The challenge of building a complete agent built around a context-sensitive asynchronous memory is thus reduced to the challenge of adapting a reasoning task to work within the experience-based agent architecture. The opportunity, then, is adapting reasoning tasks in such a way that they exploit the full potential of the experience-based agent architecture.



# PART III.

## EVALUATION

---

### *Evaluation of Context-Sensitive Asynchronous Memory and Experience-Based Agency*

Context-sensitive asynchronous memory and experience-based agency touch on many areas of artificial intelligence. To evaluate these approaches I used a three-pronged evaluation strategy including implementation, case studies and feasibility evaluations. Chapter 6, Methodology, discusses in more detail the motivation behind these evaluations.

The first evaluation strategy was a proof of concept implementation. Because the context-sensitive asynchronous memory and experience-based agent approaches are based on novel models of memory retrieval and agent architecture, I constructed a system based on the approaches called Nicole to verify that these models could actually be implemented. Chapter 7: Implementation discusses the Nicole system.

Second, to explore the generality of the approaches and demonstrate that they could provide real benefits to a range of tasks, I conducted a series of case studies designed to test these models in operation. Chapter 8: Case Study: Planning and Chapter 9: Case Study: Information Retrieval discuss the two most extensive studies of context-sensitive asynchronous memory and experience-based agency in action.

Finally, because both approaches depend on theoretical properties and performance characteristics of the context-sensitive asynchronous model of memory, I conducted a feasibility study to verify that the implemented memory had the properties predicted by the model. Chapter 10: Feasibility discusses these experiments and evaluations.

# CHAPTER VI.

## METHODOLOGY

---

### 6.1. Overview

To validate the major claims of the context-sensitive asynchronous memory approach, as well as to explore how the approach could be applied using the guidelines of the experience-based agent approach, I conducted a series of evaluations.

My method for these evaluations was to implement actual systems based on the approach, to record design decisions made as context-sensitive asynchronous memory and experience-based agency were applied, and then to test the performance of those systems both comparatively and objectively.

The focus of my efforts was on the development of the Nicole system, a toolkit for authoring experience-based agents, and on the implementation of two systems based on Nicole, the Nicole-MPA case-based planner and the Nicole-IRIA information retrieval system. The context-sensitive asynchronous memory approach was also tested in the ISAAC story understanding system.

I tested Nicole-MPA and Nicole-IRIA using a variety of methods, including rigorous experiments of particular properties as well as a set of exploratory evaluations of the systems in conditions designed to illuminate various design choices.

## 6.2. Goals of the Evaluation

All of these evaluations were designed to show how a context-sensitive asynchronous memory could be used to provide real benefit to user tasks, and secondarily to show how the experience-based agent approach was effective at exploiting context-sensitive asynchronous memory for real tasks.

Of course, the goal of using a context-sensitive asynchronous memory for real tasks was not just to provide benefit to those tasks but to show that the approach worked. Therefore, all these evaluations were directed at demonstrating the primary thesis of the context-sensitive asynchronous memory approach:

The context-sensitive asynchronous memory approach is a general and effective method for finding good answers to vague questions from large knowledge bases under resource constraints using feedback from the task and the environment.

An instrumental thesis also tested in these experiments was the viability of the experience-based agent approach to constructing agents based on context-sensitive asynchronous memories:

<p>The experience-based agent approach enables a reasoning task to exploit a context-sensitive asynchronous memory to get useful information.</p>
---

These high-level theses were unpacked into a series of lower-level claims that were tested in the implemented systems. These tests, along with experiences of actually implementing the systems, taught several lessons about how to implement systems based on the approach, how to make the approach work as intended, and what the approach could and couldn't do.

### **6.3. Methodology of the Study**

Testing theses about approaches to artificial intelligence problems requires applying the approach to some task and then conducting a set of evaluations over that application.

To carry out this methodology, I applied the context-sensitive asynchronous memory approach to the general problem of memory retrieval, developing a experience-based agent toolkit that enables the construction of reasoning tasks built around a context-sensitive asynchronous memory.

I then applied the context-sensitive asynchronous memory approach to more specific tasks that required memory retrieval, using the toolkit to implement several performance tasks based on the approach. These performance tasks included case-based planning, information retrieval, and (in collaboration with Kenneth Moorman) story understanding. Applying the approaches to these tasks taught important lessons about how use context-

sensitive asynchronous memory and experience-based agency, showing that some techniques worked and others did not.

I then used these implementations to test the context-sensitive asynchronous memory approach in several ways, including evaluating the effectiveness of context-sensitive asynchronous memory at providing useful information to reasoning tasks, evaluating the capability of experience-based agent approach to exploiting context-sensitive asynchronous memories, evaluating context-sensitive asynchronous memory's ability to meet the desiderata for a general memory system, and analyzing what features context-sensitive asynchronous memory needs to meet those desiderata.

Using the case-based planning and information retrieval applications, I constructed a set of data sets and experimental evaluations designed to test how well context-sensitive asynchronous memory and/or experience-based agency functioned at providing to the tasks the information that they needed. I furthermore conducted tests with these implementations designed to examine the properties of context-sensitive asynchronous memory itself and its sources of power.

## **6.4. Applying the Model**

The target tasks were chosen to demonstrate the novelty of the approach. While the experience-based agent approach is a general framework for applying context-sensitive

asynchronous memory to a wide variety of performance tasks, some performance tasks are better targets for experience-based agency than others.

Good performance tasks for the experience-based agent approach have a variety of properties. These tasks generally have large knowledge bases or have resource limits on knowledge access, along with difficult to solve problems. Suitable tasks generate contextual information during the reasoning process which mirrors the contextual structure present in the task domain; furthermore, suitable tasks can potentially benefit from the discovery of additional experiences during the process. Information retrieval and planning have many of these properties, making them good targets for the approach.

Experience-based agent approaches to these kinds of tasks are based on the idea of contextually-driven reminding from a store of experience and dynamic integration of those reminders into the current reasoning state. Furthermore, results of these evaluations indicate that the best way to integrate these experiences is through deliberate consideration of the relevance of those experiences to the current reasoning state, rather than driving integration from the timing of retrieval directly. Context-sensitive asynchronous memory provides contextually-driven reminders; the remainder of the experience-based agent approach provides the support for building up the store of knowledge to remind from, generating context and exploiting reminders.

## 6.5. Conducting the Evaluations

To execute the study, I first constructed the Nicole system, an artificial intelligence development system that provides an architectural infrastructure to support the construction of experience-based or other agents. The Nicole system provides core modules that serve as a context-sensitive asynchronous memory, an experience store, working memory and task control; it also provides languages and interfaces to create the knowledge representations, working memory data structures and task control structures of reasoning tasks.

Then, I constructed the Nicole-MPA application, a case-based least-commitment planner that adapts multiple plans based on case-based principles. Nicole-MPA was designed to test whether the experience-based agent approach could enable a reasoning task to exploit a context-sensitive asynchronous memory for improved performance. To explore this, I constructed a set of variant planners, planning domains, case libraries, problems and knowledge bases and used them to conduct a variety of experiments and evaluations on Nicole-MPA's performance as a planning system. Many of these initial tests were unsuccessful, requiring revision of the Nicole-MPA framework and revision of the hypotheses underlying experience-based agency itself. These revisions showed that it was possible that context-sensitive asynchronous memory could support an experience-based approach to improved planning performance, but under restricted circumstances; moreover the successful system did not fully exploit the power of the context-sensitive asynchronous memory approach.



Based on these results, I constructed the Nicole-IRIA application, a search and browsing aid for information retrieval that recommends information based on reminders. Nicole-IRIA was designed to test whether the experience-based agent approach could enable a reasoning task to exploit a context-sensitive asynchronous memory to get useful information. Several applications and data sets were constructed and used to conduct a variety of experiments and evaluations on Nicole-IRIA. These tests showed that a context-sensitive asynchronous memory approach was effective at providing users useful information.

Along the way I conducted experiments on both systems as a test of context-sensitive asynchronous memory itself using Nicole-MPA and Nicole-IRIA.<sup>16</sup> Using Nicole-MPA I conducted a series of experiments to test the context-sensitivity, asynchrony and anytime retrieval capabilities of my context-sensitive asynchronous memory implementation, as well as its raw speed. These experiments were backed up by a series of exploratory and experimental evaluations of Nicole-IRIA addressing similar issues.

## **6.6. Fidelity of the Study.**

The efforts of the study followed the guidelines of the context-sensitive asynchronous memory approach fairly closely and the experience-based agent approach somewhat less closely.

---

<sup>16</sup> The ISAAC system was not available for tests at the time this dissertation was completed.

The Nicole system as a whole implemented all of the support functions of the experience-based agent architecture except automatic cueing based on working memory data structures, and the Nicole-MOORE memory module implemented all of the context-sensitive asynchronous memory model except storage of retrieval requests.

However, the Nicole-based applications constructed did not exploit all of these facilities: Nicole-MPA did not exploit the timing of asynchronous retrieval, exploit additional cases found through feedback to improve its performance, or build up experience; Nicole-IRIA did all of these but made only minimal use of the task control system.

Finally, the study tested knowledge bases ranging over two orders of magnitude in size, these knowledge bases were not as “large” as, for example, the CYC (Lenat & Guha 1990), Botany (Clark & Porter 1996) or UMLS (National Library of Medicine 2000) knowledge bases.

## **6.7. Outline and Expected Results of the Study**

The next several chapters detail the work conducted to carry out this analysis of context-sensitive asynchronous memory. Chapter 7 presents the details of the core implementation, the Nicole system; the following chapters detail the experimental work conducted using the Nicole system and its children. All of these experiments were designed to validate whether context-sensitive asynchronous memory could provide

benefit to real tasks, determine how it could be used to provide those benefits, and to verify that context-sensitive asynchronous memory worked the way that its model predicts.

To do so, the claims about the impact context-sensitive asynchronous memory had on tasks were unpacked into subsidiary claims about planning and information retrieval. If context-sensitive asynchronous memory performed as its model predicts, the experiments should show improved performance on planning and high quality of recommended results on information retrieval. The details of these subsidiary claims and the result of the experiments based on them are discussed in greater depth in the case studies in Chapters 8 and 9.

The effort of applying context-sensitive asynchronous memory during the case studies revealed a number of features about how context-sensitive asynchronous memory could be applied. If context-sensitive asynchronous memory performs as its model predicts, reasoning tasks need certain features to exploit it and exploiting it will be easiest for certain kinds of problems, knowledge bases and contexts. The case studies in Chapters 8 and 9 and the feasibility study in Chapter 10 unpack these lessons learned from applying context-sensitive asynchronous memory, using results not only from the exploratory application of the model but also from follow-up experiments designed to verify the intuitions discovered during that application.

Claims about the performance of context-sensitive asynchronous memory itself were tested with respect to the desiderata for memory systems for general cognitive agents listed in Chapters 1 and 2. If context-sensitive asynchronous memory performed as its model predicts, the experiments should show a variety of properties such as generality, context sensitivity and scaleup. The details of these desiderata and the results relevant to them are discussed in greater depth in the feasibility study in chapter 10.

# CHAPTER VII.

## IMPLEMENTATION

---

*The Nicole agent architecture and toolkit*

Experience-based agency proposes a novel method for memory retrieval based on context-sensitive asynchronous memory, and a novel way of structuring an agent to take advantage of such a memory. While some aspects of the approach could be analyzed in the abstract, to really test the model it must be implemented in a running artificial intelligence system on which experiments and evaluations are conducted.

This implementation is Nicole, a large Common Lisp program instantiating the experience-based agent approach. Named after a sentient computer in a short story written by this author (Francis 1995), Nicole is an artificial intelligence system that provides an architectural infrastructure to support the construction of experience-based or other agents.

The support that the Nicole system provides includes several modules which are artificial intelligence systems in their own right: a core representation language configured to act as a global experience store, a global working memory system, a context-sensitive asynchronous memory process, and a control system for reasoning tasks capable of orchestrating reasoning and memory. Together, these components comprise

all the necessary components of an experience-based agent — short of the reasoning component of an actual performance task and any integration mechanisms it requires.

To facilitate the construction of reasoning tasks that work within an experience-based agent architecture, the Nicole system provides development support tools. First, Nicole's core modules expose their core functionality through application programming interfaces or APIs. Second, the Nicole system provides languages to access those interfaces, including a task language, a working memory specification language, and the aforementioned knowledge representation system. Reasoning tasks developed with these languages thus can take advantage of the remaining components of the Nicole system — knowledge representation, working memory, memory retrieval and task control — which are task neutral and support a range of reasoning tasks.

In addition to these theoretically significant components, the Nicole system also provides a set of support tools which facilitate the running and testing of applications developed in the Nicole system. This includes a system to execute Lisp or system commands under Nicole's control, a system to execute Nicole tasks automatically, and a set of tools for collecting behavior about the execution of the system.

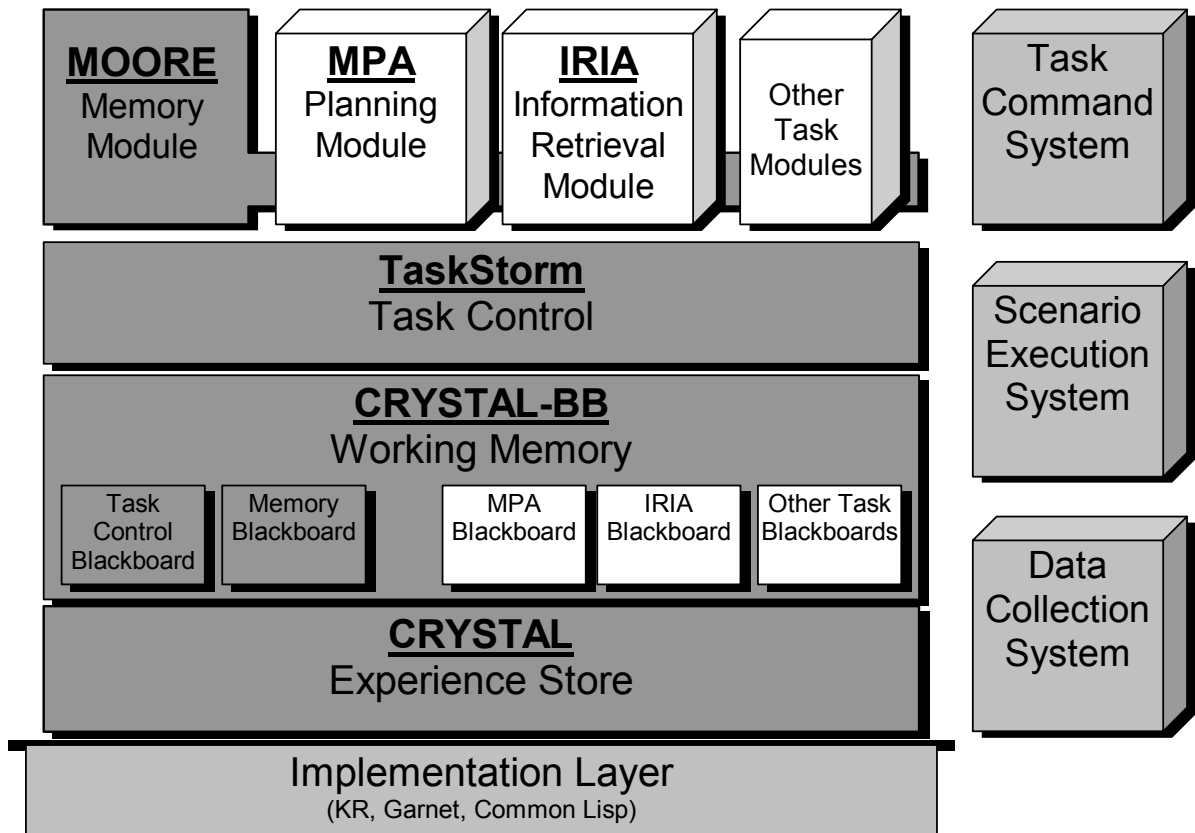
This combination of a core of experience-based agent components, interfaces and languages to construct reasoning tasks that access those components, and support tools for running and analyzing these components, means that Nicole can be described as a development system or authoring tool for the construction of experience-based agent

systems. For clarity of presentation, I will use the terms *Nicole core modules* to refer to the self-contained components of Nicole that provide functionalities like memory retrieval, *Nicole development tools* to refer to the languages and interfaces which enable the construction of reasoning tasks, *Nicole support tools* to refer to theoretically insignificant but practically useful tools for running and analyzing the behavior of those tasks, and *the Nicole system* to refer to the complete package of core modules, development tools and support tools.

The Nicole system's general-purpose design permits the construction of many different *Nicole-based applications*, including experience-based agents. Two such applications constructed with the Nicole system include the case-based planner Nicole-MPA and the information retrieval system Nicole-IRIA, discussed in the case studies in Chapters 9 and 10.

## 7.1. Overview of Nicole

Figure 7.1 illustrates the overall structure of the Nicole system. Boxes are “modules”, or groups of code that perform tasks. Horizontal layers are “services,” which are sets of functions available to all layers and modules shown above them in the diagram. The Nicole core modules are shown in dark grey. Nicole development tools exist in a one-to-one mapping to the experience store, working memory and task system and are thus not shown. Nicole support tools are shown in light grey and representative components of Nicole-based applications are shown in white.



**Figure 7.1. The Architecture of NICOLE.**

### 7.1.1. Nicole Core Modules

The primary artificial intelligence modules that make up Nicole are:

- **CRYSTAL Experience Store:**

CRYSTAL is a semantic network representation language with reason maintenance and memory extensions which forms the Nicole system's experience store. CRYSTAL data structures also form the foundation for all other modules in Nicole, including core modules such as memory retrieval and task control as well as support tools like the executive command and scripted scenario systems.



- **CRYSTAL-BB Blackboard system:**

An extension to CRYSTAL supports hierarchical blackboards, including a privileged “main system blackboard” which is the primary means of task communication in the Nicole system and holds the data structures used by Nicole’s memory retrieval and task control modules. The structure of the blackboard is defined by a set of “supertasks” which are provided to the Nicole system as part of its initial configuration. Blackboards can also be defined dynamically.

- **MOORE context-sensitive asynchronous memory retrieval system:**

MOORE is an asynchronous memory retrieval module, operating over CRYSTAL’s knowledge representation and using CRYSTAL’s blackboard data structures. MOORE is parameterizable for experimental purposes, providing both context-sensitive and traditional memory search modes.

- **TaskStorm task control system:**

TaskStorm is an interleaved, hierarchical task decomposition control module, orchestrating the execution of atomic “tasklets” of Lisp code based on hierarchical compound “tasks” which are decomposed according to the contents of the main system blackboard. The active set of tasks are all children of top-level “supertasks,” provided to the Nicole system as part of its initialization, which also define the structure of the main system blackboard.

These Nicole core modules work together to provide architectural support for experience-based agents. TaskStorm coordinates the operation of the asynchronous memory retrieval and any reasoning tasks running within Nicole, using the same representation for knowledge and the same main system blackboard to communicate.

### **7.1.2. Nicole Development Tools**

Of course, the Nicole core modules do not exist in isolation: their purpose is to enable the construction of experience-based agents that perform real tasks. The most important part of any Nicole-based application is the actual performance task itself, written using the Nicole development tools. The three languages that comprise the Nicole development tools are its representation language, its working memory language, and task control language.

- **CRYSTAL Representation Language:**

Just like the memory system, a performance task uses the CRYSTAL representation system. The programmer of a performance task must represent knowledge used by the task using the CRYSTAL language; that language will in turn automatically incorporate that knowledge into the CRYSTAL experience store.

- **CRYSTAL-BB Working Memory Language:**

Once a content theory for a reasoning task has been defined, a programmer of a Nicole-based application must create a supertask which defines that task's

processing and control. The first half of a supertask definition defines a set of blackboard data structures which the task will use as its scratch data structures.

- **TaskStorm Task Definition Language:**

The other half of a supertask definition is defining the task control hierarchy of a reasoning task using the task definition language provided by TaskStorm. This task hierarchy will execute the functions of the task under the control of the TaskStorm controller using the contents of the main system blackboard. To enable existing artificial intelligence applications to be ported to the Nicole system, TaskStorm supports the execution of arbitrary Lisp code under the direction of the task control system.

### **7.1.3. Nicole-Based Applications**

To develop a Nicole-based application, a programmer must use these languages to construct the equivalent of a new Nicole core module that performs a new task. The distinguishing element that makes a task an experience-based agent is the presence of an integration mechanism which enables the performance task to take advantage of a context-sensitive asynchronous memory, and optionally a feedback mechanism by which the performance task tries to “seed” the memory with additional information about the task context.

Two major Nicole-based applications have been constructed that exploit the features of the experience-based agent architecture: Nicole-MPA and Nicole-IRIA:

- **Nicole-MPA: Multi-Plan Adaptor**

MPA is a case-based planning system that extends Hanks & Weld's (1994) SPA system with integration mechanisms that merge multiple plans. Nicole-MPA is discussed in greater detail in Chapter 8.

- **Nicole-IRIA: Information Research Intelligent Assistant**

IRIA is an intelligent information management system which finds information based on the user's current browsing context and integrates it into ongoing searches for information based on the user's search criteria and explicit user selection. Nicole-IRIA is discussed in greater detail in Chapter 9.

In addition to these Nicole-based applications, Nicole core modules have been used in the construction of several AI systems, most notably the ISAAC system (Moorman 1997). ISAAC is a story understanding system which reads real science fiction stories at a measured comprehension level equivalent to human readers. ISAAC uses the CRYSTAL knowledge representation module to store all of its knowledge except low-level sentence processing information, and uses the MOORE memory module as its primary retrieval system. ISAAC's rich story representation and processing mechanisms placed strong demands upon — and provided a good test of — the generality of the CRYSTAL and MOORE modules.

```

(define-node 'ptrans
  physical.action
  (:agent nil)
  (:from nil)
  (:to nil)
  (:instrument nil)
  (:preconditions nil)
  (:results nil)
)

(define-node 'vulcanian
  (list person
    lifeform
  )
  (:lifespan 220)
  (:sex nil)
  (:homeworld vulcan)
)

(define-relationship
  'rank-relation
  :rank :holds-rank
  (list social-relation)
)

(define-node 'isaac
  meta-ai-construct
  (:name 'isaac)
  (:purpose 'reading)
  (:age 2)
  (:lifeform 'computer)
  (:occupation 'researcher)
)

(define-node 'spock
  (list vulcanian
    starfleet-officer
  )
  (:age 150)
  (:sex male)
  (:rank Ambassador)
)

(define-relationship
  'age-relation
  :age :age-of
  (list physical-relation)
)

```

**Figure 7.2 Sample CRYSTAL Frame Definitions**

#### 7.1.4. Nicole Support Tools

Finally, Nicole provides a variety of service and support systems to ease the task of constructing, debugging, running and collecting data from AI systems constructed with the core architecture. These components include an interactive command line, a scripting language and automatic execution system, a scripted demonstration system, and a variety of utilities for tracing and instrumenting running Nicole code and collecting experimental data:

- **Nicole Command Line:**

The Nicole system includes an interactive command line or “Listener” that enables programmers to step through cycles of execution of the architecture

(where a “cycle” is defined as the atomic execution of a reasoning step in TaskStorm coupled with a memory retrieval steps in MOORE). The command line enables programmers to execute various commands to affect Nicole’s behavior.

- **Nicole Command System:**

To facilitate the execution of tasks, the Nicole system also provides a scripting language and automatic command execution system that can simulate a programmer’s interaction at the Nicole command line, enabling complex tasks to be performed automatically.

- **Nicole Scenario/Demo System:**

Nicole’s task control module can also execute in fully autonomous mode without programmer oversight. To guide the system, Nicole supports the definition of “scenario files” which initialize the Nicole system into a particular configuration and then cause the execution of scripts through the Nicole command system. These scenarios can also be run as self-contained demonstrations.

- **Nicole Trace System:**

The Nicole system also provides a variety of utilities for tracing and instrumenting running Nicole code and collecting experimental data for further analysis.

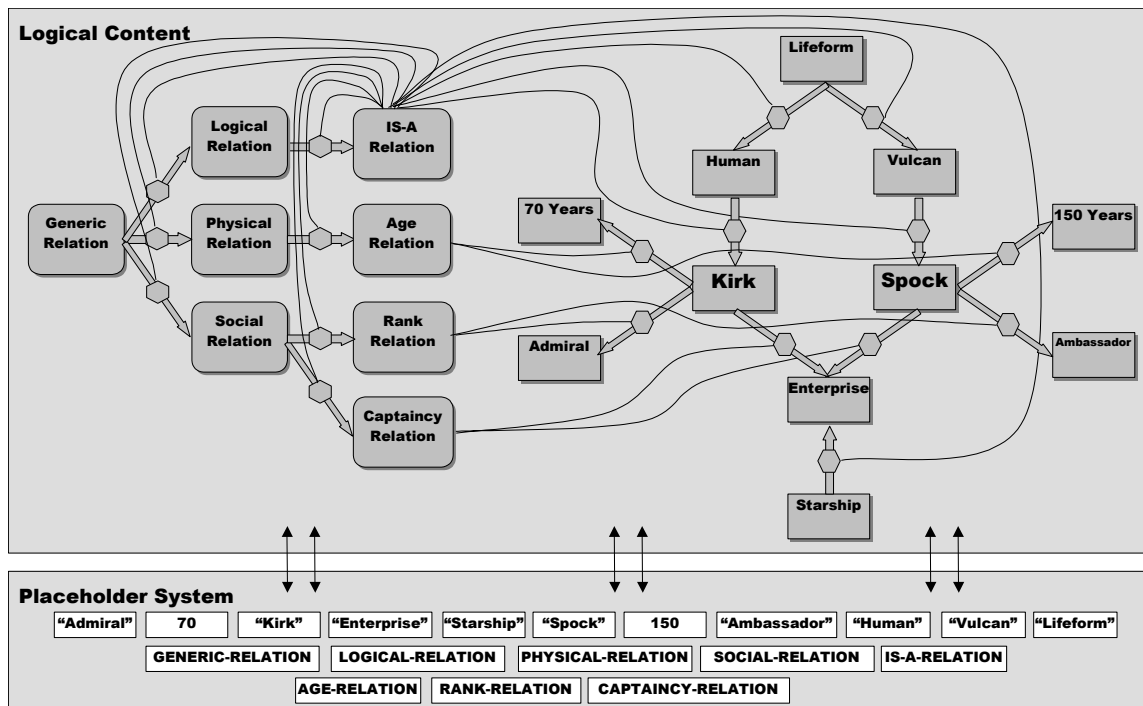
We will not discuss these support systems in any great length. Instead, we will now turn to the theoretically important parts of Nicole: its knowledge representation, memory

retrieval algorithms, task control and communication systems, and samples of reasoning tasks constructed in this architecture.

## 7.2. Knowledge Representation in Nicole

The central knowledge representation of the Nicole system is the CRYSTAL language. CRYSTAL is an extension of the frame-based KR representation/programming language (Guise 1988, 1989, 1992, Myers 1990) which organizes all knowledge in the system into a bidirectional semantic network with reified relations and grounded concepts. CRYSTAL's logical structure and overall expressive power is similar to that of KL-ONE (Brachman 1985), KODIAK (Wilensky 1986), and AQUA (Ram 1989).

The fundamental unit of knowledge in CRYSTAL is the frame, or *node* in CRYSTAL parlance; frame slots correspond to connections between nodes, called *links*. Figure 7.2 shows some typical frame definitions in CRYSTAL. Like KR, its parent programming language, CRYSTAL allows programmers to store low-level data like numbers and strings in frame slots, as well as to create arbitrary nodes and links on the fly. CRYSTAL is distinguished from a programming language, however, by its “knowledge maintenance” policy, which ensures that:



**Figure 7.3 Graphical Depiction of a CRYSTAL Network**

- all perceptual- or program-level data referenced by the knowledge base is automatically “encapsulated” in *placeholder nodes*, allowing primitive data types to automatically participate in knowledge-level relationships.
- new classes of slots on frames are automatically “reified” into *relationship nodes* as they are defined, allowing arbitrary higher-order statements to be made about relationships that exist between nodes. Relations can be both automatically and manually defined; the last two node definitions in Figure 7.2 show the special syntax used to explicitly define nodes as relationships.



- each connection between nodes in the knowledge base is encapsulated into a *relationship instance object*, or RIO, which maintains a three-way link between the source node, target node and the relation node that the link instantiates.

The result of this knowledge maintenance policy is that all nodes in the system are part of an interconnected network grounded in low-level program concepts, and that all “knowledge” in the network can be represented by relation tuples, similar to the “sea of assertions” in CYC (Lenat & Guha 1990, CYCORP 2000). Figure 7.3 shows a graphical depiction of this model: a set of concepts about Kirk and Spock are connected by links that instantiate relationships which are themselves concepts that can be reasoned about; this logical content can be accessed through extra-logical information like numbers, strings and Lisp symbols through the placeholder system. More importantly, every item mentioned in a node definition such as those seen in Figure 7.2 is automatically entered into the placeholder system and assigned its own node at the knowledge level. Moreover, every relationship mentioned as result of node definitions is similarly defined as a full-fledged relationship whether it existed previously or not.

This knowledge maintenance policy means that CRYSTAL is more than a language: use of the CRYSTAL representation automatically results in the creation of a self-maintaining knowledge base. This knowledge base takes the form of a grounded, interconnected network containing all the knowledge of each task using the representation — a unified knowledge base for all agent tasks, or more succinctly, an experience store. As reasoning tasks continue to create knowledge using the CRYSTAL

Memory Retrieval Request #1701	
Specs:	(:is-a Starship)
Cues:	(Kirk CaptainOf)
Importance:	1.0
Mode:	Better-Than
Candidates:	(Enterprise Excelsior ...)
Successes:	(Enterprise BirdOfPrey)
Rejects:	(Farragut Excelsior Spock)
...	

**Figure 7.4 Typical Content of a Memory Retrieval Request**

representation, CRYSTAL's knowledge maintenance policy ensures that this unified knowledge base for all agent tasks continues to satisfy the key properties of an experience store: it is grounded, bidirectional, contains reified relations, and encompasses all agent knowledge. When a new piece of knowledge is added, it is automatically connected with the existing knowledge base through the relationships it participates in, the data it refers to, and the automatically generated two-way relations which connect them.

Therefore, the CRYSTAL module *is* the experience store for the Nicole system, and, by extension, for all Nicole-based applications constructed using the Nicole system. How this experience store forms the basis of memory retrieval in the Nicole system is the topic to which we now turn.

## **7.3. Memory Retrieval in Nicole**

The MOORE (Memory Organization and Optimized Retrieval Engine) module forms the Nicole system's context-sensitive asynchronous memory. MOORE extends the CRYSTAL knowledge representation by adding activation levels, connection strengths and retrieval histories to nodes and links, and then layers on top of that a memory retrieval API, a matching language, and a swappable memory search algorithm.

### **7.3.1. Memory Retrieval Requests**

The fundamental data structures in the MOORE module are memory retrieval requests (Figure 7.4). The MOORE module is essentially a system which creates memory retrieval requests on behalf of reasoning tasks when reasoning tasks “post” needs for information (Figure 7.5), which maintains retrieval requests in a queue while it attempts to satisfy them, and which alerts reasoning tasks when a change occurs in a request they posted.

MOORE's memory retrieval request data structures are direct instantiations of the memory retrieval requests specified by the context-sensitive asynchronous memory approach. Each memory retrieval request contains lists of specifications and cues, an importance value, a current candidate buffer, lists of successful (accepted) and rejected candidate items, and a history of retrieved candidates.

```

(memory:post-request
  ;; Matching Specifications
  `(((is-a          ) ,mpa::problem-statement-object      )
    (:initial-predicates) ,problem                          :gonzo-match)
    (:goal-predicates  ) ,problem                          :gonzo-match)
    (:initial-objects  ) ,problem                          :gonzo-match)
    (:goal-objects     ) ,problem                          :gonzo-match)
    (:solution         ) nil                               :any          )
  )
  ;; Search Cues
  `(,problem
    ,initial-predicates-relation
    ,goal-predicates-relation
    ,initial-objects-relation
    ,goal-objects-relation
    ,solution-relation
  )
)

```

**Figure 7.5 Posting a Memory Retrieval Request**

Retrieval requests also have *matching mode* information which specifies whether the request should return exact matches, approximate matches above a certain threshold, or the best match available. An exact match corresponds to a search with (at least some) precise query terms, such as a search for relevant cases. An approximate match corresponds to a search in which any item in memory of sufficient relatedness to the specifications might be useful, such as an analogical matching process or an understanding process. A best match search is equivalent to a “reminding” task, probing memory for any information related to the current context.

Retrieval requests move through a series of processing states. When newly created, a memory retrieval request is **empty**, seeking a candidate that satisfies its matching mode (exact, threshold or any). Once a candidate has been found, the request goes into an **alert**

state, sending a signal to the reasoning task that a candidate has been found. As part of the alert process, the memory retrieval automatically shifts to a “better than” retrieval mode, only altering the candidate if it finds a better item.

The reasoning task may respond by accepting or rejecting the candidate, thus adding the item to the history of accepts or rejects. In either case, the reasoning task can specify that the memory retrieval narrow its search to seek **better** items than its previous candidates, to keep searching for **additional** items above threshold, or to broaden its search to **any** items.

### 7.3.2. Memory API in MOORE

MOORE supports all of the core memory retrieval operations proposed for context-sensitive asynchronous memories except *post-storage*; storage issues were not addressed extensively in any Nicole-based application and it was decided to defer implementation of that component until a later edition.

Memory retrieval requests are created by MOORE on the main system blackboard in response to a “posted” request for information from a reasoning task (Figure 7.5).<sup>17</sup> The

---

<sup>17</sup> “Gonzo-match” is one of Nicole-MOORE’s matching algorithms (Figure 7.7). It judges the similarity of two nodes in a semantic network by traversing all the links out from each node up to some fixed depth, collecting all the concepts found within that distance for each node, and then comparing the number of shared concepts in each node’s set.

<b>Operator</b>	<b>Signature</b>	<b>Description</b>
<b>exact</b>	<b>item</b>	Looks for this exact item on this slot filler
<b>any</b>	<b>NIL</b>	Checks for the presence of any filler on this slot
<b>member</b>	<b>list</b>	Returns 1.0 if the filler matches one or more of the items
<b>inherit</b>	<b>item</b>	Returns 1.0 if the filler is-a-p the match item
<b>fuzzymatch</b>	<b>item or list</b>	Returns 1.0 if the item(s) match any of the fillers of the slot
<b>partial match</b>	<b>item</b>	Compares two items as depth N trees, based on a recursive similarity measure that weights according to fanout
<b>recursive match</b>	<b>item</b>	Compares two items as depth N trees, based on a recursive similarity measure that weights the roots more heavily
<b>gonzo-match</b>	<b>item</b>	Compares two items based on number of shared concepts up to depth N of tree

**Figure 7.7 Matching Language of MOORE**

*post-request* operation creates a request, adds it to the main system blackboard, and generates activation from the request's cues and specifications in the form of quanta of perturbation in the spreading activation network.

At any time, the memory retrieval request may be updated, changing its cues, specifications, importance or retrieval mode. As part of an update, new quanta of perturbation are spread into the network; this process of spreading activation from a request is known as a "refresh."

Retrieval requests must be continually refreshed to continue to affect the spreading activation system. This process can be done either deliberately by the reasoning task or automatically by the memory retrieval system based on the importance of the requests in the request queue. In the implemented system, MOORE does not automatically spread cues, and instead propagates activation only when requests are created, updated or refreshed.

### **7.3.3. Matching in MOORE**

The matching language in MOORE enables retrieval requests to “walk the structure of knowledge” when looking for matching items, traversing arbitrary sequences of nodes and links in the knowledge base to test for the presence (or absence) of values. This specification language accepts a list of slots (names for particular directions on relationships) and recursively follows them from object to object, seeking the ultimate filler of the final role. The matching language is fairly broad: conditions can test for the existence of any value, the presence of a particular value among a set, an exact matching value, and a variety of partial and fuzzy matches that test the similarity of a value to an exemplar (Figure 7.7). Conditions themselves can be composed using ANDs, Ors and NOTs, and the top-level match can be a strict exact match, a threshold match, or a “best match” based on a weighted similarity metric. This general language allows the same retrieval engine to serve the retrieval needs of a wide range of tasks without task-specific modifications.

```
(:ZOG-12201
  Excelsior                ;; Residence node
  0.0987                   ;; Current zorch
  (Kirk Enterprise Starship) ;; History. Optional.
)
```

**Figure 7.8 Typical Content of a Zog (Activation Record)**

### **7.3.4. Implementing CDSA in MOORE**

While MOORE supports several different retrieval engines for experimental purposes, its most important engine is the “modified-ruminate” engine, which implements a fully parameterizable context-directed spreading activation algorithm. MOORE uses an array of data structures called *zogs* to record activation propagating within the system, each recording a certain amount of propagating activation, perturbation or “zorch” (Hendler 1989, Domeshek 1992) propagating on a specific node in the network. During the course of development of Nicole a variety of zog implementations are tested; the content of a typical implementation is shown in Figure 7.8. Each zog maintains at a minimum its current node and activation strength; the current implementation also maintains a history of the nodes a zog has visited to prevent activation from spreading back to source nodes, though informal tests of alternative implementations without a history appear to show that the use of histories has little or no effect on the cost of spreading activation or the quality of retrieval.



```

(defparameter *damping-cdsa-table*
  (make-parameter-table
    :damping-cdsa
    '(
      ;; This pair of parameters determines the total number
      ;; of nodes activation can theoretically spread to.
      (*ACTIVATION-THRESHHOLD* . 0.01 )
      (*ZORCH-ATTENUATION* . 0.9 )

      ;; This set of parameters determines spreading activation properties.
      (*BASELINE-PROPAGATION* . 0.8 ) ; Base propagation of zorch
      (*GATED-PROPAGATION* . 0.2 ) ; Additional "gated" zorch
      (*RELATION-ATTENUATION* . 0.1 ) ; Zorch leakage to relations
      (*CONTEXT-THRESHHOLD* . 0.01 ) ; Is context damping turned on?
      (*NON-CONTEXT-DAMPING* . 0.9 ) ; Zorch damping outside context

      ;; This set of parameters determines decay properties.
      (*DECAY-ATTENUATION* . 0.75 )
      (*IRRELEVANCE-DECAY* . 0.5 )

      ;; This pair of parameters determines the relative strength of
      ;; knowledge and association links.
      (*KNOWLEDGE-STRENGTH* . 1.0 )
      (*ASSOCIATION-STRENGTH* . 5.0 )
      (*LEARNING-ACTIVATION* )

      ;; These parameters determine how propagation works
      (*PROPAGATION-LIMIT* . 1000 ) ; How many get propagated
      (*DEFAULT-BASE-ACTIVATION* . 1.0 )
      (*ACTIVE-THRESHHOLD* . 0.01 )

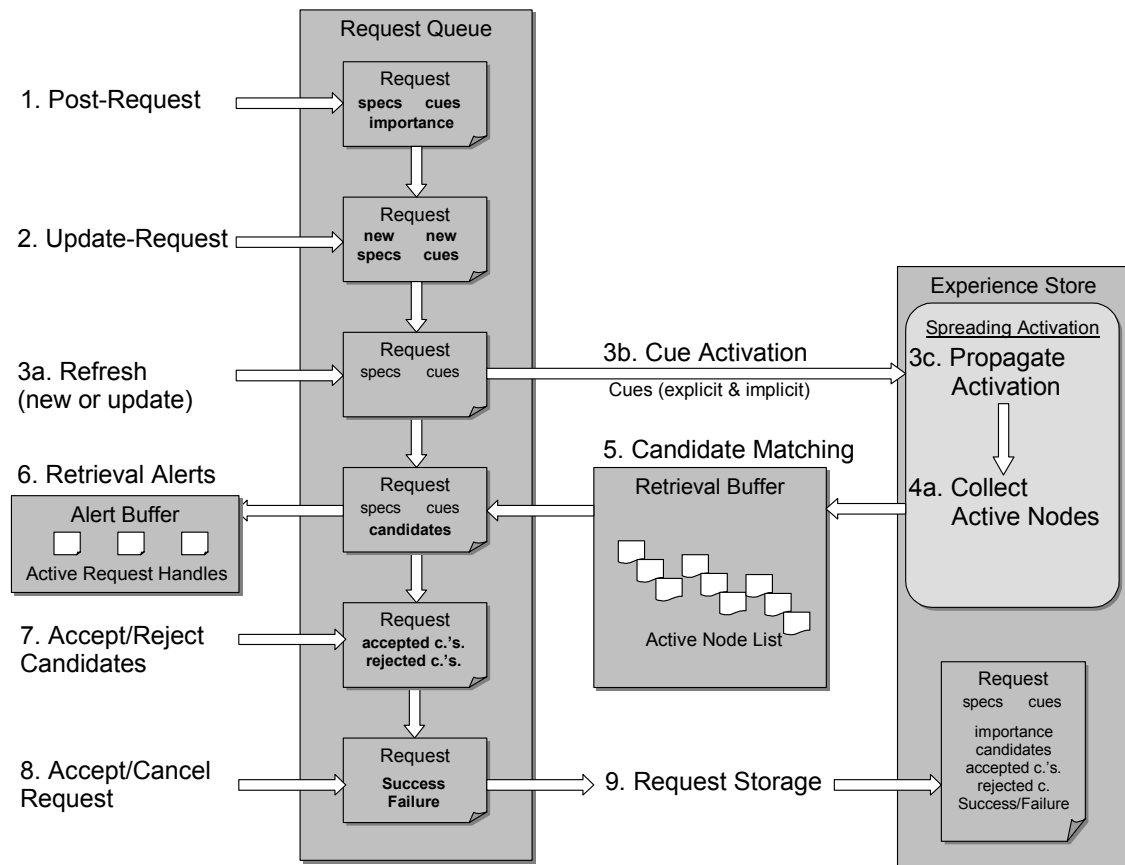
      ;; These parameters determine how retrieval itself works
      (*RETRIEVAL-THRESHHOLD* ) ; Excludes candidates if set
      (*RETRIEVAL-LIMIT* . 20 ) ; How many get matched

      ;; These parameters determine how matching works
      (*CANDIDATE-THRESHHOLD* . 0.01 ) ; Threshold of matchy goodness
      (*CANDIDATE-POLICY* . :STRICT ) ; How we replace candidates
      (*INHERIT-SPEC-MATCH* . :INHERIT) ; Do we check inheritance?
      (*INHERIT-CUE-MATCH* . :LOCAL ) ; Do we check inheritance?
    )
  )
  "A context-sensitive parameterization with non-context damping active."
)

```

**Figure 7.9 MOORE Memory Parameter Table**

Zogs propagate along links between nodes, leaving activation traces on an *active list* of nodes that represents the memory system's current context. The activation on the active list determines the weighting of the connections along which zogs propagate, according to the current parameters of the CDSA equation (Figure 7.9).



**Figure 7.10 The Life History of a Retrieval in Nicole.**

MOORE employs a comprehensive cost control policy; by setting parameters, it is possible to control how many zogs are initially generated from a query, how many are allowed to propagate, how quickly they decay, and so on. MOORE furthermore uses a variety of techniques to make the computation of the zoglist as efficient as possible in the Lisp environment, avoiding unnecessary consing and garbage collection by reusing structures when possible.

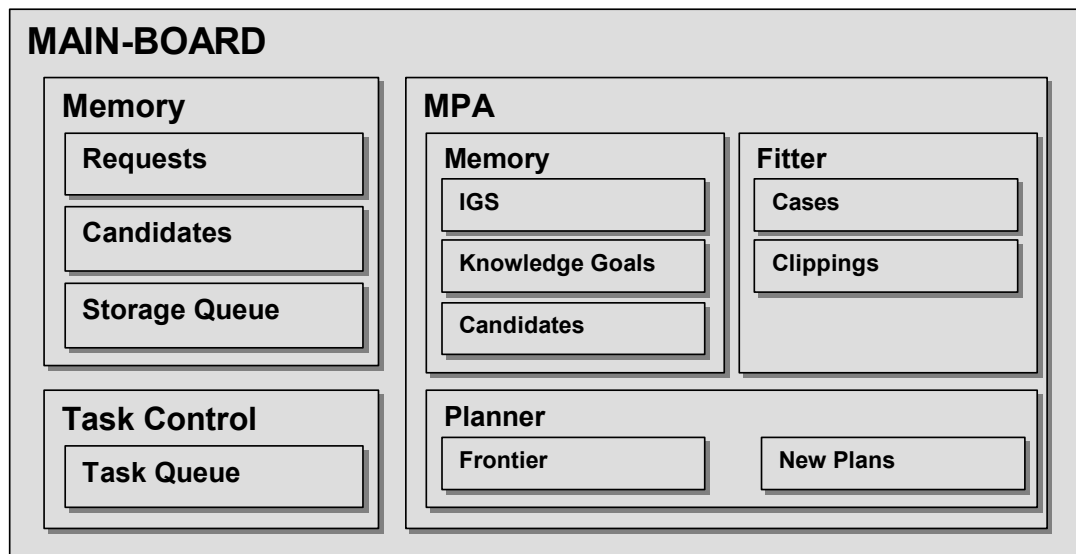
### **The Memory Retrieval Request Processing Algorithm**

- Step 1.** A retrieval begins when a reasoning module calls **post-request**, which adds a new **request-node** to the **request buffer** of the memory blackboard time
- Step 2.** A request-node may be updated with a **update-request** call at any time.
- Step 3.** Each time a request is posted or updated (or when activity occurs in other system blackboards) **activation** spreads to nodes in long term memory (the **experience store**).
- Step 4.** On every retrieval cycle, a limited number of **active nodes** (symbolized by dark circles) are retrieved to the **retrieval buffer** for consideration.
- Step 5.** Each pending request in the request buffer is **matched** against the candidates in the retrieval buffer.
- Step 6.** Successful matches are posted to the **candidate buffer** to be copied to the requesting blackboard or process.
- Step 7.** A retrieval candidate can be accepted or rejected by the **accept-candidate** and **reject-candidate** calls, which update the state of the request to allow it to more accurately select future matches.
- Step 8.** The reasoning module may at any time decide to terminate processing of a request by accepting it through an **accept-request** call or rejecting it with a **cancel-request** call.
- Step 9.** Terminated requests, both successful and unsuccessful, are stored in long term memory and used by the **storage module** (not shown) to adjust associative links in long term memory and retrieval parameters in the matching and candidate retrieval systems.

**Figure 7.11 Pseudocode for the Memory Retrieval Request Processing Algorithm**

### **7.3.5. The MOORE Algorithm**

Memory retrieval requests are processed in “cycles” in MOORE. On each cycle, the current set of retrieval requests is processed once. The active list of retrieval requests, called the reified retrieval request queue, the request queue, or simply the request buffer, is used by the retrieval engine to determine what initial activation spreads into the system’s memory, to record candidate matches of active items to request specifications,



**Figure 7.12 Upper Levels of the Nicole Main System Blackboard**

to send alerts to reasoning tasks when candidates are found, and to record acceptances and rejects of candidates by other tasks.

Figures 7.10 and 7.11 illustrate the typical life history of a retrieval request in Nicole.

## 7.4. Working Memory in Nicole

Working memory in the Nicole system is implemented as a set of hierarchical blackboard data structures. These blackboards are specified using an extension of CRYSTAL called CRYSTAL-BB.

```

The MEMORY Plane of MPA:
  KNOWLEDGE-GOALS:
    #k<KG.216>
  IGS:
    empty
  INTERMEDIATE-PLANS:
    empty
  INTERMEDIATE-PROBLEMS:
    empty
  CUMULATIVE-KG:
    #k<KG.224>
  ACCUMULATED-PROBLEMS:
    #k<SKELETON-PLAN.190>
  RETRIEVALS:
    empty
  CASES:
    #k<MPA::AT-PERSON-LOCAL>
  CLIPPINGS:
    #k<MPA::AT-LUGGAGE-VIA-PUT-DOWN>
    #k<MPA::AT-PERSON-DOMESTIC>
The FITTER Plane of MPA:
  SOURCES:
    empty
  CASES:
    #k<MPA::AT-PERSON-LOCAL>
  FITTED:
    empty
  CANDIDATES:
    #k<CLIPPING.237>
  CYCLES:
    #k<K::1-PLACEHOLDER.3>
  COMPLETED:
    empty
The CANDIDATES Plane of MPA:
  #k<SKELETON-PLAN.190>
  #k<MPA::PX2-R-APD>
  #k<REQUEST-RECORD.210>
  #k<REQUEST-RECORD.219>
  #k<KG.216>
  #k<KG.224>
  #k<SOLVED-PLAN.63>
  #k<SOLVED-PLAN.65>
  #k<SOLVED-PLAN.67>
  #k<SOLVED-PLAN.69>
  #k<SOLVED-PLAN.71>
  #k<SOLVED-PLAN.73>
  #k<SOLVED-PLAN.74>
  #k<SOLVED-PLAN.76>
  #k<SOLVED-PLAN.78>
  #k<MPA::AT-PERSON-LOCAL>
  #k<MPA::AT-PERSON-DOMESTIC>
  #k<MPA::AT-LUGGAGE-VIA-PUT-DOWN>
  #k<MPA::AT-OBJECT-VIA-PUT>
  #k<MPA::HIDDEN-VIA-HIDE>

```

**Figure 7.13 A Portion of the Nicole Main System Blackboard**

CRYSTAL-BB consists of a representation format for blackboard data structures written in CRYSTAL and an application programming interface to create, inspect, manipulate and destroy these blackboards. CRYSTAL-BB blackboards contain

CRYSTAL nodes, and support all the typical working memory operations such as adding and deleting items and testing for the presence of matching items in the memory.

When Nicole loads, it automatically initializes a privileged “main system blackboard” (Figures 7.12, 7.13) based on the current set of defined supertasks (Figure 7.14), the data structures which define the top-level tasks in the system and the working memory structures those tasks use. The main system blackboard is the default communications mechanism for all tasks, and the contents of the blackboard determine how the task control system expands and schedules tasks.

One element of Nicole implemented, but not fully tested, is the use of implicit activation spreading from the contents of working memory to automatically guide the context-sensitive memory retrieval process. While Nicole’s blackboard structures support hooks to propagate activation as items are added or removed from the blackboard, this feature has been disabled in Nicole-MPA and Nicole-IRIA to allow greater experimental control over the properties of activation and context.

## **7.5. Task Control in Nicole**

Supertasks determine the top-level tasks operating in the system, but lower-level tasks (and, ultimately, Lisp code) determine what actually happens within each task. The TaskStorm hierarchical task decomposition module determines how supertasks are decomposed, instantiated, and ultimately executed in Nicole-based applications.

```

(define-node 'mpa-supertask supertask-object
  (:taskname      "MPA")
  (:preconditions t)
  (:blackboard
    '(:mpa (:control :input      ; Statement of the Problem
              :active      ; Is the planner running?
              :mode        ; Memory/Planning Integration Strategy
              :output      ; Solved Problem
            )
      (:planner :mode          ; Planning Strategy
              :frontier      ; Planning Frontier
              :focus        ; Current Plan in Frontier
              :candidates    ; Newly Generated Plans
              :solutions     ; Solved Plans
            )
      (:memory  :knowledge-goals ; Initial Planning Knowledge
              :igs          ; Intermediate Goal Statements
              :retrievals   ; Plan Object
            )
      ;; MORE BLACKBOARD PANES ...
    )
  )
  (:task-net
    ;; Activate Main MPA Control Task
    '((:trigger ()) (:once) (:queue-task #!mpa-control)
      )
    )
  )
)

```

**Figure 7.14 A Supertask Definition in Nicole**

Each supertask defines not only a blackboard data structure but also a top level task which executes using that blackboard as a data structure (Figure 7.14). Just as MOORE processes retrieval requests in cycles, so does TaskStorm process tasks in cycles. On each task execution cycle, compound tasks are recursively decomposed and reactive atomic tasks are permitted to execute once. In its typical configuration, the Nicole system includes a top-level task for MOORE, which executes one memory retrieval cycle during every task cycle. Together, execution of a task cycle and a retrieval cycle constitute one “cycle of execution” for the Nicole system.

Tasks are hierarchical data structures (Figure 7.14) which define ordering and control of low-level actions within the Nicole system. Tasks are defined by a variety of features:

- **type:**

Determines whether a task is compound, atomic, or one of the legacy task types supported in Taskstorm.

- **parameters:**

Parameters are task variables which must be bound against the current state of working memory for the task to execute. Parameters may appear later in preconditions, method selectors, and so forth. If a task has been queued and its parameters cannot be bound, a queueing impasse occurs.

- **precondition:**

Preconditions are conditional tests written in the task language which must be satisfied for a task to execute. If a task has been queued and its preconditions cannot be satisfied, a queueing impasse occurs.

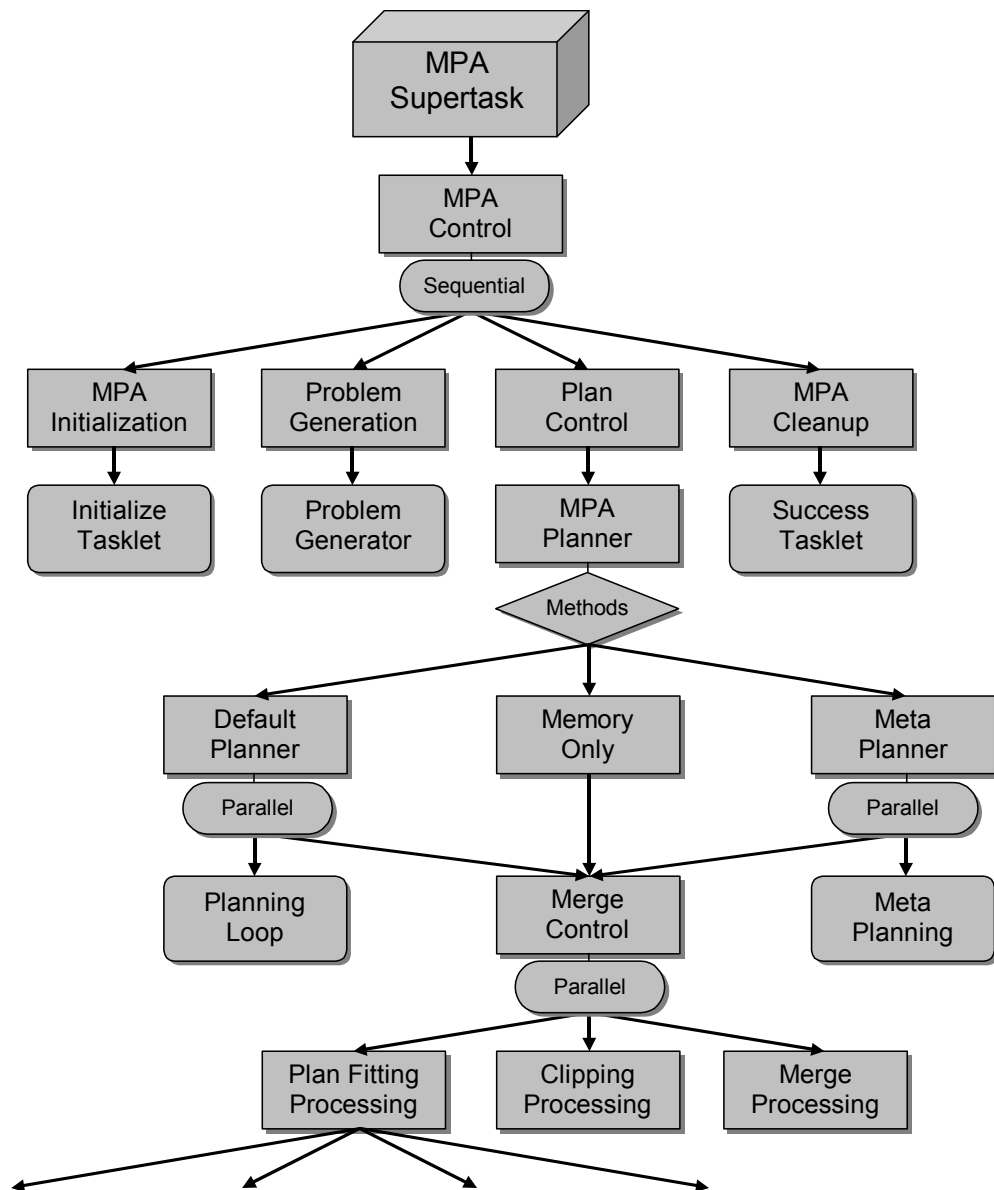
- **body:**

Atomic tasks have a task body which can be directly executed.

- **method selector:**

Compound tasks have a method selector function, written as a condition-action test in the task language, which selects one or more method expansion definitions for expansion.





**Figure 7.15 Portion of Nicole-MPA Planning Supertask Hierarchy**

- **method expansion definitions:**

Compound task methods have a specification of a set of subtasks which this task can be expanded into.

- **subtask execution instructions:**

A set of parameters, including *scheduling-type*, *synchronization-type*, *looping-type*, *completion-type*, which determine how the expanded method's tasks will be executed.

- **termination condition:**

A condition which signals the normal termination of this task.

- **abort condition:**

A condition which signals the abnormal (forced) termination of this task.

As noted, a task may be compound, in which case the method expands into a set of subtasks and instructions on how those subtasks are to be executed (Figure 7.16), or atomic, in which case its single method “expands” into a chunk of Lisp code (Figure 7.17). The ultimate result of a task expansion is a tree of expanded tasks terminating in atomic tasks at its leaves (Figure 7.15).

A task's parameter bindings, preconditions, method selection criteria and termination conditions are dynamically computed during task processing using the contents of the main system blackboard. These task parameters are written in a condition language which “walks” the structure of the main system blackboard in the same way that MOORE's matching language walks the structure of the knowledge base.

```

(define-template mpa-control ((mode :get :mpa :control :active))
  :name "MPA-Master-Control-Task"
  :type :compound
  :scheduling-type :sequential
  :synchronization-type :barrier
  :looping-type :repeat
  :completion-type :forced
  :precondition (:and (:or (:get :mpa :control :active)
                           (:get :mpa :control :output))
                      )
  :termination (:never)
  :action (!mpa-initialization
          !mpa-problem-generator
          !mpa-planner
          !mpa-cleanup)
  )

```

**Figure 7.16 A Compound Task Definition (Top-Level Control in Nicole-MPA)**

TaskStorm controls the execution of the various top-level tasks in Nicole by maintaining a task queue which it executes in repeated cycles. On each execution cycle, TaskStorm processes the current task queue, decomposing tasks into expanded task trees and executing any atomic tasks whose preconditions are satisfied.

When a task enters the queue for the first time, TaskStorm attempts to bind the task's parameters. If the binding conditions cannot be satisfied the task encounters a *queuing impasse*, remaining in suspended execution until the condition satisfies or the task's parent terminates.

When the task's parameters have been successfully bound, it is "queued" for execution, and its preconditions are tested on every task cycle. If a task's preconditions are not satisfied, the task remains inactive on the task queue, awaiting the satisfaction of

its preconditions, the satisfaction of its termination conditions, or the termination of its parent task.

Once a task's preconditions are satisfied, the task executes until its termination conditions are met. For an atomic task, this simply means executing the chunk of Lisp code it encapsulates with the appropriate parameter bindings. For a compound task, however, execution means decomposition: a method must be selected to “expand” the task into a lower-level set of task primitives.

Method selection criteria are written in the same condition language used for the preconditions and parameters of a task (unless the task has a single method, in which case the method selection criteria may be omitted). A method is simply a list of subtasks, along with instructions that determine how the tasks are synchronized (whether the tasks execute serially or in parallel) and how tasks are chosen (whether all subtasks execute or a single subtask must be selected). If a subtask must be chosen, the method must specify the choice criteria that determines how the appropriate subtask is selected. If not enough information is available to make this decision, the task encounters a choice impasse and must wait until this information is available.

Ultimately most tasks are successfully decomposed, resulting in the addition of one or more child tasks to the task tree being processed. The parent now lies “dormant”, as each of its children on the queue is decomposed and executed just as it was. The parent

```

(define-template splice-everything-carefully
  ;; Input Parameters
  ((mode :pop :mpa :planner :mode)
   (plan :pop :mpa :planner :focus)
   (cases :get :mpa :clipper :sources)
   new-plans)

  ;; Type, Permanence and Termination
  :type :atomic
  :precondition (:and (:exists plan cases)
                      (:member mode #!step-mode #!splice-mode)
                      )
  :termination (:always)

  ;; Execution Body
  :body
  (
    ;; Examine the focus plan:
    ;; Is this a good opportunity to try to splice a case in?
    (when (and plan
                (expandable-p plan)
                (simple-opportunity-for-plan-pasting-p plan)
            )
      ;; If so, expand the plan via merging
      (setf new-plans
        (loop for case in cases collect
              (mpa-merge-plan-and-case plan case)
            )
      )
    )
  )

  ;; Output Effects
  :effects
  ((:on new-plans :push-list :mpa :planner :novel-plans)
   )
)

```

**Figure 7.17 An Atomic Task Definition (Planning Integration in Nicole-MPA)**

terminates when all its children terminate, when its own termination criteria are met, or when one of its parents' termination criteria are met.

TaskStorm supports the task interleaving demanded by the experience-based agent architecture at the highest level using the supertask mechanism: by default, all active supertasks in the system operate effectively in parallel. In the portable Lisp

implementation, these tasks are interleaved but could just as easily operate in separate threads or on separate processors.

To enable integration mechanisms to respond appropriately to new retrievals as they are generated, TaskStorm also supports task interleaving at the task level; methods may be decomposed into sets of tasks executing in parallel with a variety of synchronization primitives. These primitives allow the user to specify “blocking” synchronization in which all subtasks must run to completion before the parent task is considered complete, “race” synchronization in which the first completed subtask signals completion of the parent, “spawning” synchronization in which subtasks respawn until the parent task dies, or “serial” synchronization in which each subtask must execute in turn. The synchronization language also allows the user to specify whether a task terminates when its subtasks are complete or loops over the subtasks repeatedly.

## **7.6. Sample Reasoning Tasks in Nicole**

Nicole-MPA and Nicole-IRIA illustrate the typical structure of Nicole-based applications.

Nicole-MPA consists of three main modules of code: knowledge representation, integration mechanisms, and task code. Nicole-MPA extends the SPA planner (Hanks & Weld 1996), which has its own knowledge representation for plans; Nicole-MPA provides a “plan tagging” mechanism which makes the contents of a plan visible to the

memory retrieval system. Nicole-MPA also provides plan integration mechanisms which extend SPA to enable it to compose multiple plans. Finally, to enable the integration of plans into the planning process as they are found by the memory, Nicole-MPA provides a planning supertask which encompasses a variant of the SPA plan search algorithm using Nicole's main system blackboard and plan data structures. Figure 7.15 shows Nicole-MPA's planning task hierarchy, and Figure 7.16 shows a leaf task for executing plan integration mechanisms.

Nicole-IRIA, in contrast, consists primarily of knowledge representation and interface processing.<sup>18</sup> The bulk of Nicole-IRIA's code is used to represent Internet and other information resources in the CRYSTAL language. The remainder of Nicole-IRIA is in the interface, translating user searches and clicks into traditional Nicole-MOORE memory retrievals and memory cues. Because Nicole-IRIA's information retrieval process is controlled through the user interface, no information retrieval supertask was necessary; instead, Nicole-IRIA's primary duty.

Nicole-MPA and Nicole-IRIA will be discussed in more detail in Chapters 8 and 9.

---

<sup>18</sup> Nicole-IRIA was developed at Enkia Corporation as part of an Air Force Rome Labs SBIR Phase I grant.

```

(defscenario 'run-merge-without-trace
  :user-tasks '(!mpa::mpa-supertask)
  :autonomous t
  :script
  '(:exec (unless (boundp 'mpa::*experimental-library*)
                  (setf mpa::*experimental-library*
                        (mpa::solve-domain
                          (get-argument :domain mpa::stinger)))))

    (:exec (apply #'initialize-problem-and-mode
                  (get-argument :problem)
                  (get-argument :mode)
                  nil))

    (:exec (setf taskstorm::*taskstorm-countdown*
                  (get-argument :max-cycles 100)))

    (:post-command (when (> taskstorm::*taskstorm-execution-ticks* 100)
                      (taskstorm::taskstorm-quit)))

    (:post-command (when (get-board :mpa :control :output)
                      (taskstorm::taskstorm-quit)))
  )
)

```

**Figure 7.18 Sample Nicole Scenario**

## 7.7. Additional Features

The Nicole system has a variety of additional features designed to make it easier to write, run, test, evaluate and demonstrate Nicole-based applications.

Nicole's Module system, similar to Lisp defsystem, breaks the Nicole system into smaller components which can be loaded separately, enabling the modules and tools of the Nicole system to be loaded on demand. The module system tracks all loaded files, enables the quick re-loading of modified files, and provides an automatic compilation facility.



The Nicole system's native operating environment is the Nicole command line, a Lisp shell which provides the user with the ability to "step" the Nicole one or more cycles of execution, inspect the main system blackboard, get help, and execute Lisp commands. The Nicole system can also operate in a "standalone" mode in which Nicole's task and memory system run autonomously, or in a "slave" mode in which a cycle can be executed by another program.

Nicole also provides a Command facility written in the TaskStorm language which enables users to write "scripts" or "macros" which can be executed by an autonomously running Nicole as if the user was entering commands at the Nicole command line.

Nicole also provides a more extensive Scenario facility, which encapsulates a set of commands along with a full list of Nicole parameters in an external, editable script file (Figure 7.18). A scenario enabling Nicole to be automatically started and stepped to an arbitrary particular state before control either terminates or returns to the user. The Scenario facility is in turn used by the Nicole Demo system, which uses similar, more extensive external script files to step Nicole through a predetermined scenario either automatically or stepped at breakpoints by a single keystroke.

Nicole provides a variety of ways to instrument running programs. A tracing facility enables users to toggle the display of named tracing statements for debugging or display purposes. The Nicole command line can be configured to trace the execution of the memory system and display the results. Finally, Nicole also supports "experimental

harnesses” which call Nicole scenario scripts repeatedly with different sets of parameters, collecting tracing data and outputting the data to named files for later analysis.

This combination of tracing, instrumented code and test harnesses was used to collect the data for the feasibility and case studies of the experience-based agent approach as implemented in the Nicole system; it is to these studies that we now turn.

# CHAPTER VIII.

## CASE STUDY: PLANNING

---

*Experiments with exploiting a context-sensitive asynchronous memory*

### 8.1. Overview

The first major test of the context-sensitive asynchronous memory approach's ability to provide benefit to real tasks was in planning. In this case study, a planning system was constructed on experience-based agent principles and tested against a variety of other approaches. This experience-based planning system, Nicole-MPA, was tested in both rigorous experiments and a series of exploratory evaluations designed to identify where context-sensitive asynchronous memory would provide the greatest benefit.

### 8.2. Goals of the Case Study

This case study was designed to investigate how context-sensitive asynchronous memory can be applied to a reasoning task and to illuminate the benefits that it could provide to those tasks. The specific thesis tested was:

The experience-based agent approach can provide performance improvement to reasoning tasks.
---

As the reader may recall from Chapters 3-5, the experience-based agent approach is simply a framework for constructing reasoning tasks that can exploit the properties of a context-sensitive asynchronous memory. Therefore, this study can be seen as testing several theses: first, whether experience-based agency is a fruitful approach for exploiting a context-sensitive asynchronous memory, second, whether context-sensitive asynchronous memory works at all, and third, under what circumstances context-sensitive asynchronous memory is most effective.

This chapter will review experimental results most pertinent to the first of these questions, deferring discussion of results related to the quality and applicability of context-sensitive asynchronous memory until Chapter 10.

### **8.3. Methodology of the Case Study.**

Testing the thesis that experience-based agency can provide performance improvement to a reasoning task actually requires applying it to a task and evaluating its performance with respect to alternative approaches.

I chose to apply experience-based agency to the planning task, specifically to least-commitment case-based planning. Applied to planning, the experience-based agent approach consists of building a store of planning experience accessed through a context-sensitive asynchronous memory and exploited through integration mechanisms that dynamically combine multiple cases to reduce planning effort.

The performance of the experience-based approach to this task was evaluated by comparison with alternative approaches, such as single-experience adaptive approaches and from-scratch generative planning.

## **8.4. Execution of the Study**

The testbed for these evaluations was the Nicole-MPA system, a least-commitment case-based planner built along experience-based agent principles and implemented in the Nicole architecture. Nicole-MPA is an “integrative” multi-case case-based planner with a planning module that satisfies the context-sensitive asynchronous memory approach’s requirements for memory-reasoning communication, feedback and integration.

To test Nicole-MPA, I constructed several planning domains and case libraries including the Travelworld, Stinger Missile and Abstract-3 domains; I also constructed a series of variants of Nicole-MPA, including generative, single-case, and multi-case modes. I then conducted a series of experiments using Nicole-MPA on these domains, comparing the performance of generative, single-case adaptive and experience-based modes, attempting to illustrate the conditions under which experience-based agency would provide performance improvements.

The study resulted in both failures and successes: ultimately it showed the capacity of the context-sensitive asynchronous memory approach to improve a system’s performance

but along the way demonstrated failings of the original hypotheses on how the approach could be applied.

## **8.5. Target Task: Planning**

This study explored how the experience-based agent approach could be applied to the target task of case-based least-commitment planning. Planning in general is the task of reasoning about action — finding a sequence of actions that achieves a given goal from a given initial state. Least-commitment planning is a particular framework for planning that reasons about partially complete plans. Case-based planning is a framework which attempts to exploit past knowledge to solve new problems.

### **8.5.1. Least Commitment Planning**

This study took a least-commitment approach to the planning task (Weld, 1994). Least-commitment planning departs from classical planning systems by delaying decisions about step orderings and bindings as much as possible to prevent backtracking. Least-commitment planners solve problems by successive *refinement* of a partial plan derived from the initial and goal conditions of the problem. Plans are represented as sets of steps, causal links between steps, variable bindings and ordering constraints. Beginning with a skeletal partial plan based on the initial and goal conditions of the problem, a least-commitment planner attempts to refine the plan by adding steps, links and constraints that eliminate open conditions or resolve threats.

### 8.5.2. Case-Based Planning

One drawback of planning is that solving planning problems can be computationally expensive; depending on the richness of the planning representation and planning problems, the planning task can be NP-complete at best and undecidable at worst. However, the solutions to a planning problem are determined by the environmental conditions in which action can be taken. This combination — hard problems and environmental cues — this makes planning a good candidate for knowledge re-use.

One method for knowledge reuse for the planning task is case-based reasoning, or more specifically *case-based planning* (Hammond 1989, Veloso 1994). Case-based planning involves treating each problem that the planner encounters as a case which encompasses the problem, the solution or steps taken to find a solution, and the outcome obtained by using or applying the solution derived. A case is essentially an experience that teaches a lesson: for *this* problem in *these* conditions the following solution is valid; for *that* problem in *those* conditions the following pitfalls may arise. A case-based planner solves new problems by retrieving past experiences and applying the lessons learned: for example, by adapting the previous solution to the current situation.

### 8.5.3. Prior Work in Least-Commitment Case Based Planning

The Systematic Plan Adaptor algorithm (Hanks & Weld, 1995) is a least-commitment algorithm for case-based planning. SPA is based on four key ideas: treat adaptation as a *refinement* process (based on the generative least-commitment planner SNLP;

McAllester & Rosenblitt, 1991), annotate partial plans with *reasons* for decisions, add a *retraction mechanism* to remove decisions, and add a *fitting mechanism* to fit previous plans to current situations. Reasons support the fitting and retraction mechanisms by allowing SPA to determine the dependencies between steps. Once a plan has been retrieved, it may contain steps and constraints that are extraneous or inconsistent with the new situations; during fitting, reasons allow SPA to identify extraneous steps and remove them; during adaptation, reasons allow SPA to isolate steps which are candidates for retraction.

One limitation of SPA is that it is a single-plan adaptor; even if a new problem could be solved by merging several plans, SPA must choose only one and adapt it to fit. This limitation arises from SPA's attempt to maintain *systematicity* and *completeness*. A systematic planner never repeats its work by considering a partial plan more than once; a complete planner always finds a solution if it exists. SPA ensures these properties (in part) by choosing only one refinement at a time, by never retracting any refinement made during adaptation (to avoid reconsidering plans), and by adding all possible versions of any refinement it chooses (to avoid missing a solution). Fuller discussions of completeness and systematicity in SPA can be found in (Hanks & Weld 1995, Francis & Ram 1995).



## **8.6. The Problem: Solving Hard Problems with Past Experience**

Planning was a good choice for a reasoning task for the first case study for a variety of reasons. Planning is a heavily-studied task in artificial intelligence with well-understood properties, and planning (or, more generally, reasoning about action) is required for certain kinds of real-world tasks. Even tasks that do not logically require planning can sometimes be solved by planning techniques. Planning depends on large amounts of world knowledge specific to particular contexts and situations.

Planning furthermore has several properties which make the experience-based agent approach applicable. First, planning problems are computationally difficult to solve — between NP-complete and undecidable based on the planning model — making it a good candidate for reuse of knowledge. Second, planning by its nature elaborates the description of the problem as possible actions and constraints upon action are discovered. Third, the relevance of actions and cases to a planning state is related to the initial and goal conditions of a plan by the plan's domain structure. Fourth, while plans can be large and complex they can also often be broken up into smaller components, making planning a good candidate for combination of multiple past experiences.

Taken together, this suggests that planning would be a good target for the experience based agent approach: it includes hard problems which can be solved through experience, generates contextual information during the reasoning process, contains contextual

structure in the domain, and can potentially benefit from the discovery of additional experiences during the process.

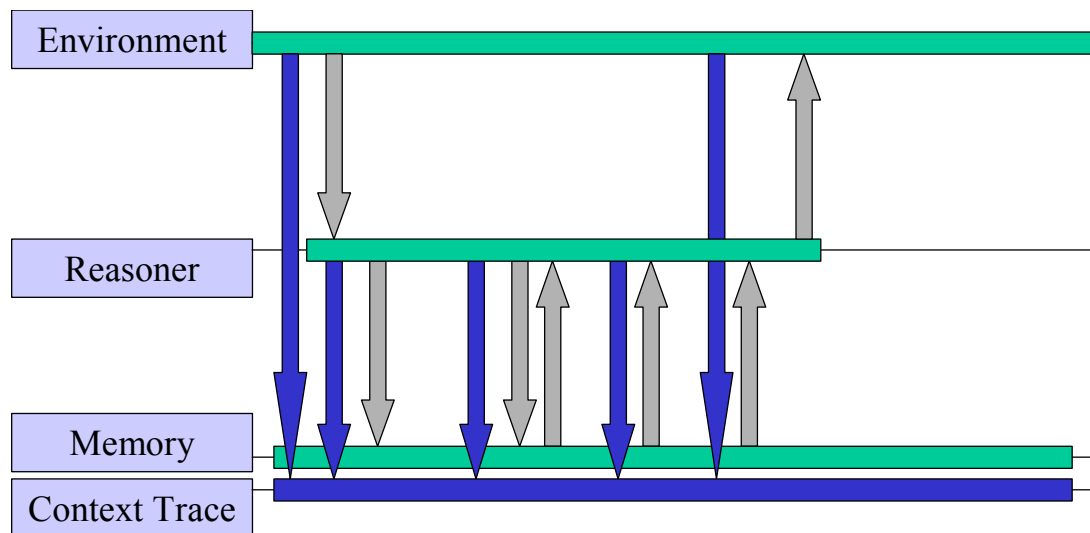
## **8.7. The Solution: Combining Multiple Past Experiences**

The experience-based agent solution to knowledge reuse in case-based planning is essentially multi-case case-based planning using context-sensitive asynchronous retrieval of cases based on feedback from the planning process.

### **8.7.1. Principles Behind an Experience-Based Agent Solution**

Experience-based agency provides a framework for applying context-sensitive asynchronous memory to the problem of reuse of knowledge in context. Context-sensitive asynchronous memory explains how to use context to retrieve experience likely to be useful for reuse. Experience-based agency explains how to take retrieved experience and reuse it by integrating it into the current reasoning state.

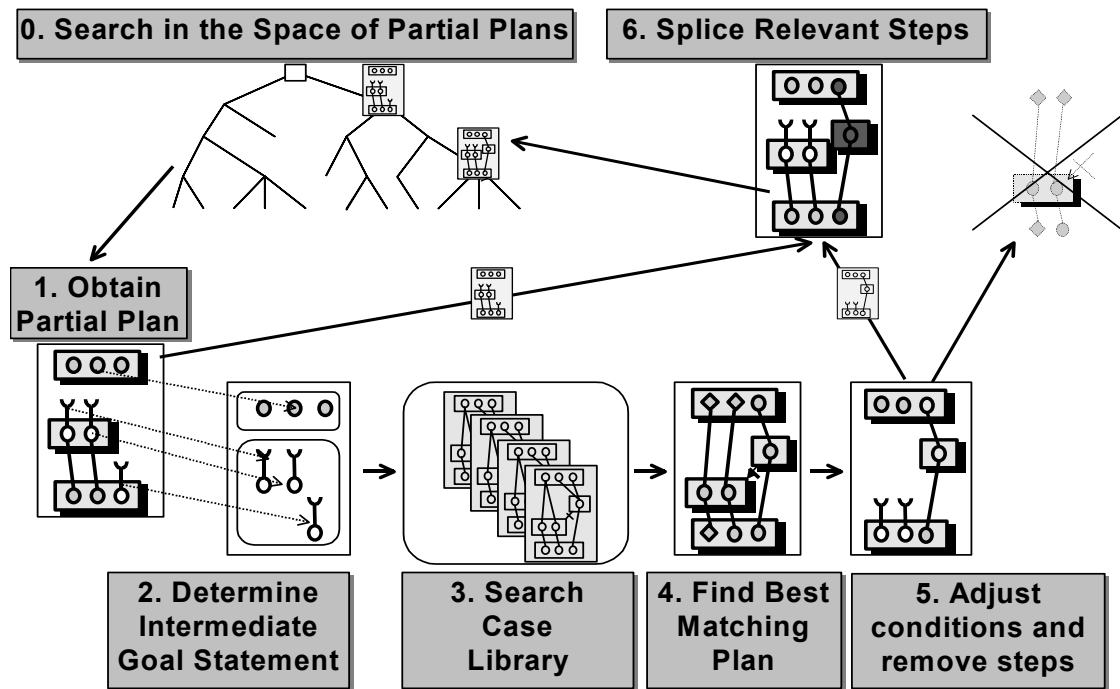
Like case-based reasoning, the experience-based agent approach is an attempt to reuse past knowledge to improve current performance based on relationship between current conditions and past experiences. Experience-based agency can be viewed as both more general than and more specific than case-based reasoning. It is more general in that the context-sensitive asynchronous memory architecture used by an experience-based agent is not limited to the retrieval of cases; it is more specific in that it imposes a particular style of memory retrieval (context-sensitive, approximate, asynchronous) and



**Figure 8.1 The Experience-Based Agent Approach to Reasoning Integration**

particular constraints on how experiences are used (multiple experiences are combined dynamically).

The experience-based agent approach to knowledge reuse is both enabled and driven by the context-sensitive asynchronous memory retrieval process. Where many traditional reasoners use the initial problem to find past experiences before beginning to work on a problem, an EBA system in contrast exploits interrupts from memory as they occur, using integration mechanisms to incorporate multiple past pieces of knowledge dynamically to solve problems that cannot be solved otherwise (Figure 8.1). The goal is to use feedback from the task process to inform the context-sensitive asynchronous memory process, obtaining additional relevant past experiences which may improve the system's ability to solve the problem.



**Figure 8.2. The Multi-Plan Adaptation Cycle**

To exploit this strategy, the reasoning task of an experience-based agent must satisfy a variety of constraints, including not only the constraints of memory (communicating with memory and providing feedback) but also of the reasoning style. A reasoner for an experience-based agent must be able to integrate multiple past experiences dynamically. Applied to case-based planning, this reasoning strategy becomes multi-plan adaptation.

### 8.7.2. Multi-Plan Adaptation

Because it adapts only one plan, SPA can resort to significant amounts of from-scratch planning even when the knowledge needed to complete the plan is present in the case library. A potential way to make more effective use of the planner's past experience is to recognize when a partial plan needs to be extended, select plans to that address the

deficiency, and then extract and merge the relevant parts of the retrieved plan into the original plan.

An approach based on multi-plan adaptation would resolve this problem in SPA by allowing the retrieval of cases at any point during the adaptation process, the dynamic extraction of relevant parts of past cases, and merging of relevant cases to improve system performance. To implement this approach, I developed a multi-plan adaptation algorithm called, appropriately enough, MPA. MPA is based on a model of plan adaptation that extends the SPA framework by adding three key mechanisms:

- a goal deriver, which extracts intermediate goal statements from partial plans
- a plan clipper, which prepares plans for merging using a modified plan fitting mechanism
- a plan splicer, which merges two plans together based on their causal structure

Intermediate goal statements are MPA's knowledge goals; they provide MPA with the ability to merge partial plans at any point of the adaptation process and contribute to its ability to dynamically extract the relevant subparts of retrieved cases. Intermediate goal statements are extracted by the inverse of a representational "trick" that SPA uses to construct a skeletal plan if it can't find a relevant case in its library. The trick is simple: build a skeletal plan by adding dummy initial and final steps whose post- and pre-conditions match the initial and goal conditions of the problem. This technique is also used in SPA's generative predecessor SNLP (McAllester & Rosenblitt, 1991) as well as a

host of other STRIPS-based planning systems. MPA inverts this trick by extracting new goals from the open conditions of a partial plan. As planning proceeds, open conditions in the goal statement are resolved, but new open conditions are posted as new steps are added. MPA constructs an intermediate goal statement by extracting these new open conditions and using them to form the new goal state, and by extracting the initial conditions of the partial plan can be extracted and using them to form the new initial conditions.<sup>19</sup>

Just like the original goal statement, the intermediate goal statement can be used to retrieve and fit a partial plan. However, the result of this process is not a complete fitted plan suitable for adaptation; it is a *plan clipping* that satisfies some or all of the open conditions of the partial plan from which the intermediate goal statement was derived. To take advantage of the plan clipping for adaptation, it must be *spliced* into the original partial plan. Together, plan clipping and splicing form MPA's integration mechanism for incorporating new plans into the current reasoning context (Figure 8.2).

---

<sup>19</sup> Unfortunately, since ordering constraints and binding constraints may be posted to the plan at any time, only the initial conditions can be guaranteed to be valid conditions for the intermediate goal statement. Conditions established by other steps of the plan may be clobbered by the addition of new steps and new ordering constraints. However, it might be possible to develop heuristics that select additional initial conditions that are likely to hold, perhaps in conjunction with more complex retrieval, fitting and splicing algorithms. In general, deciding which parts of a plan can be extracted to form a sensible and effective intermediate goal statement is a difficult and unsolved problem.

### **The Multi-Plan Adaptation Cycle**

**Step 1.** a partial plan is obtained, either directly from a goal statement, from an initial case fitting, or from ongoing adaptation processes.

**Step 2.** An intermediate goal statement is extracted from the plan, consisting of the initial conditions known to be true in the world and the set of preconditions not yet satisfied in the plan.

**Step 3.** The case library is searched for a matching plan in exactly the same way that it is searched for initial case fittings.

**Step 4.** The best matching plan is retrieved and

**Step 5.** is adjusted to have the right set of initial and goal conditions and to remove extraneous plan steps.

**Step 6.** The steps are recursively spliced into the plan, beginning with the links that match to the intermediate goal statement and then moving backwards through the plan along the paths of the causal links.

**Step 0.** Finally, the successfully spliced plans are returned for further adaptation or splicing.

**Figure 8.3 Pseudocode for the Multi-Plan Adaptation Cycle**

Ignoring for a moment issues such as overall control and the role of memory retrieval, the multi-plan adaptation cycle consists of the following steps: getting a partial plan, extracting an intermediate goal statement, searching for relevant cases, selecting the best case, clipping the best case to the intermediate goal statement, splicing the clipping into the case, and continuing processing (Figure 8.3).

MPA's splicing mechanism uses the intermediate goal statement to produce a mapping between the partial plan and the plan clipping, pairing open conditions from the partial plan with satisfied goal conditions from the plan clipping. The plan splicer uses this mapping to perform a guided refinement of the original partial plan, selecting goal conditions from the clipping and using the links and steps that satisfied them as templates

to instantiate similar steps and links in the original plan. As these steps are added, new mappings are established between open conditions in the new steps and satisfied preconditions in the clipping and are added to the queue of mappings that the splicer is processing. Hence, the plan splicer performs a backwards breadth-first search through the causal structure of the plan clipping, using links and steps in the clipping to guide the instantiation of links and steps in the original plan. Figure 8.4 briefly outlines the MPA plan merging algorithm.

### **8.7.3. Controlling Multi-Plan Adaptation**

Merely having the ability to splice plans together does not allow us to take advantage of past experience. We need to decide what experiences to combine and when to combine them. Because the MPA algorithm can potentially be performed at any point during the adaptation process — using an initial skeletal plan derived from the initial and goal statement, using a fitted plan derived from retrieval, or using an adapted plan after some arbitrary amount of retraction and refinement — we have considerable flexibility in deciding what to retrieve, when to retrieve it and when to merge it.

This flexibility — and the pitfalls that go along with it — highlights one of the key issues that distinguishes the experience-based agent approach from context-sensitive asynchronous memory proper. Context-sensitive asynchronous memory focuses on when to retrieve experiences; experience-based agency focuses on how to integrate and compose experiences together into a reasoning state. Ultimately, of course, the issue of



#### **The MPA Plan Merging Algorithm**

**Input:** A partial plan P, and a case library C.

**Output:** A new partial plan P'.

procedure **MPA** (P, C):

**Step 1.** P' ← **Copy-Plan**(P)

**Step 2.** igs ← **GetIntermediateGoalStatement**(P')

**Step 3.** plan ← **RetrieveBestPlan** (C, igs)

**Step 4.** {clipping, mapping} ← **FitPlan** (plan, igs)

**Step 5.** for cgp in mapping do

**Step 6.** if **Producer-Exists** (oc-gl-pair, P')

**Step 7.** then **Splice-Link** (oc-gl-pair, P', clipping)

**Step 8.** else **Splice-Step** (oc-gl-pair, P', clipping)

**Step 9.** **AddNewOpenCond-GoalPairs**(mapping, P')

**Step 10.** return P'

**Figure 8.4. The MPA Plan Merging Algorithm**

when and how to compose must be driven by the specifics of the agent and the tasks at hand: a complete system built on experience based agent principles must make explicit choices about its strategy for integrating context-sensitive asynchronous retrievals and experience integration.

There is a spectrum of possible control regimes for multi plan adaptation based on the commitments a control algorithm makes about when to retrieve and when to adapt. Three such algorithms are Systematic MPA, Interactive MPA, and Extreme MPA.

On one end of the spectrum, *Systematic MPA* preserves SPA's property of systematicity by splicing all retrieved cases before (generative) adaptation begins. On the

other, *Extreme MPA* never performs generative adaptation and instead uses a set of *pivotal cases* (Smyth & Keane, 1995) to guarantee completeness.

Both Systematic and Extreme MPA make extreme commitments: either integrate all knowledge before generative adaptation begins, or never generatively adapt and rely solely on past experience. An alternative approach is to allow plan splicing at any point during adaptation. In the middle stands *Interactive MPA*, a control regime that can potentially attempt a retrieval at any time, either with the initial skeletal plan or with partial plans produced as a result of adaptation. Since the results of splicing cause large jumps in the search space, this regime deliberately departs from the systematicity of SNLP and SPA in an attempt to solve the problem with less search.

However, allowing arbitrary plan retrieval and plan splicing is not without cost. Performing a full search of the system's case library at every step of the problem space could be computationally prohibitive. The costs of searching the case library at every step of the problem space could outweigh the benefits of reduced search, especially if the system enters a "slump" — an adaptation episode which begins and ends with the application of relevant clippings, but which goes through a series of intermediate plans for which the system cannot match any existing plans in its case library. Clearly, it is worthwhile to retrieve and apply clippings at the beginning and end of a slump, but a full search of the case library at each intermediate step could cost more than the benefits that the initial and final retrievals provide. This is the *swamping utility problem* — the

benefits of case retrieval can be outweighed by the costs of that retrieval, leading to an overall degradation in performance as a result of case learning (Francis & Ram, 1995).

#### **8.7.4. Expected Benefits of Multi-Plan Adaptation**

Both adapting a single partial plan and adapting merged partial plans can produce significant benefits over generative problem solving. The cost of generative planning is exponential in the size of the final plan produced, whereas fitting a plan is a linear operation in the size of the plan. Hence, the potential exists for substantial improvement through retrieval and adaptation if an appropriate past plan exists, especially for large plans. In certain domains, SPA has demonstrated significant improvements over generative planning. However, if large gaps exist in the retrieved partial plan, SPA must resort to adaptation, which, like generative problem solving, has an exponential cost in the number of steps that need to be added.

While this amount of adaptation may be a significant improvement over complete from-scratch problem solving, the potential exists to reduce that even further by using MPA to clip and splice more partial plans. Fitting a clipping and splicing a clipping are linear operations in the size of the plan being spliced. Hence, the potential exists for substantial improvement through plan merging if an appropriate past plan exists, especially if the gaps in the existing plan are large.

### **8.7.5. Prior Work in Plan Reuse and Multi-Plan Adaptation**

Experience-based agency as implemented in Nicole-MPA is not the first approach to reusing past experiences in a case-based fashion in least-commitment planning. The most relevant example of that work to this research is of course SPA, upon which Nicole-MPA builds. Other similar plan reuse systems include PRIAR (Kambhampati & Hendler 1992) an SPA-like system based on NONLIN, and XII (Golden et al. 1994), an SPA-like system that plans with incomplete information. Hanks and Weld (1995) discuss these and other plan reuse systems from the perspective of the SPA framework.

MPA's plan splicing mechanism is in many ways similar to DERSNLP (Ihrig & Kambhampati 1994), a derivational analogy system built on top of SNLP that uses eager replay to guide a partial order planner. While DERSNLP's eager replay mechanism is in some ways similar to a limiting case of Systematic MPA in which a single plan is retrieved and spliced into a skeletal plan derived from an initial problem statement, DERSNLP goes beyond SPA's reason mechanism and includes a full derivational trace of problem solving in its cases. While DERSNLP and its extension DERSNLP-EBL focus on when it is profitable to retrieve a partial plan, unlike Nicole-MPA they do not provide the capability of interrupting adaptation as a result of an spontaneous memory retrieval, nor do they provide the ability to integrate the results of multiple plans.

Similarly, combining multiple plans in case-based reasoning has a long research history: Watson & Perera (1997) review and compare a variety of such approaches. The

PRODIGY/ANALOGY system (Veloso 1994) can retrieve and merge the results of an arbitrary number of totally ordered plans during the derivational analogy process. Because PRODIGY/ANALOGY manipulates and stores totally ordered plans, it runs into significant issues on deciding how to interleave steps (Veloso, 1994, p124-127), an issue MPA was designed to avoid because of its least-commitment heritage but which in reality poses new problems in the form of a combinatorial explosion. Furthermore, PRODIGY/ANALOGY deliberately limits its capability to retrieve and combine cases on the fly in an attempt to reduce retrieval costs.

The ROUTER path-planning system (Goel et al., 1994) has the ability to recursively call its case-based methods to fill in gaps in a planned route when no exactly matching case can be found. (PRODIGY/ANALOGY has a similar capability to call itself recursively, although the full implications of this ability have not yet been explored). Like Nicole-MPA, ROUTER has the ability to combine multiple cases, although only in a synchronous fashion because its memory does not support spontaneous retrieval. However, each of these cases must be complete cases; it does not have the ability to clip out the relevant portion of a case at retrieval time. ROUTER does have the ability to break a case up into subcases at storage time, but the results show that computational costs of this storage computation outweigh the benefits in improved retrievals (Goel et al. 1994, p. 63).

The JULIA system (Hinrichs 1992) also has the ability to combine pieces of several past cases, but this is largely a domain-independent algorithm for merging declarative

structures, rather than a planning system. The CELIA system (Redmond 1990, 1992) stores cases as separate *snippets*, case subcomponents organized around a single goal or set of conjunctive goals. Snippets provide CELIA with the ability to retrieve and identify relevant subparts of a past case based on the system's current goals. Note that while snippets are superficially similar to plan clippings, plan clippings are constructed dynamically during problem solving, whereas snippets need to be computed and stored in advance.

Clippings are similar to macro operators (Fikes et al. 1972) in that they use past experience to combine several problem solving steps into a single structure that can be applied as a unit, allowing the system to make large jumps in the problem space and avoid unnecessary search. However, macro operators differ from clippings in two important ways. First, macro operators are traditionally precomputed at storage time, whereas clippings are computed dynamically; second, macro operators are fixed sequences of operators, whereas clippings are partially ordered sets of operators that may be resolved in a wide variety of ways in the final plan.

Kambhampati & Chen (1993) built and compared several systems that retrieve partially ordered case-like “macro operators”. They demonstrated that least-commitment planners could take greater advantage of past experience than totally-ordered planners because of their ability to efficiently interleave new steps into these “macro-operators” during planning. While this work focuses primarily on interleaving new steps into single past plans, the explanations the authors advance for the efficiency gains they detected

could be extended to suggest that least-commitment planners would be superior to totally-ordered planners when interleaving multiple plans. The Nicole-MPA system provided a testbed to empirically evaluate this hypothesis.

## **8.8. Executing the Study**

To execute the planning study, I constructed traditional and experience-based versions of the MPA planner, several planning domains, several case libraries, and used these to conduct five evaluations that tested seven hypotheses about experience-based agency and its application to planning.

### **8.8.1. Planning Frameworks**

The Nicole-MPA system was designed to be a highly configurable planner. It had both command line and Nicole-driven versions, and could use a variety of different control regimes to effect different planning strategies. These strategies included generative planning, single-case adaptation, multi-plan adaptation, and metaplanning multi-plan adaptation:

**Generative Planning (SNLP).** The most basic version of MPA attempted to solve plans generatively, using the SNLP plan refinement framework operating under MPA's search control. The generative planner took as input a problem statement, which it transformed into a skeletal partial plan that served as the starting point of a search space for potential solutions. Generative planning occurred by successive "refinement" of

partial plans to eliminate holes and flaws, using a frontier of partial plans searched in a best-first fashion based on a search heuristic.

**Single-Case Adaptation (SPA).** The next version of MPA attempted to solve plans through single-case adaptation, using the SPA plan fitting operations to produce partial plans. Partial plans were then adapted into complete plans in a refinement process identical to the search in generative planning.

**Search-Space Multi-Plan Adaptation.** Several versions of the MPA algorithm itself were constructed. An initial standalone implementation of the MPA algorithm was constructed to test the algorithm itself. This version of extracted intermediate goal statements from partially completed plans, used an enhanced version of the SPA fitting algorithm to extract the relevant parts of merged cases, and included a novel plan splicing algorithm to merge fitted cases with partial plans into new plans.

The next version of the multiple plan adaptor algorithm provided the additional infrastructure to merge multiple plans within the existing search space. The MPA algorithm was expanded and extended to ensure that it could handle all of the conditions that might arise in more complex plans, including matching variablized conditions to varibilized plans. The algorithm was then decomposed into component parts and was integrated into a Nicole-Taskstorm task network. This task network could simulate a variety of search techniques, including both traditional best-first search and a variety more deliberative search processes. It could also apply a variety of techniques at each

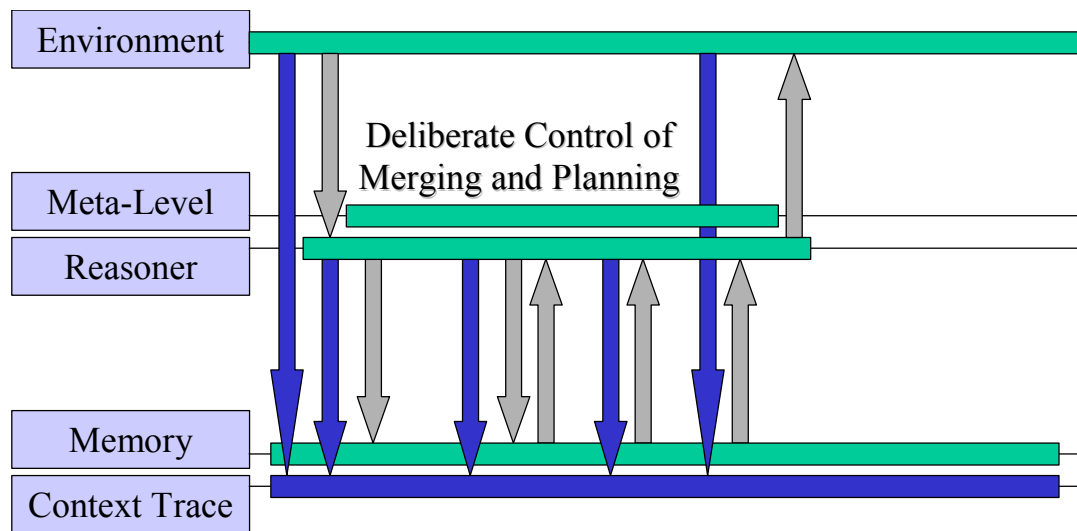


point in the search space, including heuristics for merging, the potential to discard parts of the search space when merging occurred, and the ability to perform postprocessing on merged plans. In all versions of this planner, partial plans for merging were drawn from the frontier of the search space, and merged plans were then returned to the frontier for further adaptation into a solution.

For additional tests of the algorithm, a command-line tool was constructed which mirrored the performance of the task network, enabling the latest MPA algorithm to be tested in generative (SNLP-like), adaptive (SPA-like) and multi-plan adaptive (MPA) modes without the overhead of memory retrieval and task control.

**Metaplanning Multi-Plan Adaptation.** The final version of the multiple plan adaptor algorithm used a metaplanning control strategy to attempt to provide greater control over retrieval, merging and adaptation.

Unlike a traditional planner which attempts to solve problems by searching the space of partial plans using operators at the level of the planning domain, a metaplanner searches a higher-level space whose operators are whose operators are high-level reasoning operations like generative planning, plan merging, adaptation, and other plan transformations (Stefik 1981, Wilensky 1984). Like a traditional planner, a metaplanner may possess a search frontier of partial plans it is working on; however, a metaplanner can also explicitly maintain other resources such as a buffer of retrieved cases or a library of plan transformations. At each step in the search space, the metaplanner considers the



**Figure 8.6 Meta-planning Reasoning Integration**

plans, cases, transforms and other information at its disposal and selects a high-level reasoning operation, such as plan adaptation or case merging, in an attempt to produce a solution (Figure 8.6).

The metaplanner extended Nicole-MPA's existing configuration by explicitly maintaining four data structures: a frontier of partial plans, a buffer of cases, a library of merge heuristics, and a library of adaptation heuristics. The metaplanner had two reasoning operations available to it: "greedy" adaptation and case merging. Case merging has already been described. "Greedy" adaptation takes a partial plan and quickly attempts to resolve all remaining conflicts in the plan within a few adaptation steps; if no solution can be quickly found greedy adaptation exits and other operators are applied. Because of the interaction of these two operations, the metaplanner's control regime resembled most closely the "Extreme" strategy for multi-plan adaptation, achieving the bulk of its plan adaptation through the merging process.

The metaplanner used its libraries of merge and adaptation heuristics to guide its operator selections. These heuristics consisted of quick-and-dirty rules of thumb for judging whether a plan in the plan frontier was a good candidate for merging or adaptation. For example, one merge heuristic suggested that a plan was a good candidate for merging if it contained a small number of open goals and no ordering or binding conflicts. One adaptation heuristic suggested that a plan was a good candidate for greedy adaptation if the number of conflicts in the plan was fewer than the number of plan refinements greedy adaptation was configured to attempt.

At each step in the metaplanning process, the metaplanner first checks memory for additional retrievals, adding them to its retrieval buffers if found. It then selects a plan from the plan frontier, usually in depth-first fashion, testing the plan against its library of heuristics. If the plan is not suitable for any operators it is discarded; otherwise the planner first attempts to merge in additional cases, then attempts greedy adaptation upon the merged case. Effectively, this metaplanner performs depth-first search through the space of merged plans, augmented by greedy adaptation to find solutions that were a small number of adaptation steps away from the “merge waypoints” in plan space.

### **8.8.2. Planning Domains**

I constructed several domains to test the Nicole-MPA planner; all of these domains had the property that larger plans could be coherently viewed as an aggregation of several smaller plans. These domains included:

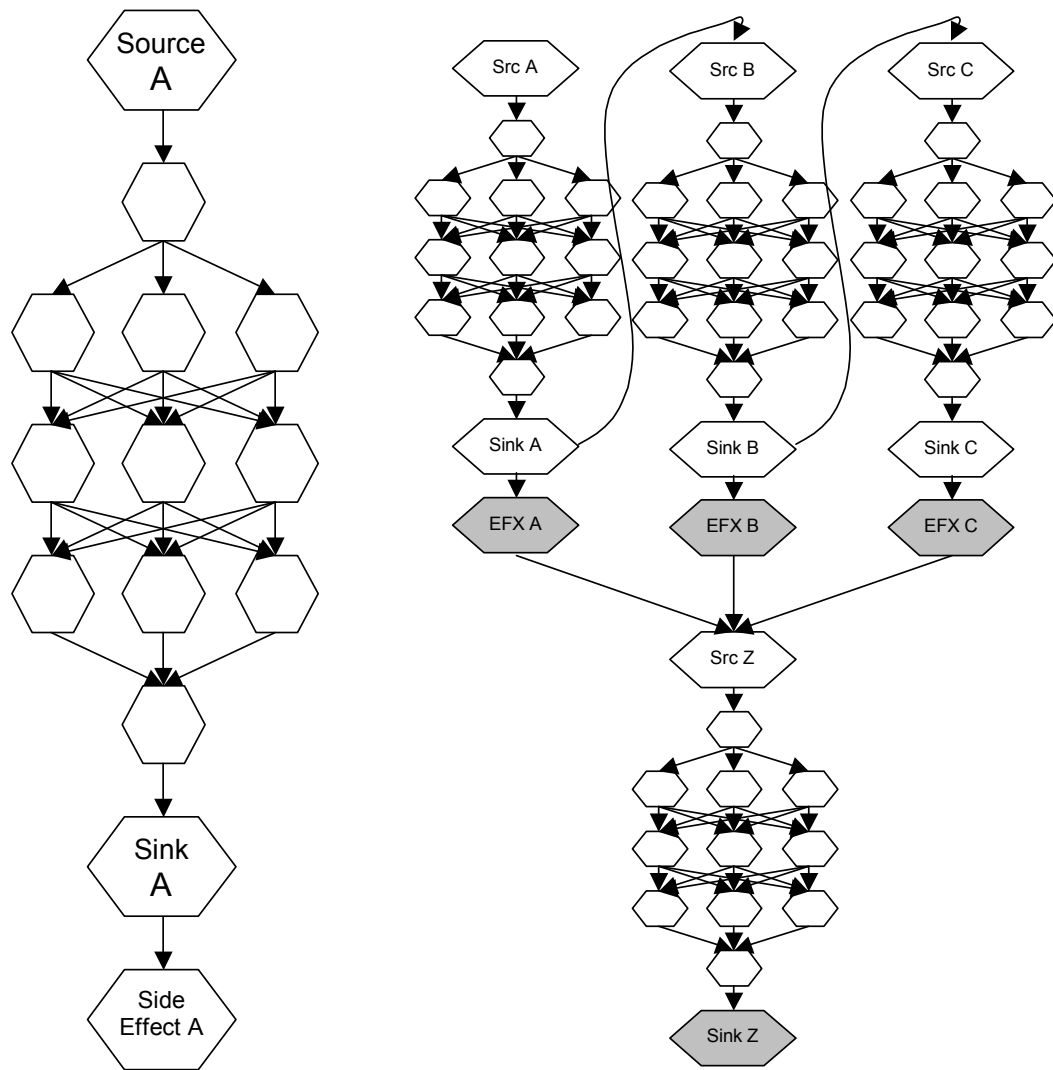
**Simple Travel Domain.** A simple domain was constructed for initial tests of the MPA algorithm. The “Simple Travel” domain was a small SNLP (STRIPS-style) planning domain that included 4 operators for traveling by car within a city, by a plane between cities, and getting and putting objects such as presentations and luggage. Appendix A.1 presents the Simple Travel domain in its entirety.

**Stinger Missile Domain.** The Simple Travel domain was completely rebuilt into a larger and more complex planning domain for testing the multi-case adaptation approach. This domain, called the “Stinger Missile” domain because it now included both tourists and terrorists, contained fifteen operators designed to support a variety of complex plans with potential subplans. The Stinger Missile was constructed as a domain for PRODIGY/ANALOGY and later ported to SPA. The additional objects and operators in this domain included enough complexity to support the construction of a larger and more realistic case library containing over 90 distinct cases. Appendix A.2 presents one variant of the Stinger Missile domain in its entirety.

**Abstract-3 Domain.** To further test the properties of the MPA algorithm, I developed an Abstract domain with the same overall structure as the Stinger Missile domain used in the earlier experiments. States in the Abstract domain are constructed entirely of sets of literals with no variables, eliminating binding problems; the complexity in the domain instead arises in its large number of operators (approximately 75) which define a large set of possible states and transitions between these states.

The Abstract domain contains many “subproblems” with a large number of operators which can be combined combinatorially in a variety of ways (Figure 8.5a); these subproblems produce side effects which contribute to other subproblems, enabling subproblems themselves to be combined into larger problems (Figure 8.5b). Just like the Stinger domain, this combinatorial complexity makes solving any individual problem difficult and solving a large problem almost impossible for traditional generative and case-based approaches.

However, the subproblems were only loosely coupled; beyond the side effect that one subproblem might contribute to the initial conditions of another, the solution to one subproblem does not impact the solution of any other subproblem. This mirrors the logical structure of the Stinger Missile domain, in which subproblems are also independent; however, it departs from the Stinger Missile domain by making this logical independence visible in the structure of individual plans rather than hidden within the domain.



(A) Subproblem Structure

(b) Overall Domain Structure

**Figure 8.5 Structure of the Abstract-3 Domain**

Appendix A.3 presents the Abstract-3 domain (so-called because of the size of the link matrices in its subproblems) in its entirety.

### 8.8.3. Case Libraries

In addition to these domains, I constructed sets of case libraries for testing these domains and testing the properties of the Nicole-MPA planner. These libraries included the Foreign Conference, Inexperienced Terrorist, Comprehensive, and Abstract case libraries.

- *Foreign Conference Library.* The Foreign Conference library contained several small cases simply to test the core MPA algorithm in isolation. These small cases (solution size between 3 and 5 steps) could be composed to solve larger problems. Appendix A.4 presents the Foreign Conference library in its entirety.
- *Inexperienced Terrorist Library.* The Inexperienced Terrorist library was a case library for the Stinger Missile domain containing 12 cases. Like the Foreign Conference library, the Inexperienced Terrorist library contained several small cases which could be composed to solve larger problems.
- *Comprehensive Library.* The Stinger Missile domain contained many other cases not found in the Inexperienced Terrorist library. In addition, there were several other planning domains used in Nicole-MPA, many imported from original SNLP or SPA domain sets or constructed by hand. The Comprehensive Library contained around a hundred cases drawn from the Simple Travel, Stinger Missile, Blocksworld, Sussman, Regswap and Truckworld libraries. It was primarily used to test the properties of the CDSA algorithm; these tests will be discussed further in Chapter 10.

- *Abstract Library.* Finally, a library was constructed for the Abstract-3 domain to test the metaplanner's ability to dynamically merge multiple cases. This consisted of four small problems and a larger problem which could in theory be solved using by composing the smaller problems together.

#### **8.8.4. Supporting Work**

In addition to the efforts outlined above, I also instrumented the Nicole-MPA planner to collect information about the progress of planning. I also built a variety of knowledge bases to test how additional knowledge in the system affected context-sensitive asynchronous memory's ability to retrieve information for Nicole-MPA, a series of evaluations discussed in greater depth in Chapter 10.

### **8.9. Fidelity of the Study**

Nicole-MPA had its limits. No version of the planner used automatic cueing of spreading activation based on the contents of working memory; instead the Nicole-MPA task control system deliberately refreshed its retrieval requests at appropriate points in the planning cycle.

The search-space version of Nicole-MPA acted primarily as a "systematic" version of MPA: while its algorithms supported integrating new cases at any time, it nevertheless actively attempted to gather as many cases as possible at the beginning of problem solving. The metaplanning version of Nicole-MPA did not have this restriction.



Finally, the matching specifications that Nicole-MPA provided to the context-sensitive asynchronous memory process were not very fine-grained, causing Nicole-MOORE's retrievals to sometimes be imprecise. This did not effect the ability of Nicole-MOORE to retrieve many useful cases or to find cases weighted by context, but it did place most of the burden of determining the quality of retrieved cases on Nicole-MPA and not on the memory.

## 8.10. Hypotheses Tested

The goal of these evaluations was to show that it was possible to get better planning performance by combining multiple experiences dynamically and that feedback from the planning process could result in better retrieval. For a variety of reasons these evaluations focused on the first of these goals; analysis of the contribution of feedback from reasoning on retrieval are discussed in Chapters 9 and 10.

The overall goal of determining whether experience-based multi-plan adaptation could result in better planning performance was unpacked into a variety of hypotheses about both multi-plan adaptation and the potential benefits of context-sensitive asynchronous memory to the planning process. These hypotheses included:

- **multi-plan adaptation is useful:**

Planning systems can merge multiple cases to solve problems, resulting in a greater performance improvement than possible with a single case.

- **least-commitment planning is a useful platform for integrative reasoning:**

The least-commitment planning approach, because it delays decisions about the ordering of steps, would easily support combining multiple plans dynamically.

- **integrative reasoning can be achieved using cases as extended operators:**

A case-based problem-solving process based on search of a problem space in which retrieved cases are spliced into the current state to provide greater “leaps” through the search space can solve problems faster than a traditional search using only standard operators.

- **integrative reasoning can be driven by spontaneous retrieval:**

the control of integrative reasoning can be driven by spontaneous retrievals

- **interactive multi-plan adaptation is useful:**

The Interactive MPA approach, which exploits context-sensitive asynchronous memory to guide the integration of planning cases, would provide benefits over competing approaches for solving difficult problems.

The experimental evaluation of these claims validated some of them and disconfirmed others. The first claim, that combining multiple planning cases resulted in improved performance, was validated. The next three claims — all claims about how multiple planning cases could be integrated into a context-sensitive asynchronous memory framework — were all initially disconfirmed. Analysis of each of these claims suggested alternative hypotheses:

- **platforms for integrative reasoning must control ordering and binding problems:**

A platform for integrative reasoning must limit the interactions between parts of dynamically combined plans to prevent a combinatorial explosion. In particular, some commitments must be made about how to order steps and bind variables as plans are combined to make multi-plan adaptation effective.

- **integrative reasoning can be achieved using metaplanning:**

Integrative reasoning can be effectively achieved by a metaplanning process which treats adaptation and integration as operators in a high-level planning process. Rather than drive reasoning by spontaneous retrieval, metaplanning makes deliberate decisions about how much effort to expend on adaptation and on how to integrate retrievals.

Additional experiments were conducted to test these hypotheses. All were confirmed, and the system which combined these components successfully demonstrated the final hypothesis, that interactive multi-plan adaptation can provide benefits over competing approaches for solving difficult problems, and by extension provided evidence that context-sensitive asynchronous memory can provide benefits to the planning task.

## 8.11. Results

I used the Nicole-MPA system to conduct 5 exploratory studies, analyzing the MPA algorithm itself, the impact of multi-plan adaptation on search, and several alternative approaches to experience-based multi-plan adaptation. These evaluations included:

- **Evaluation 8.1: Validity and Usefulness of the Multi-Plan Adaptation Algorithm**
- **Evaluation 8.2: Experience-Based Search-Space Multi-Plan Adaptation**
- **Evaluation 8.3: Detailed Analysis of Multi-Plan Adaptation**
- **Evaluation 8.4: Controlling Ordering And Binding Problems In Integrative Reasoning**
- **Evaluation 8.5: Experience-Based Metaplanning Multi-Plan Adaptation**

The efforts conducted in these experiments and their chief results follow.

### 8.11.1. Evaluation 8.1: Validity and Usefulness of the Multi-Plan Adaptation Algorithm

To provide an initial test of the MPA algorithm's ability to provide improvement by combining multiple plans in a least-commitment planning framework, an initial standalone implementation of the MPA algorithm was constructed, along with a simple travel planning domain. This algorithm and domain were used to test whether the MPA

algorithm could be used to combine multiple cases to improve performance over SPA or generative planning.

**Method.** Five small (3-5 step solution size) problems were constructed for the Simple Travel Domain, including Domestic Conference presentations, Overseas Vacations not involving a presentation, and a Foreign Conference problem combining both presentations and overseas travel. To build a case library, all of these problems were solved using strict generative planning and the number of planning nodes expanded was recorded.

Then, to test the MPA algorithm, two conditions were compared. First, SPA was allowed to solve problems using a single past case for adaptation. Then, the MPA prototype was used to merge two cases together for adaptation. Speedups were computed based on the number of nodes saved during the adaptation cycle by MPA over SPA.

**Results.** The results of this initial assay indicated speedups even though the test problems were small — 30% in the case of the “Simple Travel” Foreign Vacation problem with a solution size of only 5 steps. This comparison of single-case and multi-case adaptation showed that, as measured by number of expanded planning nodes, multi-case adaptation could provide improved performance. On the basis of these results, more extensive tests of multi-plan adaptation were conducted in Evaluation 2.

### 8.11.2. Evaluation 8.2: Experience-Based Search-Space Multi-Plan Adaptation

The MPA algorithm was designed to be the core of an experience-based agent that exploited multiple past experiences to solve problems more quickly and efficiently.

To test this, I expanded and extended the core MPA algorithm developed in the previous evaluation, ensuring that it could handle all of the conditions that might arise in more complex plans. I then integrated this revised algorithm into the Nicole task network, creating a parameterizable planner that could operate in generative, single-case adaptive or search-space experience-based modes. Finally, I developed the “Stinger Missile” domain, a larger and more complex version of the Simple Travel domain that supported the construction of more complex plans.

I then conducted a set of exploratory tests on Nicole-MPA using a “systematic” control regime that attempted to gather as many plans before reasoning as possible. These tests were conducted in generative, single-case and multi-case adaptive planning modes.

**Method.** The experience-based agent approach to improving a planner’s performance depends on exploiting multiple past experiences to solve challenging problems more quickly and easily.

To simulate a store of past knowledge, I created a case library called “Inexperienced,” consisting of nine simple problems, such as traveling within a city, taking a domestic flight, or packing luggage, which could be solved relatively quickly using generative problem solving. These cases were intended to represent the types of past experiences that an agent would have after a relatively short exposure to a problem solving domain. To simulate a set of more challenging problems that would require the use of past experience, I then constructed a problem library called “X2” (for eXperimental evaluation 2) consisting of 12 larger test problems, all of which could be solved by using one or more past cases from the Inexperienced library.

All of these problems and cases were entered into Nicole’s experience store. Because these problems and solved plans were in the native SPA plan format, each was automatically annotated with “plan tags” in the CRYSTAL format, enabling past cases to be retrieved by the Nicole-MOORE context-sensitive asynchronous memory system on the basis of cues from current problems. A more detailed analysis of the plan tagging system and its contribution to context-sensitive asynchronous memory retrieval will be presented in Chapter 10.

I then conducted a series of evaluations, testing Nicole-MPA’s ability to solve problems from the X2 problem set using past experiences from the Inexperienced library retrieved using the Nicole-MOORE context-sensitive asynchronous memory system. These tests exercised all of Nicole-MPA’s generative, single-case adaptive, and multi-case adaptive problem solving modes.

**Results.** Nicole-MPA showed the ability to solve all the plans in the X2 problem set in generative problem solving mode, but ran into combinatorial explosions in both single-case and multi-case adaptive modes using cases from the Inexperienced library.

The largest problem arose in multi-plan adaptation when adapted cases were merged back into the same search space. Nicole-MPA consistently ignored merged cases in favor of shorter non-adapted cases on the planning frontier, at the cost of extra planning effort. When Nicole-MPA was configured to discard the planning frontier after a merge, it often failed to find a solution. This problem was particularly acute on the largest problems, even though Nicole-MPA had access to a set of subcases that completely solved all components of the problem. Nicole-MPA would attempt to combine the cases but would bog down during adaptation efforts, unable to finish adapting the solution.

The original intent of Evaluation 2 was to lay the groundwork for a full head-to-head experimental evaluation, testing generative, single-case and multi-case adaptation against each other. Because of the difficulties that arose with multi-case adaptation in these exploratory evaluations, I decided to conduct a more detailed analysis of the multi-plan adaptation algorithm on the Stinger Missile domain to verify that the results of Evaluation 1 truly carried over to the new domain.

### **8.11.3. Evaluation 8.3: Detailed Analysis of Multi-Plan Adaptation**

My initial hypothesis about the results of Evaluation 2 were that there was an error in the Stinger domain or problem set, or that there was an error in the MPA algorithm. To



test this hypothesis, I conducted more detailed analyses of generative, single-case adaptive and multi-plan adaptation problems on the Stinger Missile domain using the command-line version of the MPA algorithm.

**Method.** Using the command-line tool, I directly compared generative (SNLP), single-case adaptive (SPA) and multi-case adaptive (MPA) planning using number of planning nodes expanded as my evaluation metric. In addition to tests using the Inexperienced Library and the X2 problem set, I also constructed new source cases and test problems to examine the properties of the MPA algorithm. This included greatly expanding the set of source cases beyond those in the Inexperienced library, varying problems from the X2 set to make them more or less difficult through the addition or deletion of conditions, and creating new “ideal” problems which should be easy to solve given certain sets of past cases. This wide variety of problems, from extremely challenging to ideal, was designed to probe both favorable and unfavorable conditions for multi-plan adaptation.

**Results.** Three negative results emerged from this exploratory study. First, certain large plans were too difficult for any of the problem solvers to solve even though they were qualitatively only slightly more difficult than related plans. For example, the “Hard” variant of the “Foreign Conference Problem,” which simply restated the goal of the problem to be to give a presentation in a target city rather than to just arrive in the target city with a presentation, was unsolvable by any version of the planner. On large problems, all versions of the planner — generative, single-case adaptive and multi-case

adaptive — encountered search explosions which undermined their ability to solve problems.

Second, in certain conditions combining multiple cases could produce worse effects than single-case adaptation or even generative problem solving. For example, on a less difficult variant of the Foreign Conference problem, generative problem solving examined 863 partial plans in the search space. Where adapting a single case could reduce the cost to 257 partial plans, but then adding a second case caused the problem solving cycle to explode back to 858 partial plans. The amount of effort expended was highly problem dependent; for example, on one problem in the Stinger Missile domain generative problem solving time examined more than 3000 partial plans without producing a solution, whereas single-case adaptation produced a solution by examining only 142 partial plans; a similar problem was solvable by generative problem solving by 235 plans whereas adapting a single case caused the cost to increase to 300 plan nodes. In most cases, the additional cycles appeared to arise during the process of resolving ordering and binding conflicts in the merged plan.

Third, in many conditions merging multiple plans produced a partial plan which incorporated structure from both source plans, but which could not be further refined or adapted because of ordering and binding conflicts. This occurred even when the structure of the merged plan was identical to the structure of the ultimate solved plan. Paradoxically, some of the larger cases which caused the worst search explosion did not suffer from this refinement problem.

Detailed analysis of solved and partially solved problems did not reveal any errors in the domain or problem sets; nevertheless, no set of cases in the Stinger Missile set showed improvements from multi-plan adaptation. This result contradicted the results of Evaluation 1 and made it impossible to test the application of the context-sensitive asynchronous memory approach; moreover it threw doubt on the validity of the MPA algorithm itself. I therefore decided to try to vary the properties of the domain in question to illuminate potential sources of problems.

#### **8.11.4. Evaluation 8.4: Controlling Ordering And Binding Problems In Integrative Reasoning**

Analysis of the results of Evaluations 2 and 3 appeared to show that the complexity of the Stinger Missile domain introduced additional ordering and binding conflicts which made plans either unrefinable or which caused a search explosion. I hypothesized that a planning domain that controlled the kinds of variable binding problems that could arise would eliminate these problems.

**Method.** To test this hypothesis, I developed an Abstract domain with the same overall structure as the Stinger Missile domain used in the earlier experiments. The Abstract-3 domain, as already discussed, contained a fairly complex structure, making most problems in the domain difficult to solve generatively; however, it did not contain complex binding constraints which appeared to cause many of the difficulties with the Stinger Missile domain.

```

-----
--- SOLUTION:
-----

Plan-Object #k<REFINED-PLAN.806> (IPLAN #<NIL: S=22; O=0; U=0>)
Initial:
  ( [(SOURCE-A)]      [(SOURCE-B)]      [(SOURCE-C)]      [(SINK-NULL)]      )
Order  Id      Action
1      [22]     (TRANSFORM-SOURCE-A-SINK-NULL-TO-MATRIX-ENTRY-A)
2      [21]     (TRANSFORM-MATRIX-ENTRY-A-TO-MATRIX-ELEMENT-A-1-1)
3      [20]     (TRANSFORM-MATRIX-ELEMENT-A-1-1-TO-MATRIX-ELEMENT-A-2-1)
4      [19]     (TRANSFORM-MATRIX-ELEMENT-A-2-1-TO-MATRIX-ELEMENT-A-3-1)
5      [18]     (TRANSFORM-MATRIX-ELEMENT-A-3-1-TO-MATRIX-EXIT-A)
6      [17]     (TRANSFORM-MATRIX-EXIT-A-TO-SINK-A)
7      [16]     (TRANSFORM-SINK-A-TO-SIDE-EFFECT-A)
8      [ 8]     (TRANSFORM-SOURCE-B-SINK-A-TO-MATRIX-ENTRY-B)
9      [ 7]     (TRANSFORM-MATRIX-ENTRY-B-TO-MATRIX-ELEMENT-B-1-1)
10     [ 6]     (TRANSFORM-MATRIX-ELEMENT-B-1-1-TO-MATRIX-ELEMENT-B-2-1)
11     [ 5]     (TRANSFORM-MATRIX-ELEMENT-B-2-1-TO-MATRIX-ELEMENT-B-3-1)
12     [ 4]     (TRANSFORM-MATRIX-ELEMENT-B-3-1-TO-MATRIX-EXIT-B)
13     [ 3]     (TRANSFORM-MATRIX-EXIT-B-TO-SINK-B)
14     [ 2]     (TRANSFORM-SINK-B-TO-SIDE-EFFECT-B)
15     [15]     (TRANSFORM-SOURCE-C-SINK-B-TO-MATRIX-ENTRY-C)
16     [14]     (TRANSFORM-MATRIX-ENTRY-C-TO-MATRIX-ELEMENT-C-1-1)
17     [13]     (TRANSFORM-MATRIX-ELEMENT-C-1-1-TO-MATRIX-ELEMENT-C-2-1)
18     [12]     (TRANSFORM-MATRIX-ELEMENT-C-2-1-TO-MATRIX-ELEMENT-C-3-1)
19     [11]     (TRANSFORM-MATRIX-ELEMENT-C-3-1-TO-MATRIX-EXIT-C)
20     [10]     (TRANSFORM-MATRIX-EXIT-C-TO-SINK-C)
21     [ 9]     (TRANSFORM-SINK-C-TO-SIDE-EFFECT-C)
22     [ 1]     (TRANSFORM-SIDE-EFFECT-A-SIDE-EFFECT-B-SIDE-EFFECT-C-TO-TARGET-Z)
Goal:
  ( [(TARGET-Z)]      )
No Open.

```

**Figure 8.7 A Large ABSTRACT-3 Problem Solved using Multi-Plan Adaptation**

Using the Abstract-3 domain, I conducted exploratory tests with the command-line version of the Nicole-MPA planner that paralleled the Stinger Missile tests in Evaluation 3, with the goal of determining whether there were any fundamental flaws in the MPA algorithm that could have contributed to the negative results reported in Evaluation 2 and 3.

**Results.** The command-line version of MPA was successfully able to merge multiple plans and produce adapted solutions on the Abstract domain. Like Stinger, large problems containing many subproblems in the Abstract domain are difficult for either

generative or single-case case-based approaches to solve. The largest problem in the Abstract domain was unsolvable by both the generative and single-case adaptation approaches, no matter what heuristics for search were used or how much time or effort the planner was allotted (up to the memory limits on the available machine). However, merging multiple cases by hand in the Abstract domain showed that this problem was solvable rapidly by combining multiple cases and following with a short adaptation step.

These results provided preliminary evidence that the merging and adaptation algorithms of Nicole-MPA were not faulty: given a suitable domain they can not only function but can solve problems not solvable by other approaches. The challenge now was to demonstrate that this multi-case merging could work in the wild, controlled autonomously by a planning system using a context-sensitive asynchronous memory.

#### **8.11.5. Evaluation 8.5: Experience-Based Metaplanning Multi-Plan Adaptation**

The final problem to resolve to validate the MPA algorithm was to show that a full experience-based version of the planner could successfully combine multiple experiences based on context-sensitive asynchronous memory retrieval. I hypothesized that a metaplanning version of Nicole-MPA that deliberately controlled the merging of plans could overcome the combinatorial explosion problems detected in the search-space version of Nicole-MPA.

**Method.** To test this hypothesis, I constructed a prototype of a metaplanning system within Nicole-MPA, and constructed a case library of small problems in the Abstract-3 domain. The prototype metaplanner was tested on the Abstract-3 case library and problem set in a way parallel to the tests of the search-space version of MPA on the Stinger Missile domain. Most significant of the problems in the problem set was a very hard problem from the Abstract-3 domain, shown in the previous evaluation to be unsolvable by generative or single-case adaptation.

**Results.** The metaplanner was successfully able to produce solutions to every problem given to it either directly or through use of its case library. This included the large Abstract problem, the solution to which is shown in Figure 8.7; this problem was solved in 96 seconds over 79 Nicole cycles. As mentioned, this problem could not be solved by traditional generative or case-based approaches because of its complexity.

## 8.12. Explanation and Analysis

These evaluations confirmed some hypotheses about how a reasoning task could exploit a context-sensitive asynchronous memory and disconfirmed others. For example, these results showed that a planning system could use cases retrieved by a context-sensitive asynchronous memory to improve a planner's performance; however this benefit was only shown in extremely restricted circumstances; moreover the successful use of retrievals from the context-sensitive asynchronous memory did not exploit the special properties of context-sensitive asynchronous memory in any useful way.

To understand what conclusions we can draw from these results, let us return to the hypotheses that motivated this study and examine what these evaluations verified and falsified.

### 8.12.1. Usefulness of Multi-Plan Adaptation

Planning systems can merge multiple cases to solve problems, resulting in a greater performance improvement than possible with a single case.

**Status of Hypothesis.** Both Evaluations 8.1 and 8.5 showed that multiple cases could be combined to provide improved planning performance over that available with a single case. While these were limited domains and implementations, these tests indicated significant speedups could be had with the multi-case adaptation approach over the single case adaptation approach. These evaluations also showed that those speedups were radically dependent on the properties of the domain and the problems solved.

**Discussion.** These results are consistent with prior work on multiple plans in case-based reasoning, which have been shown to be useful as far back as the first case-based reasoner, MEDIATOR (Kolodner & Simpson 1988) as well as in a wide variety of systems that followed (e.g., Redmond 1990, Veloso 1995, Goel et al. 1994). Nevertheless, it was important to demonstrate that multi-plan adaptation is useful within the least-commitment planning framework within which this case study was conducted.

Ultimately, the claim that reasoning combining multiple plans provides improvements over single-plan adaptation is not so much a claim of my research but instead a condition of the task and domain which must be true for the model to apply.

### 8.12.2. Partially-Ordered Planning as a Platform for Integrative Reasoning

The least-commitment planning approach, because it delays decisions about the ordering of steps, would easily support combining multiple plans dynamically.

The choice of least-commitment planning as the basis for this case study was based on its property of delaying decisions about step orderings and bindings as much as possible to prevent backtracking. The core idea is that a least-commitment planning framework would readily enable the combination of multiple plans at the point of causal dependencies between steps; then the specific ordering of steps in merged plans could be resolved as part of the adaptation process.

**Status of Hypothesis.** The results of these evaluations, particularly Evaluation 8.3, disconfirmed this hypothesis. These evaluations showed that while combining multiple cases into a partially ordered plan and allowing the least-commitment adaptation process to resolve ordering conflicts was effective for the small Simple Travel planning domain, it failed for the more complex Stinger Missile domain. Dynamic clipping and merging of



partially-ordered plans introduced both ordering and binding problems which required enough additional search to swamp the potential benefits of retrieval.

**Discussion.** Merging multiple plans caused ordering problems because the computational complexity of resolving ordering conflicts in a merged plan using adaptation is exponential. Even though the causal structure of an entire plan could be constructed in just two or three merging steps, resolving the dependencies in the plan was actually more difficult than planning from scratch because the merging process added steps in different orders and combinations that would be expected in normal planning. This created additional conflicts within a merged plan, and each additional ordering conflict added to the possible plans in the search space exponentially.

More importantly, Evaluations 8.3 and 8.4 showed that merging multiple plans caused binding problems. Both single-plan and multi-plan adaptation variabilize plans prior to fitting and clipping, but because cases in multi-plan adaptation are fitted to an intermediate goal statement that itself may contain variables, the resulting clippings contain more unresolved bindings than their single-plan counterparts. As a result, when these clippings are merged into a plan the resulting plan may contain binding “problems” which, while technically not flaws in the plan, nevertheless cannot be resolved during the normal adaptation process. The result of this problem was that even in circumstances when several cases were present that could be easily merged to solve a difficult problem, the planner was unable to resolve the conflicts in the merged plan and thus was unable to reach a solution using merging.

**Caveats.** It is important to mention a number of caveats. In general, design of domains in planning systems is difficult, as is the design of appropriate problems and cases; careful effort must be expended to ensure that domains and problems are constructed in such a way that a planner can find a solution. Similar caveats apply to least-commitment planning or to any planning system; algorithms which manipulate plans must be designed very carefully to prevent incorrect plan transformations from undermining the planning process. Therefore, because additional research might reveal ways around these problems — for example, a separate post-merge cleanup step could potentially remove some ordering and binding problems, a better designed algorithm might produce better intermediate plans, or a better designed domain or case library might be more amenable to merging — I cannot claim that these results falsify the hypothesis that least commitment planning would serve as a good platform for multi-plan adaptation. However, they certainly have disconfirmed the naïve form of this hypothesis.

**Overcoming the Caveats.** These results did suggest a number of solutions to these problems. For example, because a state-space planning representation by its nature does not contain unresolved orderings and unbound variables, it might serve as a better platform for multi-case adaptation. Evaluations 8.4 and 8.5 showed that another viable solution (and one that did not require building an entirely new planner) is to restrict the planning domain to reduce the potential of binding problems.

### 8.12.3. Integrative Reasoning Using Cases as Extended Operators in State Space

A case-based problem-solving process based on search of a problem space in which retrieved cases are spliced into the current state to provide greater “leaps” through the search space can solve problems faster than a traditional search using only standard operators.

Another of the original hypotheses was that merging and adaptation could occur within the same search space. As cases were adapted, at any point one could be selected for potential merging, combined with a case, and returned to the same search space.

**Status of Hypothesis.** Evaluation 8.2 showed that splicing cases in the same search space either caused a search explosion which swamped all performance improvement or led the system down garden paths in which no solution was possible.

**Discussion.** The default metric used for most planning domains is plan length. A merged plan will in general be longer than the “ordinary” plans (composed entirely from single-step operators) which populate the search space of a planner. A search algorithm such as best-first search, which attempts to return the optimal (read: shortest) plan, will “ignore” merged cases because its heuristic evaluation function will penalize the merged cases for their extra length.

These results are to some degree consistent with research on macro-operators. Like merged cases, macro-operators encapsulate past knowledge which can be used to provide

similar “leaps” through the search space, although macro-operators are learned in a slightly different way (macro-operators are generalized and abstracted away from their concrete initial conditions, where cases may not be) and are selected for application in a completely different fashion from a retrieved case (cases are retrieved based on similarity, whereas macro-operators are matched against the current problem solving state like other operators). The key discovery about macro-operators, however, which was most relevant to the early algorithms for merging cases in the early implementations of Nicole-MPA, was that the additional costs of search with macro-operators can outweigh the benefits they might provide (Etzioni 1992).

To overcome these problems, I conducted tests as part of Evaluation 3 using a variety of heuristic evaluation functions which attempted to analyze plans and preferentially select merged cases as promising choices; unfortunately no heuristic was found that improved the system’s performance. Every heuristic that successfully selected for merged plans also resulted in the generation of suboptimal plans. The only successful heuristic found was to clear the plan frontier once merging had taken place, effectively forcing the planner to adapt only the merged plan.

Unfortunately this strategy presented its own problems. The majority of partial plans generated during the adaptation process were terrible plans for adaptation, containing unresolved binding and ordering conflicts which made it impossible to fit and merge cases. Attempting a merge with each partial plan was computationally disastrous; heuristics for guiding when to merge were by and large ineffective.

The combination of these two problems was especially killing. In order to adapt a merged case, the planner needed to clear the frontier; but in order to get a suitable plan for merging the planner needed to try as many plans on the frontier as possible. In almost every circumstance, the planner made an incorrect judgment about when to clear the plan frontier, preventing it from finding a solution.

**Caveats.** As a result of these experimental evaluations, I cannot claim that this prediction was falsified —a different implementation or a superior heuristic function may have overcome these difficulties, and the failed tests in fact suggest several novel research directions in planning which might actually solve these problems — it was nonetheless disconfirmed in its naïve form.

**Overcoming the Caveats.** Once again, these results suggested a number of solutions to these problems. Because simple heuristics could not guide the system on when to merge and when to adapt, and because the combination of merging and adaptation in the same search space presented its own problems, this suggests that a more complex decision algorithm with a richer search space might be able to overcome these difficulties. This “metaplanning” approach, in which a reasoner “plans” using merging, adaptation and generation “meta-operators” that are responsible for manipulating actual operators and plans, was successfully tested in Evaluation 8.5.

#### 8.12.4. Integrative Reasoning Can Be Driven By Spontaneous Retrieval

The control of integrative reasoning can be driven by spontaneous retrievals.
---

Another original hypothesis was that the control of integrative reasoning could be driven by the timing of spontaneous retrievals — in other words, that when a case is retrieved that it could be directly merged with the plans in the current state space.

**Status of Hypothesis.** Interactive multi-plan adaptation could never be fully tested in Nicole-MPA. Under most conditions, Nicole-MOORE's memory system could find all relevant cases in the knowledge base before the planning system was even ready to begin adaptation, and so the opportunity to perform interactive multi-plan adaptation never arose.

Furthermore, the results of Evaluation 8.2 appeared to show that even if opportunities for interactive merging had arisen, directly merging cases back into the same search space could potentially incur significant extra costs. In a planner that uses a search space of partial plans, merging retrievals back into the search space could balloon the size of the search space and incur costs of attempting to merge many plans.

**Discussion.** While the Stinger Missile domain was larger than the Simple Travel domain, it still contained less than a hundred cases and did not sufficiently challenge the context-sensitive asynchronous memory system. Nicole-MOORE simply retrieved cases

too rapidly for interactive multi-plan adaptation, finding all relevant cases almost immediately.

Both systematic and semi-systematic versions of Nicole-MPA were reasonably successful at exploiting past cases in retrieval, but were unsuccessful at solving problems. On most problems, so many opportunities for plan merging arose during the planning process that the planner paid a high cost for merging and received in return an exploded search space subject to all of the problems discussed in the last section.

As planning proceeds, the search frontier can become exponentially large, making it impractical to attempt merging each retrieved case with every plan on the frontier. Furthermore, there is often little relationship between the current focus plan and an spontaneously retrieved case, making merging impractical or impossible.

Even contextually-driven spontaneous retrieval suffers from this problem. Cues are generated by a partial plan at two points: when the plan is first generated by refinement from an existing focus plan, and when the plan itself becomes the focus for refinement. However, the gap between cue generation and retrieval of a relevant plan means that the plan either has disappeared into the search frontier by the time that a retrieval is found, or that the plan has already been expanded and discarded.

**Overcoming the Problems.** Given the other issues which arose in merging multiple plans, this problem was actually minor. However, it does suggest that in order to

correctly orchestrate spontaneous retrieval and plan adaptation, a more complex approach is needed which tracks the “holes” in partial plans generated as part of the adaptation process, and which can make an explicit decision about whether to “backtrack” to a previous case when an spontaneous retrieval occurs, whether to “hold” an spontaneous retrieval in the hope that it will be relevant to future plans, or to discard an spontaneous retrieval because it is no longer relevant. A primitive version of this approach was successfully tested in the metapanning version of Nicole-MPA in Evaluation 5.

#### **8.12.5. Controlling Ordering And Binding Problems In Integrative Reasoning**

A platform for integrative reasoning must limit the interactions between parts of dynamically combined plans to prevent a combinatorial explosion. In particular, some commitments must be made about how to order steps and bind variables as plans are combined to make multi-plan adaptation effective.

One solution to the ordering and binding problems discussed the previous sections was to restrict the planning domain to reduce the potential of binding problems. This hypothesis is that least-commitment planning itself is not the difficulty, but instead that particular kinds of domains within least-commitment planning cause difficulties which can easily be resolved. In particular, the domain must restrict potential binding problems as much as possible to enable multiple past cases to be merged into a single plan without



creating irreconcilable conflicts, and ordering problems must be restricted so that merging multiple cases does not cause a combinatorial explosion.

**Status of the Hypothesis.** Evaluation 8.4 provided evidence that the merging and adaptation algorithms of Nicole-MPA were not faulty: given a suitable domain they can not only function but can solve problems not solvable by other approaches.

#### 8.12.6. Integrative Reasoning Can Be Achieved Using Metaplanning

Integrative reasoning can be effectively achieved by a metaplanning process which treats adaptation and integration as operators in a high-level planning process. Rather than drive reasoning by spontaneous retrieval, metaplanning makes deliberate decisions about how much effort to expend on adaptation and on how to integrate retrievals.

Experiments with both plan merging in the abstract and with the timing of spontaneous retrievals suggested that attempting to use a traditional best-first search in the space of plan refinements provided insufficient control to support experience-based planning, and that an approach which exercised greater deliberation over the planning process was required. I hypothesized that a *metaplanning* approach would provide the necessary control over retrieval, merging and adaptation.

**Status of the Hypothesis.** Evaluation 8.5 confirmed the hypothesis that metaplanning could overcome the difficulties of pure state-based search for integrative reasoning.

### 8.12.7. Interactive Multi-Plan Adaptation Is Useful

The Interactive MPA approach, which exploits context-sensitive asynchronous memory to guide the integration of planning cases, provides benefits over competing approaches for solving difficult problems.

The results in the previous section give away the punchline of this one. The context-sensitive asynchronous memory approach predicts that an experience based agent would show a performance improvement over pure generative problem solvers and case-based reasoning systems which gather all their cases prior to problem solving, especially in conditions in which the cues necessary to gather a case do not exist until problem solving has begun.

**Status of the Hypothesis.** This prediction was validated using in Evaluation 8.5, in which a full experience-based agent version of Nicole-MPA based on the metapanning algorithm solved problems in the Abstract domain. Under these conditions, Nicole-MPA successfully solved a problem too difficult for generative problem solving and single-case problem solving to solve by integrating multiple spontaneously retrieved cases into a single solution.

## 8.13. Lessons Learned

This case study revealed both strengths and limitations in my initial hypotheses on how a planning system might exploit a context-sensitive asynchronous memory. The two most important hypotheses of the research were validated, with some caveats:

- **multiple plans can be merged for improved performance:**

Several experiments repeatedly demonstrated the usefulness of combining multiple plans, from simple test examples to large and complex plans unsolvable by generative or single-case adaptation methods. However, these same experiments showed that merging multiple plans could incur its own costs and that controlling those costs was crucial for exploiting multi-plan adaptation.

- **context-sensitive asynchronous memory can support multi-plan adaptation:**

The “performance improvement” demonstration using the metapanning version of Nicole-MPA on the Abstract domain demonstrated that a context-sensitive asynchronous memory could support the retrieval needs of a integrative reasoner combining multiple plans. There is a significant caveat to this statement: Nicole-MPA’s context-sensitive asynchronous memory was “too good” for its own good. During the performance improvement demonstration, Nicole-MPA successfully retrieved all the cases in the knowledge base relevant to the Abstract domain without any feedback cues from the planning process. While the metaplanner did not depend on receiving all relevant cases up front — in fact, cases could be retrieved at almost any point and in almost any order without affecting its operation as long as useful cases continued to arrive — it nevertheless failed to exploit the full benefits of context-driven asynchronous retrieval.

The case study also taught two important lessons about how to integrate a reasoning task with a context-sensitive asynchronous memory:

- **relevant content must be transparently represented in the experience store:**

If a reasoning module has its own “native” representation for knowledge items, it must expose the significant content of each item in the experience store’s own content-addressable knowledge representation, or a context-sensitive asynchronous memory approach will be unable to focus its search on items semantically similar to the current context and, because of the cost control policy, will be unlikely to find any useful results. Results related to this lesson will be discussed in greater detail in Chapter 10.

- **deliberate control is necessary for reasoning integration:**

Because the timing of spontaneous retrievals is driven by the timing of the search of memory and not by the timing of reasoning, the retrieval of cases or other items is likely to be disassociated from the moment-to-moment information needs of the reasoning task. To fully exploit spontaneous retrievals, a deliberate control process is needed which monitors the information needs of the reasoning task as they are generated, become active, and are satisfied, suspended or abandoned, which can examine both cases as they are received, and which can make deliberate decisions about when to apply cases to the current reasoning state.

Finally, the case study taught important lessons about case-based planning:

- **deliberate control is necessary for multi-plan adaptation:**

The initial model of multi-plan adaptation, in which cases were treated as operators provide greater leaps through the search space, proved seriously flawed.

Treating cases like operators led to an exploded search frontier filled with unrefinable plans. The alternative metaplanning approach, which used the same underlying algorithms but exercised more deliberate control over what to merge, when to merge, and when to reintegrate the results of merging into the search space, suggested that deliberate control of planning at a high level was necessary to properly exploit multi-plan adaptation.

- **ordering and binding problems must be controlled when merging multiple plans:**

The original model also introduced additional binding and ordering threats into the partially-ordered plans for most domains, requiring extensive additional planning effort to repair which outweighed the benefit of retrieval for large cases. This suggested that the dynamic clipping and splicing model is more appropriate for partially-ordered planning domains without difficult-to-resolve binding or ordering constraints, or to state-space planners or other approaches which constrain the binding and ordering explosion through explicit representations of the state of the world.

In sum, the context-sensitive asynchronous memory approach proved sufficient to enable a system to retrieve and merge multiple cases to provide improved performance not possible with traditional methods, but in turn this improved method did not exploit the full benefit of the context-sensitive asynchronous memory approach.

## 8.14. Overcoming the Caveats

Because of the difficulties in the performance task which were by and large unrelated to context-sensitive asynchronous memory itself, this case study did not fully test the properties of context-sensitive asynchronous memory or its use in a performance task — it did not show a system that *simultaneously* provided feedback, used it to guide memory, and used the results to improve performance. Analysis of the results of the case study seemed to indicate that achieving this in the planning framework would be difficult.

No standardized planning domains existed that simultaneously contained problems difficult enough to make multi-plan adaptation useful, domain structure rich enough to make feedback from planning possible, and a sufficiently large body of problems which made context-sensitive asynchronous memory applicable. For example, the bulk of the planning domains provided with SPA produced plans that were not sufficiently complex enough to be usefully constructed out of coherent subplans, making multi-plan adaptation inapplicable. Other available domains contained many components and possibly could have served as a test of the MPA algorithm, but did not provide (or readily support the construction of) a library of problems large enough to be a useful test of context sensitive asynchronous memory.

Because no suitable benchmark domains existed, a novel case-base also have to be developed that was large enough to challenge a context-sensitive asynchronous memory, either for an existing domain or a new domain.<sup>20</sup>

Finally, the problems with the least-commitment planning approach identified would not necessarily go away with the choice of a new domain; fully addressing these issues in planning may therefore have required not only building a new domain and case base but perhaps a new planner as well.

Because of these challenges, I decided to apply the context-sensitive asynchronous memory approach to a new task that more fully exercise the approach's capabilities, while simultaneously demonstrating its generality across tasks. To test the approach, the target domain needed to challenge memory enough to make the approach applicable, provide feedback which memory could use to improve performance, and exploit improved retrievals to provide improved benefits to an overall task; to test generality it also needed to be sufficiently different from the planning task to require new representations and integration strategies.

---

<sup>20</sup> Around the time this work was being completed, a variety of domains were being developed for the AIPS-98 Planning Competition (Long 2000, McDermott 2000). Some of these domains, including some with many problems and large problems, could potentially serve as a testbed for future research.

The task I chose was information retrieval. The system I developed which demonstrated the capability of the context-sensitive asynchronous memory approach to exploit feedback to improve task performance was IRIA, described next.



# CHAPTER IX.

## CASE STUDY: INFORMATION RETRIEVAL

---

*Validating context-sensitive asynchronous memory's benefits for real tasks*

### 9.1. Overview

The second major test of the context-sensitive asynchronous memory approach's ability to provide benefit to real tasks was in information retrieval. In this case study, an information retrieval system was constructed on experience-based agent principles and tested against human judgements of relevance. This experience-based information retrieval system, Nicole-IRIA, was tested a variety of application areas in both rigorous experiments, user evaluations and exploratory evaluations, all designed to identify the benefits that context-sensitive asynchronous memory provided.

### 9.2. Goals of the Case Study.

As in the previous study, this case study was designed to investigate how context-sensitive asynchronous memory could be applied to reasoning tasks and to illuminate the benefits that it provided to those tasks. The specific thesis tested was:

Experience-based agents can retrieve information relevant to a task based on context derived from reasoning or the environment.
---

Again, the experience-based agent approach is simply a framework for constructing reasoning tasks that can exploit the properties of a context-sensitive asynchronous memory. Therefore, this study can be seen as testing the thesis that context-sensitive asynchronous memory can provide relevant information to a task based on contextual information.

Hand in hand with validating that experience-based agency works is unpacking how it can be applied to a particular task domain and what its strengths and limitations are. This case study explored in particular the implications of building a store of experience that could be accessed through context and what its strengths and weaknesses were for benefiting real tasks.

### **9.3. Methodology of Study**

Testing the thesis that experience-based agency can retrieve information relevant to a task requires applying it to a task and evaluating retrieved knowledge with respect to some metric of relevance.

I applied experience-based agency to information retrieval to investigate and evaluate its capacity to exploit contextual information to find useful information. An experience-based agent approach to information retrieval consists of building a store of knowledge about information resources and recommending information from that store based on context-driven reminding.

The performance of the approach was evaluated with respect to human judgments of information relevance, in both formal experimental settings and subjective user evaluations.

## **9.4. Execution of the Study.**

The testbed for these evaluations was the Nicole-IRIA system, an information management system built along experience-based agent principles and implemented in the Nicole architecture. Nicole-IRIA guides its presentation of information based on context derived from user behavior at an interface.

Nicole-IRIA was tested on real-world data sets drawn from Web search and employment search applications. I conducted a series of experiments using Nicole-IRIA on these domains, comparing the relevance of its presentation of information with human judgments of relevance, attempting to determine how well it used contextual information to improve its quality of presentation of information.

The study resulted in successes: in a variety of conditions Nicole-IRIA was able to use context to recommend useful information. Along the way, unexpected strengths and some weaknesses in the system were discovered that illuminated how the context-sensitive asynchronous memory approach can be most effectively applied.

## **9.5. The Task: Information Retrieval**

The specific target task of this study of experience-based information retrieval was relevance determination for browsing assistance.

Information retrieval is the task of finding some information in a knowledge base that satisfies a human user's information need. In traditional data retrieval approaches, the user's desire for information can be represented in a logical or numerical formula which specifies an exact set of items in a database. In contrast, in information retrieval a user's information need may not map directly to any single query specification and therefore items in the knowledge base may satisfy the query to varying degrees; this degree of satisfaction is relevance.

There are two primary tasks in information retrieval: retrieving and browsing. Retrieving takes as input a query and returns a set of results from a knowledge base relevant to the query. Browsing takes as input a document and a document collection and a user navigation action and returns a new document from the same collection.

The approach to information retrieval discussed in this chapter focuses on supporting browsing by finding information relevant to the user's current context. This approach in general uses existing information retrieval systems to handle the retrieval task, although it is not limited to doing so.

## **9.6. The Problem: Getting Good Answers to Bad Questions**

Information retrieval is a good choice for the experience-based agent approach for several reasons. First, traditional information retrieval approaches potentially yield large result sets which users have limited resources to navigate. Second, user browsing and feedback can provide additional information beyond the initial query. Third, relevance of documents from user interests is dependent upon the content of individual documents. Fourth, while both knowledge bases and information needs change over time, there are similarities between queries which may make past search effort relevant.

Taken together, these facts suggest that information retrieval would be a good target for the experience-based agent approach: it includes large bodies of information and limited resources to process them, generates contextual information during the reasoning process, contains contextual structure in the domain, and can potentially benefit from the discover of additional experiences during the process.

### **9.6.1. Modern Information Retrieval**

By almost any metric one cares to measure, the amount of information available to the average user has exploded in the past ten years. Resources as diverse as magazine articles, classic books, news wires, historical archives, intelligence data and personal biographies are now all readily available with just a few clicks, brought from hundreds of different kinds of systems to a single interface thanks to the power of the Internet and web browsers. Today's ordinary users and information workers now have a single point

of access to a vast array of heterogeneous information resources. There is good news and bad news in this picture.

The good news is that the web brings an entire world of information to an individual user. The Internet makes it possible to ask questions which could never before be answered by drawing information from a vast array of heterogeneous information resources to single point of access: the web browser. Using search engines, directories and other information navigation schemes, a user can get a mountain of information on a single topic.

The bad news is that the web brings an entire world of information to an individual user. The heterogeneous, aggregate, unedited nature of the Web means that a simple question can lead to a deluge of potentially useless answers. Search engines, directories and other information navigation schemes can deliver a mountain of chaff hiding only a few grains of wheat.

This need for improved tools to search the Web and other heterogeneous information resources is widely acknowledged. The Web is vast, containing approximately a billion pages at the time of this writing and growing. This continent of data is trackless, swamped by dead links and slow-loading pages that make direct browsing impractical or at least frustrating for most users (Kehoe et al. 1998). The search techniques developed so far to guide users across this desert have proved inadequate, leading to widely recognized exasperation (Lawrence & Giles 1999, Pitta 1999, Riggs 1999, Internet World

1998, McWilliams 1998). Even expert searchers are stumped two-thirds of the time trying to find information quickly, often taking a quarter of an hour to find useful information (Kehoe et al. 1998). One of the largest problems is the sheer volume of information available: even simple searches yield overwhelming numbers of results, and users often lack the knowledge they need to appropriately focus their queries upon the objects of their search — presuming, of course, that the structure of the index admits such a focusing.

In the end, users are left adrift on a sea of unorganized, and hence useless, data. But the typical user does not wander out into this trackless desert without a goal in mind. Users have immediate needs for information which motivate them to embark upon their journeys into the World Wide Web. Moreover, the Web has a structure, both intentionally imposed by local page authors and unintentionally emerging from similarities between pages, which reflects the content that it contains. And each search that a user conducts that successfully results in a new addition to their swollen bookmark files or an e-mailed pointer to a friend or co-worker builds up a library of experience about the content of the Web that is relevant to that user's interests, a library which helps the users improve their search over time.

In short, users search for information on the Web in a particular context, the Web itself is structured in such a way that information related to a particular context can be identified by its semantic similarity, and repeated searches of the Web can build a library

of experience that can guide future searches. Therefore, search for information on the Web is a natural application for the context-sensitive asynchronous memory approach.

### **9.6.2. Existing Systems and their Limits**

Traditional information research systems tend to focus on improving individual aspects of information retrieval on the web without considering the context of the user's actions or getting feedback from their task.

For example, web search engines attempt to improve on the speed and coverage of browsing by using a memory-based approach: a web search engine builds an index of a subset of the web, by crawling or other means, which users can query to quickly get answers. Unfortunately, by their very nature search engines must apply general algorithms to large amounts of unstructured data very quickly, and do not have the ability to refine an ambiguous query into the user's true information needs. As a result, the answers returned by search engines are often imprecise and swamped with irrelevant information. The tools available to refine lists of results are primitive, requiring users to manually specify categories or keywords. Search engines do not adapt as users digest the results of the search, rarely learn from the results of searches and maintain little or no memory of past searches (Etzioni 1997, Cheong 1996). Metasearch engines attempt to improve on the search engine model by sending a query to multiple search engines simultaneously. This strategy is designed to improve comprehensiveness but can result



in searches that are less precise, and still does not deal with resolving ambiguity, refining results, or learning from search.

To improve upon these approaches, Etzioni and others have suggested that next generation of internet search systems should “move up the information food chain” (Etzioni 1997) and collate and organize information at a higher level. Search engines like HotBot, Lycos and AltaVista are effectively *information herbivores*, browsing the web directly to build large, massive indices of pages optimized to answer queries and little else. As proposed by Etzioni, *information carnivores* prey upon information herbivores, digesting their output to produce more comprehensive responses to queries. Examples include systems such as ShopBot, Metacrawler (Etzioni 1997), and SavvySearch (Howe & Dreilinger 1997). However, most existing information carnivores do little to resolve ambiguity or deal with feedback from users, nor do they exploit the history of past queries.

Beyond the hierarchy of information herbivores and carnivores lies the possibility of harvesting information from the food chain and storing it to answer future questions or guide future search. The *information agriculture* approach attempts to make search more precise and relevant by building a map of available information based on actual user questions, responses from information sources, and user reactions to those responses, and using that map to guide the search, ranking and presentation of information to the user.

The information agriculture approach appears in various guises. Collaborative filtering systems (Varian 1999) collect preferences from populations of users and use it to organize and recommend information to individual users (Pazzani et al. 1996), but the structure of this knowledge is often hidden and opaque. Adaptive web sites collect information about web site usage patterns and use it to organize the structure of the site (Wexelblat & Maes 1997, Perkowitz & Etzioni 1998) but this information is aggregated across many users and may not reflect any one users' needs or interests. Web harvesting systems such as Harvest (Bowman et al. 1994) collect indexing information from a variety of sites and use it for both search and caching, but again are not optimized to deal with an individual user's needs or interests. Finally, machine learning techniques such as clustering and classification have been applied to databases of collected web statistics to improve the precision of searches, including as Direct Hit's 10 most visted pages (DirectHit 1999), Netscape's What's Related button (Lyons 1998), Alexa (Alexa 1999), and GROUPER (Zamir & Etzioni 1998); however, these techniques depend on the structure of the web or the preferences of millions of users without learning much about individual needs or interests and without providing users a way to search or inspect the knowledge these systems have gleaned.

### **9.6.3. Building on Existing Work**

The Nicole-IRIA system builds on much of the pioneering work in information retrieval on the Internet. Nicole-IRIA is an information harvesting search and browsing aid that builds on previous approaches by implementing them within the Nicole

framework and extending them with context-sensitive search, asynchronous retrieval and experience-based accumulation of knowledge.

For example, Nicole-IRIA's semantic map builds on existing approaches to cataloging information resources, such as web search indices like Alta Vista, Lycos and Web Crawler (Cheong 1996, Ardö & Lundberg 1998), web directories like Yahoo (Yahoo 1999), information resource mapping systems such as Harvest (Bowman et al. 1994) and search ontologies as used in Ariadne (Knoblock et al. 1997) and InfoSleuth (Jacobs & Shea 1996). The knowledge map goes beyond these approaches by unifying indexing with a writable information space in which a user or workgroup can annotate and update descriptions of known information resources.

Nicole-IRIA's asynchronous search extends the incremental web search approach pioneered by systems like the early versions of Lycos (Cheong 1996) by reifying user queries as explicit semantic knowledge objects; this supports the novel capability to retrieve "best-guess" retrieval of pages while the queries are being processed in the background, as well as allowing the explicit manipulation of query properties during incremental search.

Nicole-IRIA's context-sensitive web search extends the browsing aid approach pioneered by such systems as Fish (de Bra & Post 1994) and WebWatcher (Joachims et al. 1997) by using feedback from the user's browsing to aid ongoing asynchronous searches in addition to aiding browsing itself.

This use of feedback is similar to relevance filtering, which also use feedback from the user to modify the retrieval or presentation of documents (Robertson & Sparck Jones 1976, Sparck Jones 1979, Salton & Buckley 1990; also see Sparck Jones & Willet 1997 for an overview). However, the bulk of these systems focus on augmenting or updating queries to perform new searches from document retrieval systems, rather on actual ranking and filtering of documents that have already been retrieved.

Ostensive information retrieval approaches (Cambell & van Rijsbergen 1996) attempt to retrieve documents by example without a query, in which users select relevant sets of documents and the system attempts to present semantically similar approaches. Ostensive retrieval shares similarities with the Scatter/Gather approach (Cutting et al. 1992), discussed shortly. The WebCSCA system (Crestani & Lee 1999) applies spreading activation to the ostensive retrieval approach, taking as input a set of selected documents from and conducting a web spidering process guided by constrained spreading activation (CSA). The CSA approach uses inference rules and depth limits to bound the spread of activation, unlike the CDSA approach which uses uniform equations with self-limiting properties. Another difference between the two systems is that the Nicole-IRIA approach uses an explicit query to guide its initial selection of documents, rather than crawling the Web itself. Like the Nicole-IRIA approach, the WebSCSA approach applies similarity metrics to retrieved pages prior to presenting them to the user, although WebSCSA must download pages to compute its similarity metrics, whereas Nicole-IRIA relies on pure activation level to make recommendations.

The search mediation engine used by Nicole-IRIA applies a harvesting or metasearch approach similar to Harvest (Bowman et al. 1994) and the MetaCrawler (Etzioni 1997) but differs in that the rich structure of the knowledge map permits the direct browsing of the database of pages.

A variety of information retrieval approaches are based on networks that represent documents, words, terms, concepts, and their relationships (see a review in Turtle & Croft 1990; also see Santos et al. 2000, Williams & Pottenger 2000, Wu & Dai 2000). For example, Cohen & Kjeldsen (1987)'s GRANT system pioneered the use of spreading activation to retrieve documents in a semantic network. However, this network required a great deal of domain representation and was painstakingly constructed over four person-months. In contrast, Nicole-IRIA's relatively simple ontology of information resources and connecting concepts was developed in a few weeks and enables semantic maps to be generated on the fly. Other approaches to information retrieval that share similarities to the Nicole-IRIA approach include approaches based on spreading activation such as (Salton & Buckley 1988 and Crestani & Lee 1999) and approaches based on building adaptive, learning maps of information resources as discussed in (Heylighen & Bollen 1996 and Heylighen 1999).

Another relevant approach is the "information scent" approach (Pirolli & Card 1995, 1999, Pirolli 1997). This work applies the theory of information foraging (Pirolli & Card 1999) to the Scatter/Gather browser interface developed by Matti Hearst at Xerox Parc (Cutting et al. 1992), which provides a clustered view of large document collections

which can be dynamically rearranged based on user inputs. The Scatter/Gather interface presents clusters of documents in a collection as a set of *meta-documents* summarizing each collection; at this interface users can then gather clusters they feel are relevant which the system then scatters into new clusters for more finely-grained inspection. To analyze and predict user behavior at this interface, Pirolli and Card applied a variant of John Anderson's ACT-R system called ACT-IF, which used spreading activation to model the "information scent value" of stimuli and which it used to then predict what information foraging actions a user would take. This model can be seen as logically complementary to the Nicole-IRIA approach: where the information scent work uses spreading activation to attempt to predict user behavior, Nicole-IRIA uses spreading activation to predict what information will be relevant to a user.

Information foraging was also used as a motivation for a spreading activation approach to organizing the presentation of pages on a web site (Pirolli et al. 1996). In this approach, a semantic network was constructed from the hyperlink topology of a web site, augmented with additional links that summarized interdocument similarity. This network was then weighted by the history of users navigating the site over time. When a user navigates to a page on the site, activation is spread into the network to recommend pages that are similar to the current page. This approach differs from the Nicole-IRIA approach in several ways. Nicole-IRIA focuses on the general problem of recommending information from a variety of data sources, rather than organizing the presentation of information within a web site; for this reason Nicole-IRIA's semantic network is

dynamically and incrementally constructed from pages and/or page summaries, and is not tied to the topology of a particular site. Furthermore, Nicole-IRIA does not use interdocument similarity or user navigation history to weight its network; instead, Nicole-IRIA represents the connections between documents, titles, words and so forth explicitly. This structured representation enables Nicole-IRIA maintain a richer representation of user interests and context, which in turn enables Nicole-IRIA to spread activation from sources other than the web page that the user is currently viewing. Despite these differences, it is possible that these two approaches could be combined in some way to improve the performance of either approach.

## **9.7. The Solution: Experience-Based Information Retrieval**

The experience-based agent approach to information retrieval is information agriculture on steroids. Experience-based information retrieval models recommendation as context-based reminding from information stored as experience.

### **9.7.1. Relevance as Reminding in Information Agriculture**

The general experience-based agent approach to a task is based on collecting experience and exploiting it in context: build a store of experience that reflects the semantic structure of the environment, and then use the task and environmental context to drive the context-sensitive asynchronous retrieval of information from that store likely to be relevant to the context.

Applied to information retrieval, the experience-based agent approach naturally suggests an information agriculture strategy: build *semantic maps* of information resources based on the structure of the Web and the history of user searches, and then use the user's current search and browsing behavior to drive the context-sensitive asynchronous presentation of information from the map relevant to the user's current needs.

**Building Semantic Maps.** An experience store can be built statically over a fixed collection or dynamically in response to user queries, harvesting information from a larger non-experience-based store of documents. The latter approach is particularly important for the Web: because the Web is far larger than any individual human or machine's ability to process, an information agriculture approach is crucial for dealing with its vastness.

The Web's sheer bulk, as well as the hordes of users that may swarm any individual site, limit the amount of intelligence traditional information retrieval methods can apply to any one request. In contrast, the experience-based version of the information agriculture approach piggybacks on top of traditional "quick and dirty" information retrieval methods, passing queries to traditional search engines and loading the results dynamically into the experience store. This approach allows the agent to focus its intelligence on the specific questions being asked by its users, and limits the size of its knowledge base to a scale manageable by human memory algorithms by focusing it on information related to the history of a user's search.



**Reminding From a Semantic Map Based on User Action.** User navigation actions as they browse search results can be modeled as a process driven by a user's information need and their perception of how relevant actions will help them satisfy that need. Relevance of documents can be modeled as a function of how their content relates to the content of an information need. To apply the experience-based agent approach, documents to be retrieved must be stored in the experience store labeled by content extracted from those documents. Some subset of documents is presented to the user along with a set of available actions (browsing, thumbs up/thumbs down recommendations, and so forth).

Actions are also modeled based on the relationship to these resources and the action's intention. When users select an action, the content of that action can be used to build up a context of the user's likes and dislikes. Browsing to a resource provides evidence that the user has interests in the resource's topic; browsing back or explicitly rejecting a resource from a list provides evidence that the topic is not of interest; combining this information together into a model of the user's interests enables spreading activation to generate reminders of relevant information from the experience store.

### **9.7.2. The Information Retrieval Intelligent Assistant**

To test this model of how the experience-based agent architecture could be applied to information retrieval, I designed a context-sensitive, experience-based approach to information retrieval problems called the Information Retrieval Intelligent Assistant, or

IRIA. To test this approach, and by extension test the context-sensitive asynchronous memory and experience-based agent approaches to constructing artificial intelligence systems, the IRIA architecture was implemented in a system called Nicole-IRIA.<sup>21</sup>

The core of Nicole-IRIA is a *reminding engine* consisting of a context-sensitive *search mediator* which uses a unified semantic knowledge base called a *knowledge map* to represent indexed pages, queries, and even browsing sessions in a single format. This uniform representation enables the development of an experience-based map of available information resources, along with judgments about their relevance, allowing precise searches based on the history of research for an individual, group or online community. The knowledge map is furthermore a browsable information resource in its own right, accessible by standard internetworking protocols; with appropriate security precautions, this enables workgroups at remote sites to view and exploit information collected by another workgroup.

The reminding engine is the core component of IRIA that makes this understanding possible. The reminding engine directly applies the experience-based agent approach to the problem of information search, consisting of a search engine query system, a unified

---

<sup>21</sup> Nicole-IRIA was developed at Enkia Corporation as part of an Air Force Rome Labs SBIR project, with Air Force sponsor Ray Liuzzi, principal investigator Anthony Francis, and research scientists Mark Devaney and Ashwin Ram. Patents have been applied for both the recommendation engine embodied in Nicole-IRIA and for the context-sensitive asynchronous memory system embodied in Nicole itself.

semantic knowledge base called a knowledge map that represents indexed pages, queries, and even browsing sessions in a single format, and a context-sensitive search mediator that retrieves and organizes information in the map based on user interests. This uniform representation enables the development of an experience-based map of available information resources, along with judgments about their relevance, allowing precise searches based on the history of research for an individual, group or online community. The knowledge map is furthermore a browsable information resource in its own right, accessible by standard internetworking protocols; with appropriate security precautions, this enables workgroups at remote sites to view and exploit information collected by another workgroup.

The ultimate goal of this technology is an intelligent information management toolkit, deployable in a wide variety of environments and configurations, which can proactively provide a working user with useful information by searching its knowledge map in parallel with user action, by actively querying multiple information resources and collating their results, and by capturing the history of these searches in the knowledge map that becomes an information resource in its own right. The user will be able to interact directly with the knowledge map to browse and rapidly locate relevant information, or to use it as an intelligent assistant working unobtrusively in the background.

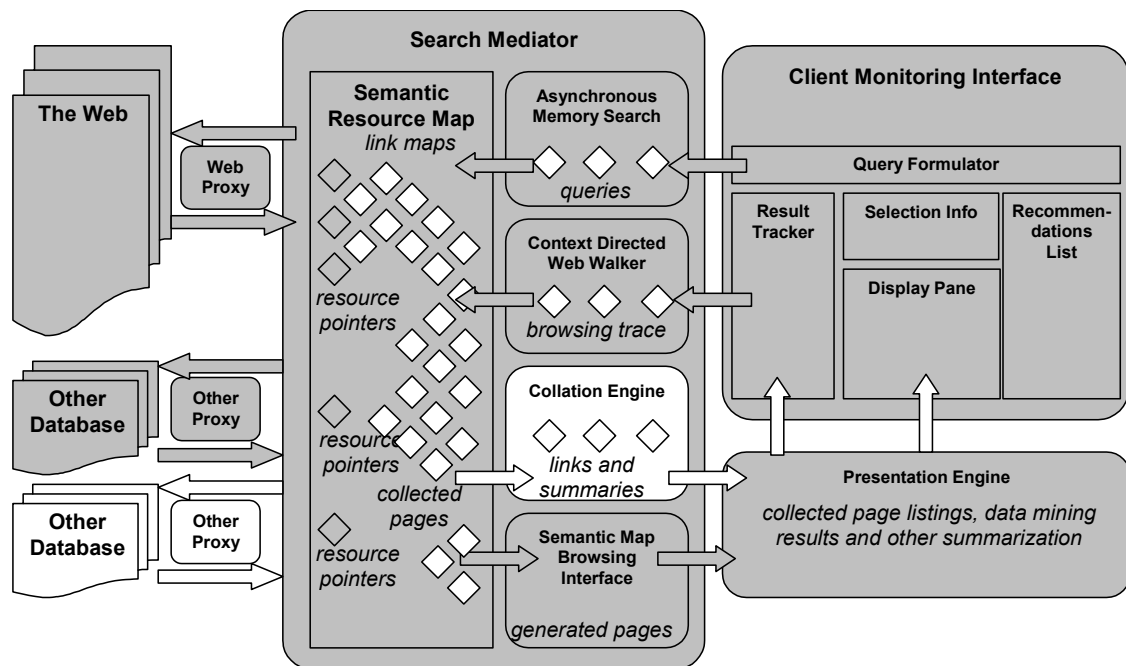


Figure 9.1. Architecture of Nicole-IRIA

### 9.7.3. Architecture of Nicole-IRIA

The Nicole-IRIA system is composed of a set of functional components which work together to assist a user or group's web research. This set of components, illustrated in Figure 9.1, includes:

- **Knowledge map:**

A unified semantic knowledge base for describing, summarizing, and categorizing available information resources based on task relevance, availability, format, and content. Also known as a "semantic map" or "semantic resource map", the knowledge map is an innovative approach to cataloging information resources which provides the benefits of many traditional approaches such as web search

indices, web directories, and information resource mapping systems, yet goes beyond them to provide a writable information space in which a researcher or workgroup can annotate and update descriptions of known information resources.

- **Asynchronous memory:**

"Fire and forget" search of information resources based on user query. This is responsible for maintaining a list of pending queries (and standing requests for information) and alerting the user when useful information is found. Asynchronous search extends existing incremental web search approaches by reifying user queries as explicit semantic knowledge objects; this supports the novel capability to retrieve "best-guess" retrieval of pages while the queries are being processed in the background, as well as allowing the explicit manipulation of query properties during incremental search.

- **Context-sensitive memory:**

Working hand in hand with the asynchronous memory is the context-sensitive search system, which uses a trace of user actions at the interface to attempt to identify the context of a users search and direct search of the semantic network accordingly. Context-sensitive web search is a browsing aid approach that uses relevance feedback from the user's browsing to aid ongoing asynchronous searches in addition to aiding browsing itself.

- **Search mediation engine:**

The knowledge map, asynchronous search and context-sensitive search modules

must be embedded in an overall architecture which can accept queries from the user on the one hand and access information resources on the other. The search mediation engine is a server-side architecture for accessing, integrating, summarizing, and indexing data from large-scale heterogeneous information resources which uses the knowledge map and asynchronous search algorithms as its core. The search mediation engine goes beyond existing harvesting or metasearch approach in that the rich structure of the knowledge map permits the direct browsing of the database of pages.

- **Monitoring search interface:**

To take maximum advantage of the context sensitive search process, the user interface to the search system must be instrumented to permit the system to unobtrusively monitor the user's behavior while searching an information resource.

- **Intelligent collation and ranking:**

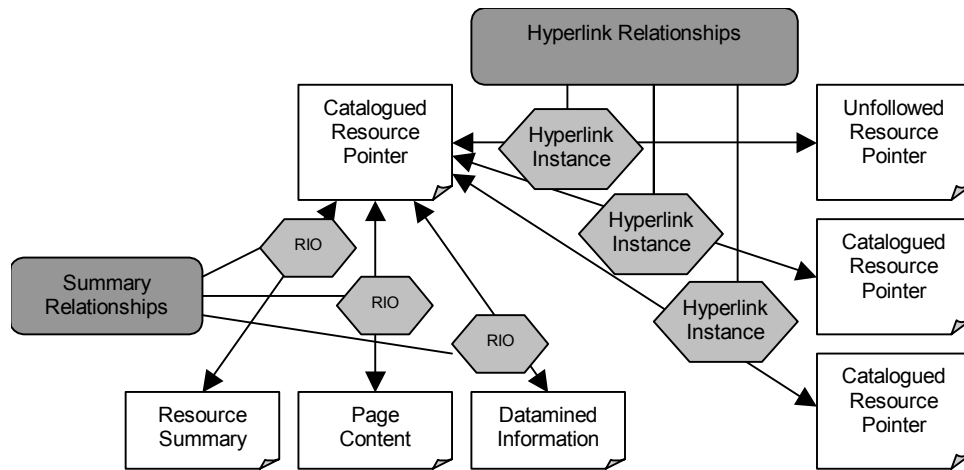
This component must collect and analyze retrieved data sources to identify relevant clusters of information, to rank those groups for presentation to the user, and to extract summary information when possible. This includes both natural language tools and data mining facilities.

- **Information presentation engine:**

This uses techniques for displaying multimedia search results and summaries. Once data has been collected, it must be presented to the user in a way that

communicates information clearly and effectively. The information presentation engine will also enable users to browse the knowledge map as an information resource in its own right.

The initial prototyping effort focused on the search mediation engine, semantic map, and context-sensitive asynchronous search system, along with limited prototypes of other components, such as the monitoring search interface, as were needed to demonstrate the feasibility of the design. Future efforts already underway at Enkia Corporation will explore expanding these capabilities and adding new ones, such as more sophisticated collation and presentation systems.



**Figure 9.2. Representation of Information Resources in an Experience Store**

#### 9.7.4. Details of the Approach

Experience-based agency brings the knowledge map, asynchronous search and context-sensitive search to the information retrieval problem. Each of these components brings unique capabilities to information retrieval systems.

**Knowledge Map.** An experience store is a natural representation for the knowledge map. Information resources, be they single web pages or entire databases, can be represented by nodes in the experience store, and links between resources can be represented by relationships between nodes. As additional information about a resource is discovered, such as the text of a page retrieved by HTTP, metadata about the content and format of a database extracted through knowledge discovery, or annotations about a resource entered by users of the system, it can be added incrementally to a node in the form of links to subsidiary concepts and data groundings. Figure 9.2 illustrates this



general idea, although several variants of this approach have been instantiated and tested during the course of the case study.

Figure 9.3 illustrates a subset a knowledge map used in the experiments detailed in this section. In this variant, the structure of the knowledge map labels information resources using a subset of the Dublin Core metadata (Weibel et al. 2000, Dublin Core Metadata Initiative 2000) structured into a graph which links words, titles and summaries, pages and links into a common format. In the figure, nodes are displayed on the right as boxes and categories displayed on the left as rounded boxes. IS-A links between them are suppressed. Other link types are indicated by reified relation objects depicted as hexagons. Note that an individual resource's connection to words is segregated into separate link types for title words, summary words, and so on, to enable the application of the CDSA gated spreading activation process.

Because the experience store represents both the map of resources and the knowledge derived from those resources in a graph structure, the experience store itself can be browsed like any other hypertext data structure. A user (or, with appropriate security, a workgroup) can thus explore the past history of information research with Nicole-IRIA, perform additional analyses or data mining, annotate information resources, edit the descriptions of resources that have been modified, or delete resources that are no longer needed. The search mediator must thus act as not only a gateway for requests but also as a server of information.

Layered atop the knowledge map is the context-sensitive asynchronous search system which searches the existing semantic map and newly harvested information from the Internet in a unified interface, using feedback from the user to provide context-sensitive recommendations. The system is accessed via an integrated interface that combines search and recommendations functions with a knowledge browser that allows users to inspect and share the knowledge map.

**Asynchronous Retrieval.** A user's need for information may be pressing, yet accurate and comprehensive search of a large knowledge base can take a long time. Asynchronous retrieval enables a system to satisfy a user's urgent need for information without sacrificing comprehensiveness by returning a "first guess" set of information resources within a specified deadline for user inspection while continuing to search the knowledge base more comprehensively in the background, returning additional results as they are found.

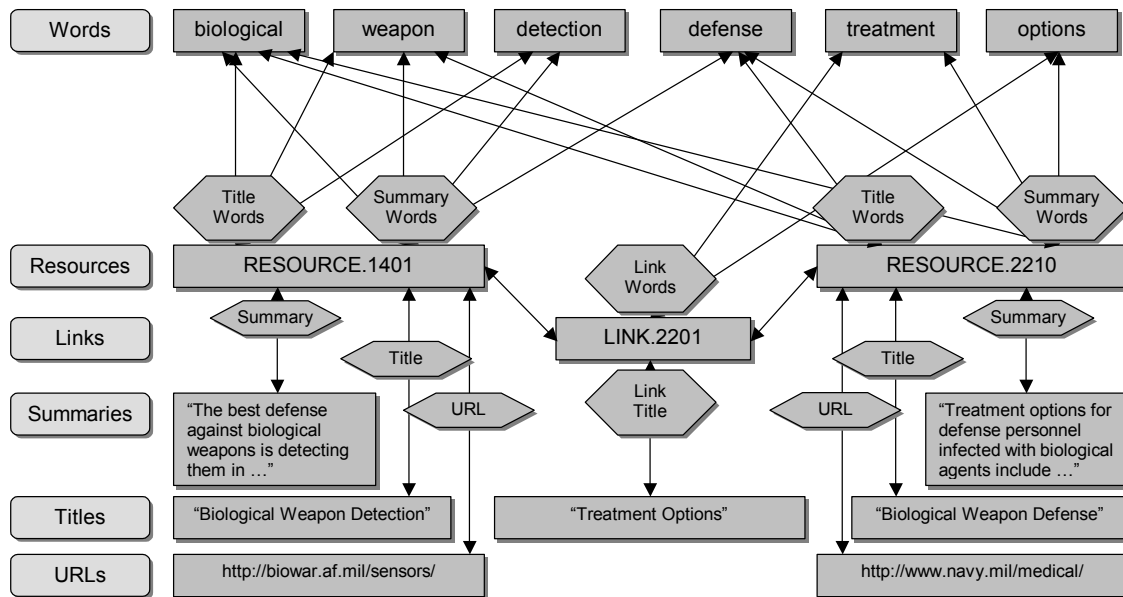
**Context-Sensitive Search.** Embedded within the search mediator, the context-directed spreading activation algorithm provides a natural algorithm for searching the World Wide Web, either in response to user queries or in an autonomous information harvesting mode. The context of a user's browsing activity can be used to guide search through the knowledge map for appropriate information resources; when relevant resources require additional processing in response to a query (such as links to pages which have not yet been retrieved, or pointers to database gateways which must be queried) the activation levels of retrieved nodes provide guidance to the search mediator

about how to make use of its available search and analysis resources. In addition, the history of search of a user or workgroup can serve to guide an ongoing, autonomous search for frequently used pages which can be cached locally.

#### **9.7.5. Structure of an IRIA-Based Application.**

A typical Nicole-IRIA system is deployed as an information management extension to an existing interface. For example, Nicole-IRIA could be deployed as a search engine assistance application which displays search results on the left and the user's current selected result in the center. As the user browses, Nicole-IRIA is reminded of pages and displays these dynamically computed results on the right, enabling users to quickly focus on relevant results (Figure 9.4).

In the figure, a user has initiated a search for "biological warfare" information on the Internet. On the left hand side of the interface, the original search results are displayed, including information on how to deal with biological warfare threats ("1. Information Paper: DoD Biological Warfare Threat Analysis"), information on how to treat people infected by biowarfare agents ("10. Medical Defense Against Biological Warfare Agents") and even political articles on biological warfare ("3. Chemical and Biological Warfare Unmasked" and "7. Biological Warfare - The Poor Man's Atom Bomb").



**Figure 9.3. Portion of a Typical IRIA Knowledge Map**

The user, however, is interested only in dealing with biological warfare agents and has clicked on several links including the first link, “Information Paper: DoD Biological Warfare Threat Analysis”. Using these click as evidence of the user’s interests, Nicole-IRIA is reminded of relevant links through the context sensitive asynchronous memory process and recommends several links to the user on the right hand side. Out of the nine links recommended, 7 are directly relevant to military assessment of biological warfare threats, including links on how to protect troops like “Overview: DoD Biological Warfare Defense Immunization Program” and links on how to detect threats like “8. NIST: Sensing chemical or biological warfare agents”. In effect, Nicole-IRIA used the user’s information to find links dozens or hundreds of items down on the search engine list and to recommend them directly to the user with a minimum of search.

Layered atop the knowledge map is the context-sensitive asynchronous search system which searches the existing semantic map and newly harvested information from the Internet in a unified interface, using feedback from the user to provide context-sensitive recommendations. The system is accessed via an integrated interface that combines search and recommendations functions with a knowledge browser that allows users to inspect and share the knowledge map.

## **9.8. Executing the Study**

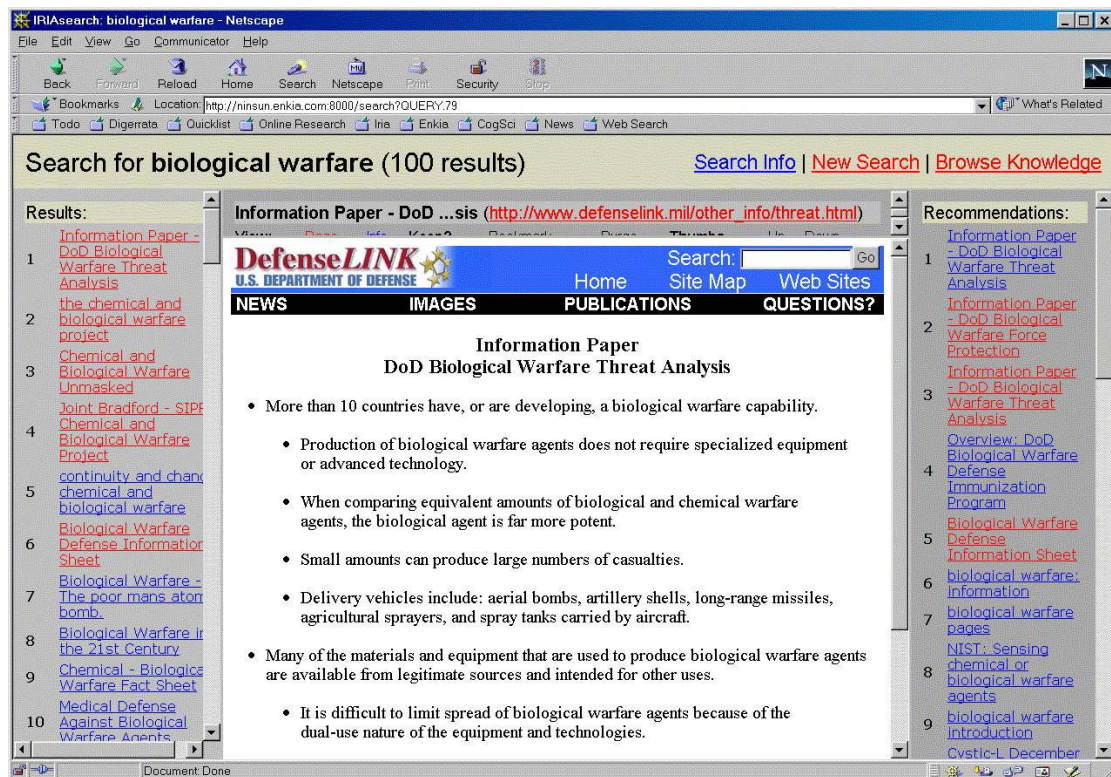
To conduct this study, I developed a prototype of the experience-based agent approach to information retrieval in a system called Nicole-IRIA. Nicole-IRIA was used to develop several applications, and several data sets were collected and categorized to conduct experiments on these applications' performance.

### **9.8.1. Applications Constructed**

The applications constructed included:

- **Nicole-IRIA Server-Side Prototype.**

A prototype of the Nicole-IRIA system was constructed at Enkia Corporation as part of an Air Force Phase I SBIR project (Francis et al. 2000a,b). The Nicole-IRIA prototype is a web server plug-in accessed through browser-based and standalone interfaces. The prototype was developed primarily in Common Lisp using the CL-HTTP web server and Nicole AI toolkit, with Perl extensions. The prototype uses a metasearch system to execute a query



**Figure 9.4. IRIA: The Phase I Prototype of the IIM Toolkit**

on existing WWW search engines (e.g., Altavista, Yahoo, etc.) and then summarizes the returned hits into a knowledge map.<sup>22</sup>

- **Web (Re)Search Application:**

A prototype client/server web search assistance and research application was constructed using Nicole-IRIA. This application accesses the core IRIA server capabilities through a standard browser interface. Figure 9.4 illustrates this prototype.

---

<sup>22</sup> Thanks to Mark Devaney for programming the Perl harvester interface which plugged into the Lisp IRIA core I developed using Nicole.

- **Computer-Aided Problem Based Learning Application:**

The Nicole-IRIA prototype engine was also used to construct an educational whiteboard system called CAPABLE (P. Ram et al. 2000). This application accessed the original and additional IRIA server-side functions through a Java client interface.<sup>23</sup>

- **Other Demonstrations:**

The IRIA engine was used to construct a number of other proprietary demonstrations. Furthermore, in a follow-on Phase II SBIR project, the Nicole-IRIA engine is being regenerated as a commercial-grade product. While the behavior of the commercial product is equivalent or superior to that of the Nicole-IRIA prototype, the empirical tests discussed in this chapter were conducted on the prototype.

### **9.8.2. Datasets Constructed**

The datasets collected included:

- **Dolphin Data Set:**

The “dolphin” data set was a typical web search result set, consisting of 100 search results drawn from the AltaVista search service (Ray et al. 1998, AltaVista 2000). Each web page in the set was grouped into three separate

---

<sup>23</sup> Thanks are due again to Mark Devaney for constructing the Java client-side interface which provided an alternative way to access and test the Lisp IRIA core.

categories: dolphin vacations, marine biology, and Miami dolphins, plus a small number of miscellaneous results.

- **Employment (Dice) Data Set:**

The “dice” data set was drawn from a search on programmers on the Dice employment web site. This consisted of 100 programmer results, separated into 4 different categories.

- **Content Provider Site Data Set:**

An additional data set of 3000 items was collected with permission from a large regional content provider’s web site as a means of testing the system’s performance on larger data sets.

- **Educational Data Set:**

A data set of educational resources was constructed based on the knowledge base of the Virtual Sherlock problem based learning program (P. Ram 2000) as a means for testing the system’s recommendations of non-web related information.

These applications and data sets were used to test Nicole-IRIA’s performance, both experimentally and in a series of exploratory surveys.



## 9.9. Fidelity of the Study

Nicole-IRIA had its limits. It made limited use of the task control system, instead invoking execution cycles of the Nicole architecture in response to user actions at an interface. It furthermore did not use automatic cueing of spreading activation, also driving this in response to user actions at the interface. As a consequence, while processing of retrieval requests and reminding was asynchronous and incremental and continued over the course of many user actions, the implemented prototype only partially exploited the additional resources that were available during pauses between user actions.

Second, as an experience-based agent Nicole-IRIA's integration mechanisms were trivial. Because each retrieved resource was atomic and search results were simply ranked sets, the act of integration simply consisted of retrieval evaluation based on the activation level of a retrieved resource (a measure of relevance to current user interests) and based on the specifications of the user query (necessary to eliminate resources which were active but did not contain terms relevant to a the user's query).

Finally, Nicole-IRIA relied heavily on its harvesting infrastructure to provide an initial list of results drawn from an outside database; even when its internal knowledge base was well stocked it performed little in-memory "harvest". This was a deliberate adherence to the experience-based agent approach at the expense of the information retrieval task in the following sense. Nicole-IRIA retrieved information from its own knowledge map only in response to its current understanding of the user's interests; as

Nicole-IRIA's understanding of the users' information need evolved, additional results were often found and retrieved asynchronously. While this fit the experience-based agent model, in practice it meant that Nicole-IRIA would often ignore some resources in the knowledge map that would have been retrieved by an exhaustive boolean or vector matching process until the user's browsing actions provided cues to Nicole-IRIA to guide it to retrieve those items.

## 9.10. Hypotheses Tested

The goal of these evaluations was to show that context-sensitive asynchronous memory could provide useful information to reasoning tasks based on contextual information. This overall goal was unpacked into a series of hypotheses about the potential benefits of context-sensitive asynchronous memory to the planning process:

- **Context-sensitive asynchronous memory can recommend useful information**

Context-sensitive asynchronous memory can retrieve information relevant to reasoning tasks as judged by human users.

- **Feedback can improve context-sensitive asynchronous retrieval**

Contextual feedback from task performance can improve the relevance of context-sensitive asynchronous memory retrievals at providing relevant information.

- **Asynchronous retrieval is useful**

Asynchronous retrieval of information can enable a context-sensitive asynchronous memory to incrementally retrieve additional useful information.

- **Generality of the context-sensitive asynchronous memory approach:**

Context-sensitive asynchronous memory can recommend useful information for a variety of tasks.

The experimental evaluation of these claims by and large validated these claims and revealed fortuitous findings along the way.

## 9.11. Results

I used the Nicole-IRIA system to conduct several evaluations, analyzing the power of the context-sensitive asynchronous memory system to recommend useful information to users and workgroups, its ability to support applications other than web search, and the contributions of the context-sensitive asynchronous memory parameterization to retrieval quality. These evaluations included:

- **Evaluation 9.1. Providing Useful Answers to Individual Users**
- **Evaluation 9.2. Providing Useful Answers to Workgroups.**
- **Evaluation 9.3. Support for Educational Applications.**
- **Evaluation 9.4. Support for Other Applications.**
- **Evaluation 9.5. Analysis of Serendipitous Results.**

### 9.11.1. Evaluation 9.1. Providing Useful Answers to Individual Users

Quantitative evaluations on two actual search data sets were conducted to evaluate the prototype and collect feasibility results. These experiments were designed to test how good Nicole-IRIA’s context-sensitive search was at recommending useful information to users.

**Evaluation.** The model behind these experiments was of a user with a specific need for information who enters an ambiguous query — for example, a researcher on dolphin cognition simply searching for “dolphin” — who then receives an overwhelming number of results. Confronted with this list, the user then begins browsing through the result lists, seeking relevant results. The hypothesis of these experiments was that if a user selected initial pages relevant to their desired category, Nicole-IRIA would be able to provide other pages relevant to that category and speed the user’s search. Thus, the number of relevant pages recommended in, say, Nicole-IRIA’s Top 10 or Top 20 list should be significantly better than chance.

**Method.** The user’s task in these experiments was web search: finding web pages relevant to the query. We assume that the simulated user has an information need not explicitly stated in the query, assume that returned web search results can be categorized into several major semantic groupings, and that one or more of these groupings are relevant to the user’s search.

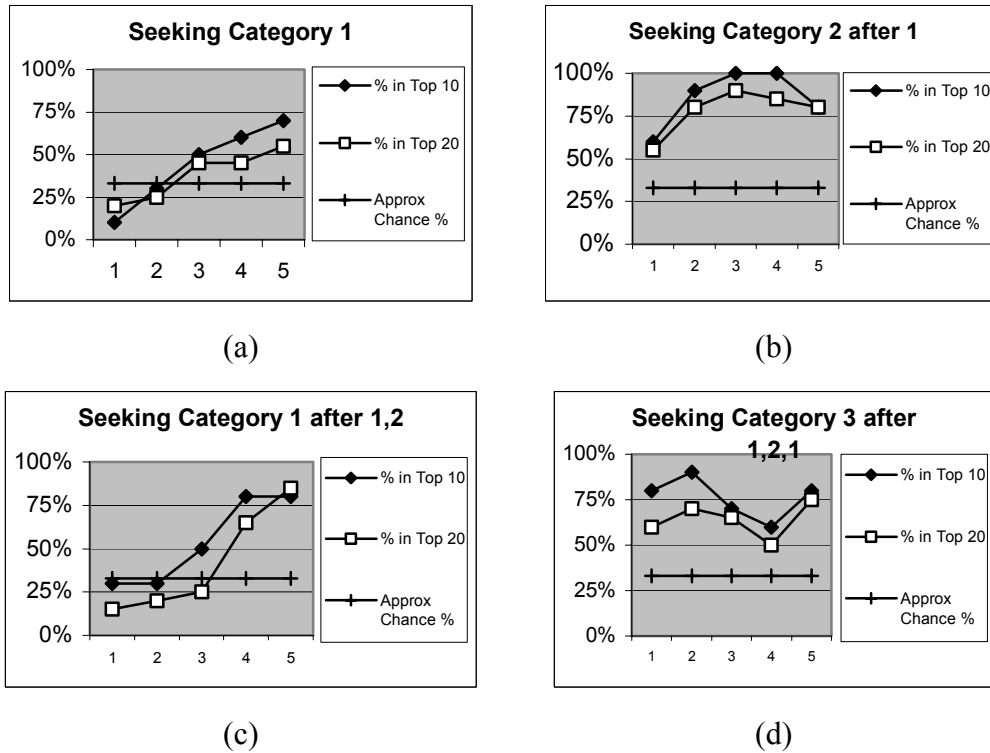
The system's task was to focus its presentation of information to the user on categories relevant to the user's hidden information need, using feedback from browsing behavior to perform the search. The dependent variable was percentage of relevant results displayed in the Top 10 and Top 20 lists; therefore, the system's task was to maximize the percentage of relevant results.

The baseline for this task is a random presentation of results; on average the percentage of relevant results displayed in the Top N list would be the average category size divided by the total number of results returned. The hypothesis tested was that Nicole-IRIA could exceed this chance benchmark in a small number of browsing events or clicks.

In this evaluation, two data sets were conducted for controlled offline search, including one generic search from AltaVista on "dolphin" and one special-purpose search of job postings from the dice.com database to show feasibility on an e-recruiting application. For each data set, all the results were hand-categorized by a judge into logical groupings — for example, "dolphin vacation pages", "dolphin research sites", and "Miami Dolphins. These groupings were not provided to the system, but were instead used to gauge the quality of the system's results. The hypothesis of the experiment was that if a user selected pages relevant to, say, dolphin vacations, Nicole-IRIA would be able to quickly provide recommendations of other relevant pages, thus reducing the need for the user to wade through overwhelming numbers of results.

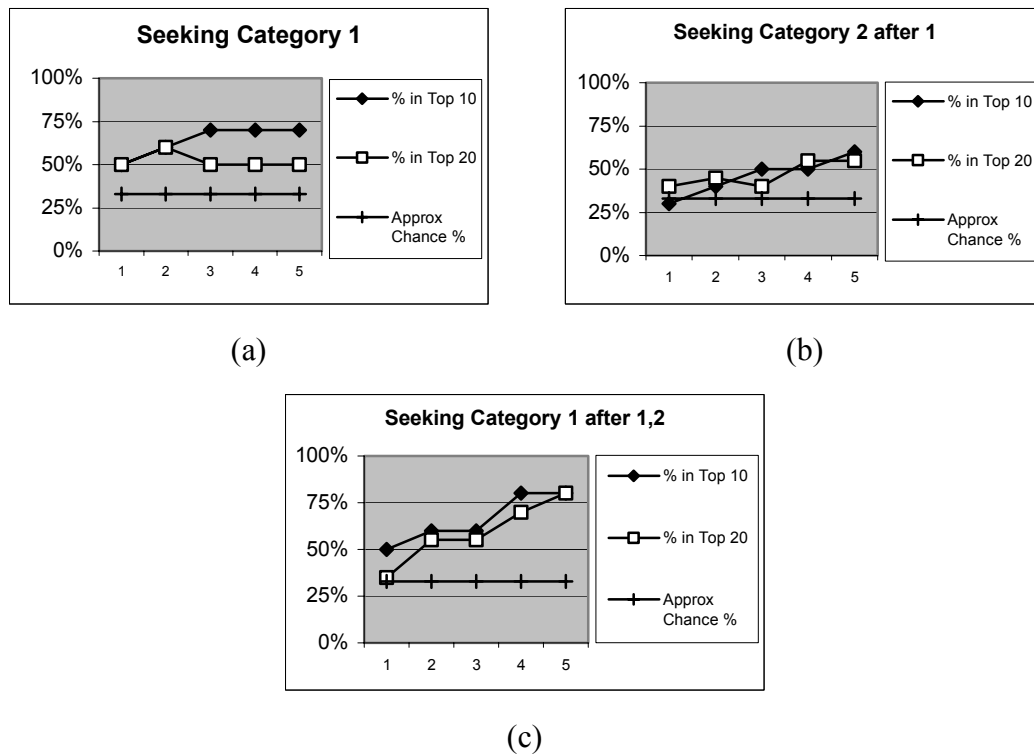
For the actual experiment, a test user distinct from the judge who grouped the pages entered the search term, and an experimental harness captured the queries and returned data from the pre-categorized test sets. The categories, known to the evaluator but unknown to Nicole-IRIA, were used to simulate a naturalistic browsing sequence: first the test user searched for results in one category, then simulated becoming distracted by searching for results in another category. Finally, the test user returned to the first category to continue the search, and then moved on to a third category.

At each step during the user's browsing experience, the Nicole-IRIA prototype recommended 30 sites considered relevant according to the context-sensitive search algorithm. To evaluate Nicole-IRIA's performance, the proportion of results out of the Top 10 and Top 20 recommendations that were drawn from the same category that the test user was seeking were measured. On both data sets, the proportion of relevant results produced by a random recommendation algorithm should be approximately 33% (assuming 3 categories).



**Figure 9.5. Quality of Recommendations on the “dolphin” Search Data Set**

**Results.** On both data sets and in all three conditions, Nicole-IRIA presented a proportion of relevant results relevant greater than chance after only one to three selections by the user. In certain conditions on the dolphin data set, Nicole-IRIA produced 90-100% relevant recommendations; on the tougher dice.com data set Nicole-IRIA was still able to produce 60%-80% relevant results within a few user clicks. Results from the “dolphin” data set are displayed in Figure 9.5. Results from the “dice” data set are displayed in Figure 9.6.



**Figure 9.6. Quality of Recommendations on the “dice” Search Data Set**

### 9.11.2. Evaluation 9.2. Providing Useful Answers to Workgroups.

A qualitative evaluation was conducted to investigate how Nicole-IRIA could support a workgroup. This evaluation was designed to test whether Nicole-IRIA’s context-sensitive search could recommend information to groups of users that no single user could have found alone.

**Evaluation.** The model behind these experiments was of a workgroup with a joint need for information that divided research efforts on that need between multiple users — for example, an intelligence taskforce searching on information about a particular conflict which divided its efforts between individuals studying countries involved in the region,



major political issues, geographical and military concerns, and so on. The model was that individual users would enter queries at the Nicole-IRIA search interface as in the previous experiment, but would share the same knowledge map and would both contribute to the context.

There were two hypotheses in these experiments. First, I hypothesized that the context developed by multiple users would contribute to recommending information no single user would have found alone through the context-sensitive search process. Second, I hypothesized that a shared knowledge map would enable serendipitous resource findings in which a search conducted by one user would contribute results to a search by another user through the asynchronous retrieval process.

**Method.** In this evaluation, a workgroup situation was simulated by having two independent users seated at different workstations searching for different types of information on the web, sharing a single knowledge map containing contributions from both user's searches. The primary example tested was of a workgroup performing a search on Middle East conflicts. One user searched for and browsed through information on the Middle East. A second user searched for and browsing through information on biological warfare. As a baseline, both users performed individual searches and tracked the sets of relevant recommendations the system provided as well as the total number of resources found by Nicole-IRIA. After the baseline tests, the system was reinitialized and the two users conducted their search simultaneously over the same knowledge map

and sharing the same context, noting the sets of recommendations and number of resources found to detect any differences.

**Results.** As a result of the two users collaborating on search, Nicole-IRIA found and recommended resources relevant to the workgroup's information need that no single user found on their own. One example from the Middle East conflict example was a page from a political analyst discussing whether Iraq might still have biological weapons, a page relevant to both searches which was not recommended to either user in the standalone condition.

As a result of the two users collaborating on search, Nicole-IRIA found and recommended resources relevant to the workgroup's information need that no single user found on their own. One example from the Middle East conflict example was a page from a political analyst discussing whether Iraq might still have biological weapons, a page relevant to both searches which was not recommended to either user in the standalone condition.

Furthermore, Nicole-IRIA's asynchronous search system detected results from each user's search that were relevant to the other user's searches, updating its presentation of results for each user accordingly. This had the effect of adding between 5 and 10 results to each search.

In sum, in the workgroup condition, Nicole-IRIA was able to recommend information that no single user found alone, and was able to find additional results beyond what it found for each user on their own.

**Discussion.** This evaluation was exploratory in nature, relying on kludges in the system's server to simulate a workgroup experiment. More extensive experiments would require the extending the core knowledge map and context sensitive search system, an effort which is currently underway.

However, this did show the feasibility of the idea of sharing a context between users in a workgroup. Furthermore, Nicole-IRIA's knowledge map also contained information about why this page was recommended; this can be used in future versions to provide users with an explanation of the system's reasoning if desired.

### **9.11.3. Evaluation 9.3. Support for Educational Applications.**

The CAPABLE project (P. Ram et al. 2000, Francis et al. 2000c) tested the Nicole-IRIA system's ability to support a pedagogical approach called *problem based learning* (Barrows 1988, P. Ram 1999) through its ability to help students do self-directed research.

The CAPABLE system contained a knowledge base which was seeded with information from an extended science lesson in forensic chemistry, and provided an "Intelligent Whiteboard" interface into which students could enter their thoughts about

the problems they faced. The system's task was to use the notes, ideas and hypotheses about the case that students entered into the Intelligent Whiteboard to recommend information relevant to the student's current thoughts, and to collate the information entered into the Whiteboard into a persistent "Virtual Notebook" that students could return to later.

In the initial stage of using the system, recommendations are drawn from a seed knowledge base containing relevant information selected by the teacher as part of the lesson plan, designed to help students formulate their thoughts without biasing them towards one or more hypotheses. As students progressed from recording facts and ideas onto the Whiteboard to generating hypotheses, the CAPABLE system extended its search to the Web to help the students find additional resources to aid their understanding. The CAPABLE system also allowed the students to bookmark relevant resources in the Virtual Notebook so they could return to them later or print them out for self-directed study.

No empirical tests were conducted on these projects, but a formal user evaluation was conducted with two science teachers to evaluate the utility of the prototype as a tool they could use to help teach problem-based learning courses. The teachers were given a sample lesson plan implemented in CAPABLE and were asked to simulate the problem-based learning process their students would conduct using the Intelligent Whiteboard. The researchers were present to answer general questions and debug interface problems in the prototype but did not provide the teachers with guidance on how to use the system.

The teacher's comments were logged during the session. In addition, the teachers themselves provided a postmortem written report evaluating their experience using the system and judging its suitability for use in their classes.

The results of this evaluation were favorable. The teachers comments indicated that they found the recommendations provided by the system to be useful. They furthermore felt that that the Virtual Notebook would be a useful tool for students conducting self-directed research as well as for their evaluation of student's progress. The time the teachers spent with the system was limited and clearly further evaluation needs to be done to validate the approach; however, the positive reviews so far suggest this further evaluation is definitely warranted.

#### **9.11.4. Evaluation 9.4. Support for Other Applications.**

Other experiments tested the context-sensitive asynchronous memory approach to information retrieval for different information retrieval tasks than web search. Other projects applied the Nicole-IRIA technology to browsing databases of pre-generated content to aid users navigation.

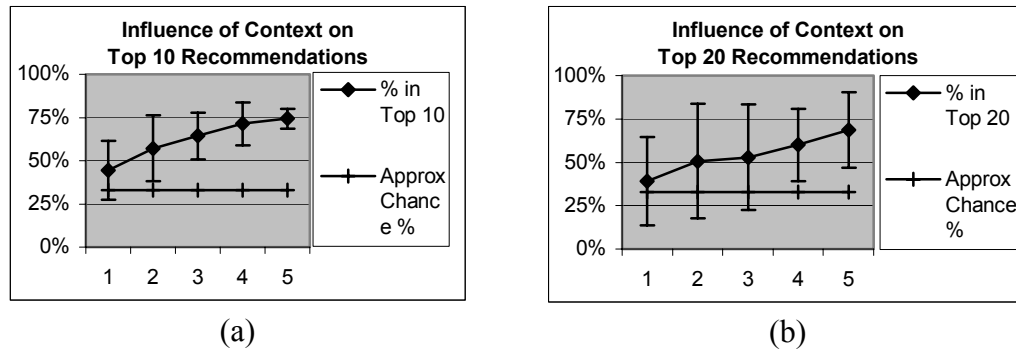
While no empirical tests were conducted on these projects, feedback from user evaluations consistently showed that users felt the system helped them get information they would not have found searching on their own.

#### **9.11.5. Evaluation 9.5. Analysis of Serendipitous Results.**

One interesting result arose during the test of Nicole-IRIA on a much larger data sets. As the data set grew larger the speed and performance of recommendations began to degrade because of insufficient contextual focusing. Analysis of the active nodes and relations revealed that some gated spreading activation was active but that it was insufficient to fully restrict search, and I hypothesized that increasing the amount of activation of relevant relations would improve the system's performance. In an attempt to improve the recommendation performance of the system, of the system, I performed a search for the module of the code that was activating the relations prior to search.

No such module existed. Intrigued, I performed a further analysis. Relations relevant to the task that the system was performing were automatically becoming active in part due to the *relation leakage* parameters discussed in Chapter 3. Because of the volume of nodes activated by Nicole-IRIA was much larger than in Nicole-MPA, relation leakage was now playing a significant role in automatically restricting the system's search to improve its performance.

While this result was intriguing I did not have time to explore the possibilities of relation leakage further as part of those experiments, and instead incorporated code to perform relation activation deliberately as part of the retrieval requests' cue list. This modification successfully improved both the speed and quality of the system's performance.



**Figure 9.7. Quality of Recommendations with Increasing Context**

It remains an interesting topic for future work to return to the topic of relation gating, varying its parameters to demonstrate whether or not the automatic focusing it contributes is useful and whether or not it can be increased to perform focusing at the same level as deliberate cueing.

## 9.12. Explanation and Analysis

### 9.12.1. Context-sensitive asynchronous memory can recommend useful information

Context-sensitive asynchronous memory can retrieve information relevant to reasoning tasks as judged by human users.

This thesis was demonstrated most strongly through the experiment conducted in Evaluation 9.1, in which the system's performance was compared against precompiled human judgements of relevance. These evaluations showed that the system was able to produce presentations of information which contained far more relevant information than

chance with only a few clicks acting as context; in a few limiting cases the presentation of relevant information maxed out at 100%.

This formal evaluation is backed up by the qualitative and subjective measures of Evaluations 9.2-9.4, in which human users in naturalistic settings repeatedly judged the recommendations provided by the system to be relevant.

### **9.12.2. Feedback can improve context-sensitive asynchronous retrieval**

Contextual feedback from task performance can improve the relevance of context-sensitive asynchronous memory retrievals at providing relevant information.

Contextual feedback, which in Nicole-IRIA consists of features users can observe about the information resources they are presented (such as words in the titles, summaries or text of a page) along with relationships connecting those features to potential answers (such as the title word and summary word relationships) was shown to improve the relevance of retrieval. The observable features provided the context-sensitive search system with information about what users found to be relevant; the relations enabled the CDSA algorithm to focus spreading activation effort towards potential answers.

The effect of feedback was demonstrated most strongly in Evaluation 9.1, in which the percentage of relevant recommendations was shown to increase steadily as more contextual information became available. Figure 9.7 averages recommendation quality



across all data sets: the percentage of relevant recommendations in the Top 10 rises from near-chance levels with a single click to around 75% relevant recommendations after 5 clicks.

The confidence intervals of these figures also illustrate the importance of context. Error bars are set to a 95% confidence interval with a sample size of 7. There is a ceiling effect on top 10 recommendations: by the second click, the mean is always above chance levels with a high degree of confidence. However, for top 20 recommendations it takes four to five clicks worth of context to achieve the same effect. Moreover, the confidence interval is fairly broad, providing a statistical way to express the intuition one might get from Figures 9.5 and 9.6: that different sets of contextual information have a significant effect on the quality of recommendations.

This was also shown more qualitatively in Evaluation 9.2, in which context drawn from a simulated workgroup enabled the recommendation of information not found using the context drawn from a single user.

### **9.12.3. Asynchronous retrieval is useful**

Asynchronous retrieval of information can enable a context-sensitive asynchronous memory to incrementally retrieve additional useful information.

This was demonstrated most strongly in Evaluation 9.2, in which the asynchronous search system was able to extend its additional presentation of results to a single user over time by drawing additional resources from the knowledge map based on its current context. Some of these asynchronous results were drawn from the original knowledge map based on context-driven search; other asynchronous results were found as a result of additional harvesting caused by searches from the other users in the simulated workgroup. In addition, some of the demonstrations conducted in Evaluation 9.4 showed a similar effect: when information contained in the knowledge map was relevant to a search, it was retrieved on the basis of the user's current context.

#### **9.12.4. Generality of the context-sensitive asynchronous memory approach:**

Context-sensitive asynchronous memory can recommend useful information for a variety of tasks.
--

The breadth of evaluations conducted demonstrate this in three ways.

First, the context-sensitive asynchronous memory approach to recommending information based on reminding was shown to work for a variety of different kinds of information — a variety of general web searches, employment data, educational information and other precompiled data sets.

Second, the fact that this same recommendation process could be easily embedded into several different applications also demonstrates its generality.

Finally, the fact that the same context-sensitive asynchronous memory process that worked for memory retrieval for Nicole-MPA and ISAAC could also work for information retrieval is a strong indication of its generality.

### **9.12.5. Summary of Results**

Overall, this case study revealed both strengths and limitations in my initial hypotheses on how the experience-based agent approach could be applied to information retrieval from the World Wide Web. The two most important hypotheses of the research were validated, with some caveats:

- **context-sensitive asynchronous memory can provide users useful information based on unobtrusive feedback:**

Empirical tests and user evaluations of the web search prototype demonstrated that the context-sensitive asynchronous memory approach to information retrieval is a feasible technology for managing and organizing information in a database and presenting useful information to users in response to their questions based on unobtrusive feedback derived from user browsing activity.

- **context-sensitive asynchronous memory retrieval can provide useful answers for many different information retrieval tasks:**

Empirical tests of the web search prototype and user evaluations of other prototype applications demonstrated that context-sensitive asynchronous memory based information retrieval is a general approach, capable of making useful

recommendations from several different kinds of information databases using several different kinds of contextual information.

Together, these results demonstrate the feasibility of context-sensitive asynchronous memory approach to provide a real benefit to a real task — to provide useful answers to user questions based on feedback from user browsing.

```
(defparameter *iria-cdsa-table*
  (make-parameter-table
   :iria-cdsa
   '(
    ;; This pair of parameters determines the total number
    ;; of nodes activation can theoretically spread to.
    (*ACTIVATION-THRESHHOLD* . 0.001 )
    (*ZORCH-ATTENUATION* . 0.9 )

    ;; This set of parameters determines spreading activation properties.
    (*BASELINE-PROPAGATION* . 0.2 ) ; Base propagation of zorch
    (*GATED-PROPAGATION* . 0.8 ) ; Additional "gated" zorch
    (*RELATION-ATTENUATION* . 0.5 ) ; Zorch leakage to relations
    (*CONTEXT-THRESHHOLD* . 0.01 ) ; Is context damping turned on?
    (*NON-CONTEXT-DAMPING* . 0.5 ) ; Zorch damping outside context

    ;; This set of parameters determines decay properties.
    (*DECAY-ATTENUATION* . 0.75 )
    (*IRRELEVANCE-DECAY* . )

    ;; This pair of parameters determines the relative strength of
    ;; knowledge and association links.
    (*KNOWLEDGE-STRENGTH* . 1.0 )
    (*ASSOCIATION-STRENGTH* . 5.0 )
    (*LEARNING-ACTIVATION* )

    ;; These parameters determine how propagation works
    (*PROPAGATION-LIMIT* . 500 ) ; How many get propagated
    (*DEFAULT-BASE-ACTIVATION* . 1.0 )
    (*ACTIVE-THRESHHOLD* . 0.0001 )

    ;; These parameters determine how retrieval itself works
    (*RETRIEVAL-THRESHHOLD* ) ; Excludes candidates if set
    (*RETRIEVAL-LIMIT* . 50 ) ; How many get matched

    ;; These parameters determine how matching works
    (*CANDIDATE-THRESHHOLD* . 0.001 ) ; Threshold of matchy goodness
    (*CANDIDATE-POLICY* . :STRICT ) ; How we replace candidates
    (*INHERIT-SPEC-MATCH* . :INHERIT) ; Do we check inheritance?
    (*INHERIT-CUE-MATCH* . :LOCAL ) ; Do we check inheritance?
    )
  )
  "Tweaked CDSA for IRIA."
)
```

Figure 9.8 One Memory Parameterization used in Nicole-IRIA

## 9.13. Lessons Learned

Because this case study extended the context-sensitive asynchronous memory approach to new areas, new lessons were learned about how the theory could be applied.

The most important of these lessons include:

- **comprehensive cost control policy is required for good performance**

Used as a search engine, the Nicole-IRIA system accepts far more queries at one time than arose in the planning case study or the ISAAC system. As users ask questions, Nicole-IRIA builds its knowledge map and continues to seek answers to old questions; as the number of questions grow in size the performance of the memory retrieval system degrades, both from the cost of matching the specifications of each query and because of the amount of activation spreading out from all active queries. To prevent the system from grinding to a halt, it is not enough to simply limit the amount of activation that can spread from a single node. A comprehensive cost control policy must also limit:

- **Number of active queries ( $R_{max}$ ):**

The number of active queries — queries are generating cues for the context-sensitive search process and which are actively being matched — must be bounded. Otherwise, the time cost of each retrieval cycle increases linearly with the number of active queries. The solution is to limit the number of active retrieval requests. For Nicole-IRIA  $R_{max}$  was set to 3-5.

- **Complexity of each active query ( $R_{cuemax}$ ):**

The amount of activation that an individual query propagates must also be controlled. The amount of activation spreading in the knowledge base, and hence the cost of each cycle of context-sensitive search, is a linear function of the number of nodes propagating activation. Even if only one query is active, that single query can swamp the system if it has enough active cues. This issue also arose during the design of the MPA system. The solution is to bound the number of active cues for any one retrieval request. For Nicole-IRIA  $R_{cuemax}$  was set to between 20 and 30.

- **Complexity of matching ( $R_{specmax}$ ):**

Similarly, the complexity of matching the specifications of a retrieval request must be controlled. As Tambe & Rosenbloom (1994) point out, matching is a computationally complex operation which can swamp all other processes in the system if not controlled. Currently, no existing context-sensitive asynchronous memory implementation imposes such a limit; instead, the limits are at the discretion of the designer.

- **Complexity of fan-out:**

One final issue is the expense of individual node propagations in spreading activation. As knowledge bases grow very large individual nodes may have tens of thousands of connections to other nodes. Even though no activation is ever likely to propagate along those nodes because of fanouts and thresholds,

computing those connections and determining that their weights are too small can become prohibitively expensive. Individual node connection/propagation computation cost must be controlled, especially on serial systems with very large knowledge bases. One solution is to place a threshold limit on fanout, preventing propagation when the number of connections exceeds some upper bound. Finding effective metrics for bounding fanout is an interesting open research area.

- **parameterization of memory retrieval**

This case study provided feedback about the proper parameterization of the memory retrieval system for general performance. The planning case study used a narrow candidate buffer (20 items) and lightweight context weighting; these parameters were sufficient to enable retrieval of planning cases. However, the same parameters provided much poorer performance on the information retrieval domain, and required some updating (bold items in Figure 9.8):

- **activation threshold:**

The high fanout of nodes in the information retrieval domain mean that the amount of propagating perturbation fell off rapidly. To ensure a good sample of nodes were activated, the lower limit on propagating perturbation (**ACTIVATION-THRESHOLD** in Figure 9.8) was decreased from 0.01 to 0.001, effectively letting the system visit ten times as many nodes. Overall cost was still controlled by upper limits on the number of propagating quanta

(**PROPAGATION-LIMIT** in Figure 9.8) which was reduced from 1000 to 500 without affecting the quality of results.

- **size of candidate buffer:**

The small buffer size caused problems on the information retrieval domain because retrieval cues swamped the candidate buffer. One consequence of fanout in spreading activation is that the cues of retrieval requests are often the most active items in memory; a candidate buffer must be sufficiently large to capture other active items in order for retrieval to be successful. The large number of words cues active in information retrieval swamped the smaller buffer size; a larger buffer (increasing **RETRIEVAL-LIMIT** in figure 9.8 from 20 to 50 items) provided improved performance.

- **strength of context-directed spreading activation:**

Originally, the information retrieval domain used a lightly weighted version of the context-directed spreading activation algorithm, which provided the high quality results illustrated in the experiments. However, when the system was scaled up to other applications with large precompiled databases with thousands of items and more simultaneous searches the quality of recommendations tended to degrade. Increasing the strength of context-directed spreading activation improved the quality and speed of the system's performance without degrading earlier experiments (setting the **BASELINE-PROPAGATION** and **GATED-PROPAGATION** parameters in Figure 9.8).



- **auto-generated context:**

An interesting aspect of the context-directed spreading activation algorithm is that some amount of activation can effectively “leak” up from links along which activation travels to the relations those links instantiate (setting the **RELATION-ATTENUATION** parameter in Figure 9.8 to 0.5). In the information retrieval domain, this resulted in automatic activation of relations related to the questions the system was being asked, which further focused retrieval along those links, which further activated the relations in a self-reinforcing cycle. This automatic generation of context significantly improved the performance of the system by limiting activation flow to other parts of the knowledge base not related to the user’s current context without the need for hand-coding a context set.

- **limitations of retrieval as knowledge bases grow very large**

As knowledge bases grow large within a particular task, domain and problem — when more than a thousand items of one specific category in the knowledge base grows to the thousands and the total number of nodes that define those item grow to the tens of thousands — the quality of retrieval can degrade. As an example, consider an information retrieval domain for recommending movies. Context-sensitive asynchronous retrieval could provide useful retrieval for a knowledge base with thousands or tens of thousands of entries assuming that the knowledge base contained many different kinds of recommendations and that each entry in the knowledge base was richly described. However, if the database contained

only a few categories — say five thousand science fiction movies and five thousand mysteries, each thinly described — it will be difficult for a context-sensitive asynchronous memory approach to provide useful recommendations. The reason is when many similar entries in a category exist, the fan-out from each potential cue to each potential target becomes very large and prevents activation from spreading from cues to targets. The solution to this problem is to impose additional structure on the knowledge base, enabling activation to flow appropriately from cues to target based on the context. This structure can be provided directly as part of the knowledge base — for example, by subcategorizing the elements in each category or by providing additional descriptions of each item. Alternatively, structure may be generated automatically through a dynamic memory approach (Schank 1982) which develops knowledge organization for a knowledge base based on the history and content of memory storage and retrieval requests. One example of such a system is the CYRUS reconstructive memory system (Kolodner 1983) which automatically generates indexing and generalization structures to aid in the organization and retrieval of information.

## **9.14. Conclusions**

The information retrieval case study demonstrated several key things about context-sensitive asynchronous memory, experience-based agency and how they could be applied to the information retrieval tasks.

The case study showed that the experience-based agent approach is a viable approach to information retrieval. The core context-sensitive asynchronous memory algorithm is capable of recommending useful information to users based on highly impoverished information, and the rich representation of information in an experience store enables not only the construction of knowledge maps that enable these recommendations but also enables the rapid development of a variety of information retrieval tasks.

Generalizing from these results, the case study also provided evidence that the context-sensitive asynchronous memory approach is useful in its own right. Context-sensitive asynchronous memory can provide real users with both quantitative and qualitative benefits on a real task, is flexible, capable of being rapidly extended to new areas, and is scalable, capable of being applied to larger domains with only minor modifications to the approach's parameters and without modifications to its core algorithms.

Taken as a whole, the case study provides strong evidence for the core claim of the thesis, that the context-sensitive asynchronous memory approach is a general solution to finding useful answers to vague questions using feedback from task processing.

# CHAPTER X.

## FEASIBILITY OF THE APPROACH

---

*Evaluating the core claims of the context-sensitive asynchronous memory approach*

### 10.1. Overview

The planning and information retrieval case studies investigated how context-sensitive asynchronous memory could be applied to solve real problems. However, these studies only obliquely addressed the details of the context-sensitive asynchronous memory approach and what contributes to its performance.

This chapter discusses a feasibility study of the context-sensitive asynchronous memory approach as a general memory retrieval system for intelligent agents. The feasibility study was conducted on the two existing implementations Nicole-MPA and Nicole-IRIA. In the study, the context-sensitive asynchronous memory approach was evaluated using results from previously conducted experiments, as well as novel experiments designed to illuminate certain properties of the system more closely, all with an eye of evaluating its fitness as a memory retrieval system.

## 10.2. Goals of the Feasibility Study

This case study was designed to illuminate the suitability of context-sensitive asynchronous memory as a general memory retrieval approach. The specific thesis tested was:

The context-sensitive asynchronous memory approach is a viable approach to memory retrieval for general intelligent agents.

In chapter 1, this general assertion was unpacked into a series of desiderata for memory systems for general cognitive agents. The feasibility study was designed to validate that the approach did indeed satisfy the elements of these desiderata.

## 10.3. Methodology of the Study

Testing the feasibility of a memory approach requires applying the approach to memory retrieval problems, evaluating its performance, comparing it with other approaches, and testing its parameters to determine what contributes to that performance.

I chose to apply context-sensitive asynchronous memory to memory problems drawn from the planning and information retrieval problems from the case studies, ensuring the system was tested against data used by real AI systems.

To test the approach, I built variant versions of a context-sensitive asynchronous memory and alternative memory approaches, along with sets of knowledge bases for

retrieval. Using this foundation, I evaluated various properties of retrieval across memories, memory parameterization and retrieval approaches.

## **10.4. Applying the Model**

Rather than the task of information retrieval or the task of solving planning problems, the task of the feasibility study was simply the task of memory retrieval: how good is context-sensitive asynchronous memory at retrieving information on the basis of the provided specifications? The core issue was exploring the properties of context-sensitive asynchronous memory, including issues such as context-sensitivity, asynchronous retrieval, and anytime retrieval, using real memory retrieval problems drawn from real tasks to both validate the theory and help understand how it could be applied.

## **10.5. Conducting the Study**

The testbed for these feasibility evaluations was Nicole-MOORE, the memory retrieval module of the Nicole system. Within Nicole-MOORE's framework, I developed a variety of algorithms, problem sets and other support structures, including:

- **Memory Retrieval Approaches:**

To compare context-sensitive asynchronous memory with other approaches, I developed several alternative retrieval systems with an identical functional profile.

- **Problem Sets:**

To test retrieval performance, I developed retrieval problem sets drawn from actual case retrieval problems used in Nicole-MPA and from actual web searches suitable for Nicole-IRIA.

- **Knowledge Structure Strategy:**

To test the impact of knowledge base structure on retrieval, I developed several knowledge structure strategies for Nicole-MPA's plan tagging system.

- **Knowledge Bases:**

To test the impact of knowledge base size and heterogeneity on retrieval, I developed several test knowledge bases of varying size and composition, and developed ways for Nicole-IRIA to harvest large knowledge bases on its own.

- **Context Sets:**

To test the impact of contextual information on retrieval, I developed several context sets of varying degrees of complexity for Nicole-MPA's retrieval system.

The following sections review this experimental framework in soporific detail.

### **10.5.1. Memory Systems Constructed**

To test the performance of the CDSA algorithm with respect to other approaches that did not incur its overhead (the cost of spreading activation) or its limits (no guarantee of optimal retrieval) I constructed two memory systems within the Nicole-MOORE framework:

- **Ruminator:**

The Ruminator was a variant of Nicole-MOORE which employed an asynchronous memory manager and a fully parameterizable context-sensitive spreading activation search process. The default parameterization of the Ruminator is the CDSA algorithm proposed in this research.

- **Brutalizer:**

The “Brutalizer” was another variant of Nicole-MOORE, employing the same asynchronous memory manager as the Ruminator but using a brute-force exhaustive memory retrieval process. While the Brutalizer algorithm is an exhaustive matching algorithm, it does not incur the space or time overhead of CDSA and is guaranteed to return an optimal result from the knowledge base.

In addition, to test the contribution of the CDSA algorithm to memory retrieval, I constructed a series of parameterizations for the Ruminator system, enabling it to simulate a variety of different retrieval regimes:

- **Traditional Spreading Activation**

The simplest parameterization of the Ruminator did not use any context effects ( $P_{context}$  and  $R_{gating} = 0$ ) and simulated a “traditional” spreading activation process. For parallelism with the CDSA system, this parameterization included a parameter to decay items retrieved but not matching any outstanding retrieval.



- **CDSA:**

Another parameterization of the Ruminator used a full set of CDSA parameters, including both primed and gated spreading activation. This parameterization also included a decay parameter (**IRRELEVANCE\_DECAY** = 0.5) that decreased the activation of active nodes which did not match any outstanding retrieval request. This parameter has the effect of causing the system to more rapidly traverse its active list until it finds relevant items.

- **Other CDSA Parameterizations**

There were a variety of other CDSA parameterizations developed, from “light” CDSA with little context effects and no irrelevance decay to “extreme” CDSA with little or no normal propagation and heavy decay and attenuation parameters. These parameterizations were not used in the formal experiments but were used in the development of Nicole-MPA and Nicole-IRIA.

### **10.5.2. Retrieval Problem Sets**

To test Nicole-MOORE’s retrieval capabilities, I developed two retrieval problem sets for problem solving and information retrieval.

- **The X2 Problem Set**

This problem set was drawn from the Nicole-MPA X2 Problem Library. The Nicole-MPA system can automatically translate each unsolved problem into a

retrieval request that it could post to Nicole-MOORE. The X2 Problem Set is presented in its entirety in Appendix B.1.

- **Information Retrieval Problem Set**

This set of test retrieval queries for Nicole-IRIA consisted of a set of typical queries entered by actual users of Nicole-IRIA, such as “centaur” or “java software development”. The Nicole-IRIA system automatically translates each query into a retrieval request that it could post to Nicole-MOORE. The Information Retrieval Problem Set is presented in its entirety in Appendix B.2.

- **Metaspy Problem Set**

To provide a more ecologically valid problem set drawn from a larger user population, I harvested 100+ typical queries from the MetaSpy search monitoring service, which displays actual queries entered by users into the MetaCrawler search engine ([www.metaspy.com](http://www.metaspy.com)). This dataset was collected from the “clean” version of MetaSpy, which filters more colorful language and topics. A small set of queries ( $N < 10$ ) were manually tweaked because they caused problems for the prototype’s data harvesting system; this included several queries which used Boolean operations not fully supported by Nicole-IRIA and three queries which caused errors in the Perl Harvester. The MetaSpy Problem Set is presented in its entirety in Appendix B.3.

### 10.5.3. Knowledge Structure Strategy

The Problem Solving Problem Set retrieval requests were generated by a parameterizable *knowledge structure strategy* defined by Nicole-MPA. Because Nicole-MPA used the native SNLP/SPA plan representation, stored plans in the knowledge base had to be augmented with “plan tags” that made the internal structure of the plans visible. A series of plan tagging algorithms of varying degrees of detail were constructed, enabling both retrieval of plans (important for the planning case study) and tests of the contribution of knowledge base structure to retrieval (important for this feasibility study). These included “predicate”, “atomic”, and “nuclear” cueing strategies.

**Predicate Cueing.** Predicate cueing is the least structured of all the plan tagging strategies explicitly studied. In predicate cueing, SPA plan objects are tagged using the predicates and objects in the initial and goal conditions in the plan. For example, a SPA plan object that included the initial condition literal:

**(AT-PERSON RESEARCHER AIRPORT)**

and the goal condition literal:

**(HOLDING RESEARCHER LUGGAGE)**

would be annotated with four sets of CRYSTAL links: an initial predicate list containing **AT-PERSON**, an initial object list containing **RESEARCHER** and **AIRPORT**, a goal predicate list containing **HOLDING** and a goal object list containing **RESEARCHER** and **LUGGAGE**.

```

#k<PLAN.455
  :initial-predicates    (AT-PERSON.400)
  :initial-objects      (RESEARCHER.421 AIRPORT.401)
  :goal-predicates      (HOLDING.411)
  :goal-objects         (RESEARCHER.421 LUGGAGE.422)
  ...
>

```

Thus, the predicate plan tagging representation does not make explicit the relationship between the predicates of a literal and the objects that are the arguments of that predicate.

**Atomic Cueing.** In atomic cueing, the literals that make up the initial and goal conditions of SPA plan objects are extracted as “atoms” — complete CRYSTAL knowledge objects that represent the content of a literal, including its predicates and objects, in a frame format. For example, the SPA plan object mentioned earlier would be annotated with two atoms:

```

#k<PLAN.598
  :initial-condition     ATOM.599
  :goal-condition        ATOM.600
  ...
>

```

The initial atom represents **(AT-PERSON RESEARCHER AIRPORT)** as

```

#k<ATOM.599
  :predicate            AT-PERSON.400
  :negated              FALSE.321
  :argument-1           RESEARCHER.421
  :argument-2           AIRPORT.401
>

```

and the goal atom represents **(HOLDING RESEARCHER LUGGAGE)** as

```

#k<ATOM.600
    :predicate      HOLDING.411
    :negated        FALSE.321
    :argument-1     RESEARCHER.421
    :argument-2     LUGGAGE.422
>

```

While the names of predicates and objects are shared between atoms, each atom is unique. Thus, if a new plan was created which also had the goal of (**HOLDING RESEARCHER LUGGAGE**), a new atom would be created, sharing the **HOLDING.411**, **FALSE.321**, **RESEARCHER.421** and **LUGGAGE.422** nodes but with a unique name and identity (e.g., **ATOM.701** or some such).

Thus, while the atomic plan tagging representation makes the relationships between the predicates and objects of a plan literal explicit, it does not note identity relationships between whole literals when those relationships exist.

**Nuclear Cueing.** Nuclear cueing has the richest structure of all the plan tagging structures discussed so far. It augments atomic cueing by representing not just the structure of cues but their identity through use of the CRYSTAL placeholder system. Thus, the plan listed earlier would be annotated not only with two atoms but also two additional placeholder objects representing the initial and goal conditions:

```

#k<PLAN.700
    :initial-condition  ATOM.701
    :initial-kcond     AT-PERSON-RESEARCHER-AIRPORT.721
    :goal-condition     ATOM.722
    :goal-kcond        HOLDING-RESEARCHER-LUGGAGE.745
    ...
>

```

A new plan with an identical initial or goal condition would share this exact object. Thus, the nuclear plan tagging representation captures both the structure and the identity relationships of plan literals explicitly in the knowledge base.

**Other Plan Tagging Strategies.** Many other plan tagging strategies were constructed for experimental and development purposes but were not used in the formal experiments.

#### **10.5.4. Knowledge Bases Constructed**

To test how the content of a knowledge base would affect retrieval, four different knowledge bases were constructed:

- **Null (“empty”) knowledge base**

The Null knowledge base contained 315 nodes, the absolute minimum necessary to support the core ontology used by the memory, task and MPA modules plus a small casebase of plans drawn from the Inexperienced Library that made retrieval possible.

- **Distractor (non-task) knowledge base**

The Distractor knowledge base contained 696 nodes, which included the core ontology and Inexperienced Library of the Null knowledge base, as well as a body of knowledge irrelevant to the planning task drawn from the ISAAC knowledge base.

- **In-Domain knowledge base**

The In-Domain knowledge base contained the core ontology and Inexperienced Library of the Null knowledge base, plus a large casebase of potentially relevant plans drawn from the same Stinger Missile domain as the retrieval problem set.

The size of the In-Domain knowledge base varied on the Nicole-MPA knowledge structure strategy used. With the simple predicate plan tagging strategy it contained 597 nodes, whereas using the more complex nuclear plan tagging strategy it contained 1288 nodes.

- **Non-Domain (in-task) knowledge base**

The In-Domain knowledge base contained the core ontology and Inexperienced Library of the Null knowledge base, plus a large casebase of potentially irrelevant plans drawn from different planning domains. The size of the Non-Domain knowledge base varied on the Nicole-MPA knowledge structure strategy used.

With the simple predicate plan tagging strategy it contained 696 nodes, whereas using the more complex nuclear plan tagging strategy it contained 2579 nodes.

In addition, other knowledge bases were constructed for a variety of informal tests, including various combinations of the Nicole-MPA, Nicole-IRIA and ISAAC knowledge bases.

### **10.5.5. Context Sets Constructed**

In normal circumstances, a knowledge structure strategy also specifies a set of context items which are used to affect context-directed spreading activation. However, to explicitly test how the content of contextual information would affect retrieval independently from the knowledge structure strategy, five different “context sets” were constructed:

- **Null Context Set (Control Condition)**

The null context provided no cues to spreading activation other than the cues that Nicole-MOORE automatically generated from the retrieval specification. This was a control condition.

- **Objects Seen Context Set**

In the “objects seen” condition the list of objects contained in the problem statement, such as the Researcher or his Luggage, were used to cue spreading activation. This condition was designed to simulate a context search strategy in which items an agent had seen caused activation of potentially relevant past cases, potentially making it easier to retrieve them through priming spreading activation.

- **Generic Concept Context Set**

In this condition, general concepts in the planning domain — including the superclass objects for atoms, plans and solutions — were used to cue spreading activation. This condition was designed to simulate a context search strategy in which an agent considered the kinds of objects it would need to retrieve,



potentially activating those objects and making it easier to retrieve them through priming spreading activation.

- **Relations Context Set**

In the relations condition, relationships that connected objects in the domain with potential solutions were used to cue spreading activation. Sample relations included the relations connecting solutions to initial and goal condition atoms. This condition was designed to simulate a context search strategy in which an agent considered the relationships between objects it had seen and the kinds of items it needed to retrieve, potentially making it easier to retrieve those objects via gated spreading activation.

- **Relations+Objects (Combo) Context Set**

In this condition, both the objects seen in the problem statement and the relations that might connect those objects to potential solutions were activated. This condition was designed to simulate a context search strategy in which an agent considered both the objects it had seen and the relationships between those objects and the kinds of items it needed to retrieve, potentially making it easier to retrieve those objects via gated spreading activation.

#### **10.5.6. Other Efforts**

In addition to the above efforts, a variety of other tests were conducted using Nicole-MPA, Nicole-IRIA and Nicole-MOORE proper, including profiling the memory retrieval system, observing the system's performance over long periods of time and large

knowledge bases, and analyzing how various parameters and features affected its performance.

## **10.6. Fidelity of the Study**

As usual, there were limits to the study. Since neither Nicole-MPA nor Nicole-IRIA made use of automatic cueing based on working memory contents or incorporated a storage module, these aspects of the theory were not tested.

A more important limitation was on scaleup — while the knowledge bases tested were large with respect to the knowledge bases of planners like SPA (Hanks & Weld 1994) and PRODIGY/ANALOGY (Veloso 1994), they were by no means large with respect to the knowledge bases found in CYC (Lenat & Guha 1990), the Botany Knowledge Base (Clark & Porter 1996) or the UMLS medical database (National Library of Medicine 2000).<sup>24</sup> Continuing to test the system with larger and larger knowledge bases is an important area of future research.

---

<sup>24</sup> Note that MPA used an expansion of SPA's released case library, containing a few hundred cases; PRODIGY/ANALOGY was tested with approximately 1000 cases; the maximum Nicole-IRIA knowledge base tested so far contained approximately fifteen thousand resource pointers comprising 50,000 semantic network nodes. The UMLS and Botany Databases, in contrast, have hundreds of thousands of nodes and the CYC network has millions.

Another issue was the degree to which the memory retrieval problems provided by Nicole-MPA and Nicole-IRIA fully exercised the context-sensitive asynchronous memory system. The quality of a memory retrieval system can be judged against the requirements of the task, as was shown in the Nicole-IRIA evaluations; however, when judging a memory system *qua* a memory system the evaluation of the system's performance should be compared with respect to the specifications that are provided to it. This proved to be limiting in the case of Nicole-MPA, whose memory retrieval specifications were "good enough" for problem solving but placed much of the burden of assessing the quality of retrieval on the planning task. Therefore, some of the tests of the quality of retrievals for Nicole-MPA encounter ceiling effects in which the relative quality of multiple possible retrievals was identical as far as the specifications it was provided. Examining Nicole-IRIA's retrieval performance, in contrast, enabled a more direct analysis of the quality of retrieval.

## 10.7. Hypotheses Tested

For each of the desiderata, the context-sensitive asynchronous memory approach advances testable claims about how it satisfies those desiderata which were the focus of this feasibility study.

- **Desideratum 1: Able to store a wide variety of information**
  - A reified, grounded, bidirectional semantic network is sufficient to represent knowledge for a variety of tasks.

- **Desideratum 2: Provides a general method for accessing that information**
  - A context-sensitive asynchronous memory can retrieve knowledge in service of a wide variety of reasoning tasks in a wide variety of domains.
- **Desideratum 3: Scales to large multifunction knowledge bases**
  - Context-sensitive asynchronous memory can effectively perform retrieval on large knowledge bases.
  - Context-sensitive asynchronous memory can effectively perform retrieval from heterogeneous multi-task knowledge bases.
- **Desideratum 4: Manages cost of retrieval**
  - Context-sensitive asynchronous memory is efficient enough to serve the retrieval needs of a typical performance task.
  - Context-sensitive asynchronous memory performs as well as or superior to existing approaches that do the same task (in other words, competing memory systems with the same functionality).
  - Context-sensitive asynchronous memory's cost control policy ensures that memory retrieval performance remains efficient and effective even when a system is faced with large or numerous retrieval requests.
- **Desideratum 5: Preserves accuracy of retrieval (in the face of resource limits)**

- Context-sensitive asynchronous memory can search a knowledge base incrementally, expending resources in parallel with other reasoning processes.
- Context-sensitive asynchronous memory can act as an anytime retrieval system, improving quality and/or quantity of retrieval over time while at any time retaining the ability to return the best result it has found so far.
- **Desideratum 6: Exploits task/environmental information**
  - Context-sensitive asynchronous memory can heuristically adjust its search based on context (information not contained within the content of a query) to improve its retrieval performance.
- **Desideratum 7: Exploits extra resources if available**
  - Context-sensitive asynchronous memory can improve the comprehensiveness and accuracy of retrieval if given additional resources to continue search.
- **Desideratum 8: Potentially interleavable with reasoning**
  - Context-sensitive asynchronous memory can detect when a suitable retrieval is found and alert reasoning without being explicitly polled.
- **Desideratum 9: Provides guidelines for reasoning integration**
  - Experience-based agency provides guidelines for constructing reasoners that interoperate with context-sensitive asynchronous memory and can integrate spontaneous retrievals into their current processing state.

The experimental evaluations by and large validated these claims.

## **10.8. Results**

The feasibility study consisted of a series of evaluations testing the generality, scalability, and efficiency of the system; testing how contextual information affected retrieval; testing the contribution of cost control to system performance, and testing the contribution of anytime and asynchronous retrieval for exploiting resources. These evaluations include seven explicit experiments and four interpretive summaries of evidence collected in the previous case studies and in exploratory studies. The complete set of evaluations included:

- **Evaluation 10.1: Representational and Retrieval Generality**
- **Evaluation 10.2: Evaluation of Scaleup I: Retrieval Performance Across Knowledge Bases**
- **Evaluation 10.3: Evaluation of Scaleup II: Interactions between Platform and Scaleup**
- **Evaluation 10.4: Evaluation of Scaleup III: Simulated Accumulation Tests**
- **Evaluation 10.5: Evaluation of Scaleup IV: Followup Accumulation Tests**
- **Evaluation 10.6: Evaluation of Scaleup V: Effect of Seed Knowledge Bases**

- **Evaluation 10.7: Comparative Performance Analyses**
- **Evaluation 10.8: Contribution of Context I**
- **Evaluation 10.9: Contribution of Context II**
- **Evaluation 10.10: Analysis of Cost Control**
- **Evaluation 10.11: Analysis of Asynchronous and Incremental Search**

### **10.8.1. Evaluation 10.1: Representational and Retrieval Generality**

To test the generality of the context-sensitive asynchronous memory approach, the CRYSTAL and Nicole-MOORE systems which instantiate that approach were used to construct a set of applications.

**Method.** Nicole-CRYSTAL and Nicole-MOORE were used to develop knowledge representations and memory retrieval for three systems: the planning system Nicole-MPA, the information retrieval system Nicole-IRIA, and the story understanding system ISAAC. In addition to these systems, several other artificial intelligence systems were built using the CRYSTAL representation include MOORE, TaskStorm and Command core modules of the Nicole system itself.

**Results.** Nicole-CRYSTAL successfully supported the memory representation needs of all of these systems, each of which had slightly different requirements for knowledge representation. In addition, the Nicole-MOORE module was successfully used to support the retrieval needs of Nicole-MPA, Nicole-IRIA, and ISAAC. All three successfully use

the Nicole-MOORE module to retrieve information in service of three very different tasks. In addition, the Nicole-MOORE module was tested in a variety of targeted demonstrations to evaluate its effectiveness.

*Nicole-MPA.* Nicole-MPA is an integrative case-based planning system which merges multiple cases together dynamically, using feedback from the planning process to inform the context-sensitive asynchronous memory process and aid the retrieval of additional cases. Nicole-MPA represents domains, problems and goals natively in the CRYSTAL knowledge representation. To plan, Nicole-MPA employs and extends the SPA planning algorithm, which employs its own native plan representation. Because the contents of these native plans are opaque to the experience store, MPA exposes the important contents of each case using plan tags which represent elements of a planning state in sufficient detail to retrieve relevant cases.

While virtually all of Nicole-MPA's data structures are implemented in CRYSTAL, its most extensive use of the system is for the plan tagging system. Nicole-MPA defines a set of top-level schemas representing domains, problems, plans and knowledge goals, and the establishes sets of relationships connecting these to enable context-directed retrieval in Nicole-MOORE. Figure 10.1 shows a subset of these relationship and frame definitions.



```

(create-relationship mpa-problem-of-relation (:problems :parent-domain))
(create-relationship initial-tags-relation (:initial-tags :initial-tags-inverse))
(create-relationship goal-tags-relation (:goal-tags :goal-tags-inverse))
(create-relationship initial-predicates-relation (:initial-predicates
                                                :initial-predicates-inverse))
(create-relationship goal-predicates-relation (:goal-predicates
                                                :goal-predicates-inverse ))
(create-relationship initial-objects-relation (:initial-objects
                                                :initial-objects-inverse ))
(create-relationship goal-objects-relation (:goal-objects :goal-objects-inverse))
(create-relationship predicate-relation (:predicate :predicate-inverse ))
(create-relationship argument-1-relation (:argument-1 :argument-1-inverse ))
(create-relationship argument-2-relation (:argument-2 :argument-2-inverse ))

(define-node 'mpa-domain-object mpa-object
  (:documentation "An MPA Domain" :system)
  (:domain-name nil :system)
  (:nicknames nil :system)
  (:literals nil :system)
  (:operators nil :system)
)

(define-node 'problem-statement-object mpa-object
  (:documentation "A Problem Statement" :system)
  (:parent-domain nil)
  (:type :original)
  (:problem-parent nil)
  (:initial nil)
  (:goal nil)
  (:solution nil)
  (:statistics nil :system)
)

(define-node 'knowledge-goal-object mpa-object
  (:documentation "A Knowledge Goal" :system)
  (:type :original)
  (:problem-statement nil)
  (:results nil)
  (:requests nil)
)

(define-node 'plan-object mpa-object
  (:documentation "A Plan Tag" :system)
  (:type :original)
  (:plan-value nil :system)
  (:initial-conditions nil :system)
  (:goal-conditions nil :system)
)

(define-node 'atom-object plan-element-object
  (:documentation "A Plan Atom Tag" :system)
  (:type :step)
  (:knowledge-goal nil)
)

```

**Figure 10.1 Relationships and Schemas in Nicole-MPA**

*Nicole-IRIA*. *Nicole-IRIA* is a context-sensitive asynchronous retrieval system which uses feedback from user browsing to recommend relevant information. *Nicole-IRIA* represents hypertext documents, hyperlinks, text, words, and other document-like objects natively in the CRYSTAL knowledge representation, using a rich ontology of Internet

information resources and an elaborate offline harvesting system which extracts and prepares information for entry into the experience store.

Nicole-IRIA uses CRYSTAL to create a rich representation of the information retrieval domain, representing not only queries, information resources, hyperlinks, words, summaries, and so forth as objects but also representing many different kinds of relationships between these objects to guide context-directed spreading activation. Figure 9.3 shows a graphical representation of a Nicole-IRIA knowledge network.

*ISAAC.* ISAAC is a story understanding program that embodies a theory of creative reading (Moorman 1997). ISAAC reads real science fiction stories written by human authors; to represent these stories ISAAC employs a rich ontology, written in the CRYSTAL language, that covers low-level text, events and objects, story structures, and rich cultural and background information.

Figure 10.2 shows a portion of an early ISAAC ontology, showing not only a rich top-level ontology of mental and physical objects and so forth (on the left), but also some complex lower-level structures representing things such as stories and expectations as hierarchically nested sub-objects within that top-level ontology (on the right).

<pre> (define-node 'super-thing frame   (:tense nil))  (define-node 'agent super-thing   (:name nil)   (:age nil)   (:occupation nil)   (:lifeform nil)   (:number 1)   (:represented-by nil)   (:emotional.state nil)   (:physical.state nil)   (:social.state nil)   (:temporal.state nil)   (:mental.state nil))  (define-node 'object super-thing   (:number 1)   (:object nil)   (:physical.state nil)   (:temporal.state nil))  (define-node 'physical.object object   (:description nil))  (define-node 'mental.object object) (define-node 'social.object object) (define-node 'emotional.object object) (define-node 'temporal.object object) </pre>	<pre> (define-node 'expectation mental.object   (:support nil)   (:expectation nil))  (define-node 'story-model mental.object   (:title (define-node 'anticipation-title anticipation     (:anticipation '(before paragraph 1)       (before author)       (is-a phrase))))   (:author (define-node 'anticipation-author anticipation     (:anticipation '(before paragraph 1)       (after title)       (is-a agent))))   (:setting (define-node 'anticipation-setting anticipation     (:anticipation '(is-a location))))   (:characters (define-node 'anticipation-characters anticipation     (:anticipation '(is-a agent))))   (:protagonist nil)   (:antagonist nil))  (define-node 'first-person-narrative story-model   (:protagonist 'I)) (define-node 'second-person-narrative story-model   (:protagonist 'self)) (define-node 'third-person-narrative story-model   (:where-protagonist 'who-knows)) (define-node 'story-title mental.object   (:title nil)   (:story nil)) </pre>
---	--

**Figure 10.2 Schema Definitions in ISAAC**

*Other Systems.* The CRYSTAL language has been used to develop a variety of small AI systems, including a course advisor expert system and several small planning and search systems. In addition, the MOORE, TaskStorm and Command modules of Nicole itself use CRYSTAL to represent retrieval requests, task structures, and command scripts. Chapter 7 presents examples of many of these data structures.

### **10.8.2. Evaluation 10.2: Evaluation of Scaleup I: Retrieval Performance Across Knowledge Bases**

The next major evaluation was a more extensive experimental evaluation of Nicole-MOORE's performance on the planning task. This experiment examined the performance of a typical case retrieval in Nicole-MPA, varying the size and character of the knowledge base against a set of test problems. The experiment measured several variables, including quantity, quality and comprehensiveness of retrieval results, speed

and quality of best guess retrieval, and overall retrieval effort as measured by nodes visited by spreading activation, total number of retrieval cycles and total time for retrieval.

**Method.** This experiment tested the memory retrieval performance of Nicole-MOORE on memory retrieval problems generated by the X2 problem set, drawing cases from the Inexperienced Library and using the nuclear plan tagging strategy. The experiment tested four major conditions:

- **The Null Knowledge Base:**

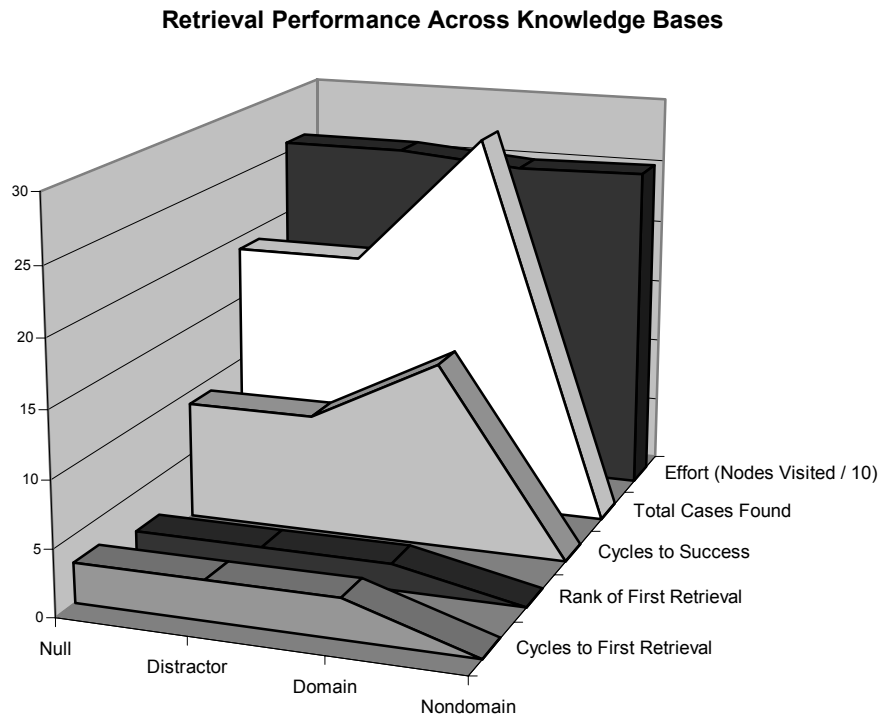
Containing only the Inexperienced Library of cases and limited amounts of other knowledge, the Null knowledge base was the smallest, least challenging knowledge base in this experiment and was used as a control condition.

- **The Distractor Knowledge Base:**

This knowledge base tested adding many nodes from the ISAAC knowledge base to test how knowledge from outside the domain pertaining to a questions would affect retrieval performance.

- **The Domain Knowledge Base:**

This knowledge base added many nodes from the Stinger Missile domain to test the ability of the system to efficiently find a best guess retrieval from a large set of potentially relevant choices.



**Figure 10.3. Effect of Knowledge Base on Retrieval Performance**

- **The Non-Domain Knowledge Base:**

This knowledge base added many nodes from planning tasks outside the Stinger Missile domain in an attempt to determine how the system would cope with many irrelevant cases which could not easily be discriminated from relevant cases.

For each of these conditions, the identity of the knowledge base and its size were recorded as dependent variables. In each condition, 12 problems from the X2 library were presented to Nicole-MPA, which generated and posted a request to Nicole-MOORE to find relevant cases. The experiment measured the following independent variables:

- **cycles to first guess**

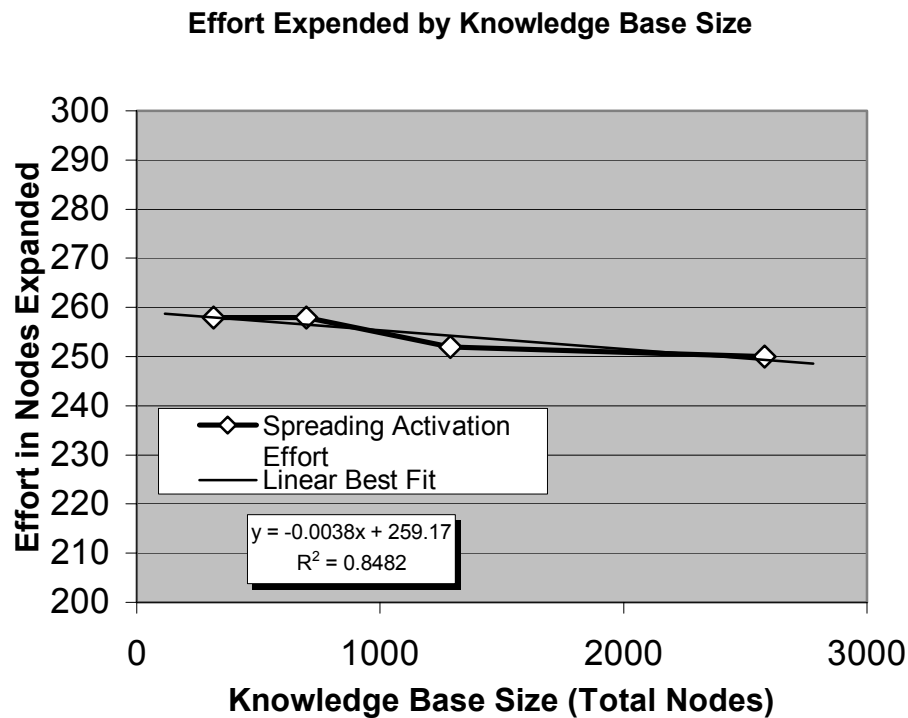
How many retrieval cycles did it take for the system to retrieve a “best guess” drawn from the target planning domain (which could potentially be used for adaptation)? Retrieval cycles to the first guess are a measure of how effective anytime retrieval is at quickly finding relevant information.

- **quality of first guess**

Another metric of anytime retrieval is the quality of the first guess as compared to all other items in the knowledge base, as measured by the similarity metrics provided to the memory system? Quality of retrieval was measured by sorting all the nodes in the knowledge base by similarity to the retrieval specifications and then measuring the rank of the retrieved node. Items with identical weights were given the same ranking. This ranking measure was based on the retrieval specifications provided by Nicole-MPA and was thus not very fine-grained. The best ranking item in the knowledge base was always a case that matched the current problem exactly (ranking 1), problems from the same domain tended to cluster at the same similarity value (ranking 2), and problems from other domains tended to rank highly as well (ranking 3 and higher).

- **total cases retrieved**

A metric that examines how asynchronous retrieval contributes to the system’s performance is the total number of cases the system retrieved — in other words,



**Figure 10.4. Scaleup of Spreading Activation Effort**

how many cases had been found at the point where further asynchronous retrieval stopped providing additional cases?

- **cycles to max cases**

A second metric that examines how asynchronous retrieval contributes to the system's performance How many retrieval cycles did it take for the system to retrieve as many cases as it could? (At what point did further asynchronous retrieval not provide any additional cases?)

- **total effort expended**

Total retrieval effort can be measured by how many nodes the spreading activation system examined in the knowledge base over the entire course of

retrieval. The total effort measure illustrates the effectiveness of the cost control policy in limiting effort expended, especially when plotted against knowledge base size.

- **total time expended**

Total retrieval effort can also be measured by the amount of time it took for retrieval. The relationship of total time expended to knowledge base size is also a measure of the effectiveness of cost control.

**Results.** The experiment showed that total effort expended was largely constant across all knowledge bases, but that quality and quantity of retrieval could degrade if a knowledge base contained many irrelevant distractors (Figure 10.3).<sup>25</sup> Stepping through each of the independent variables in turn:

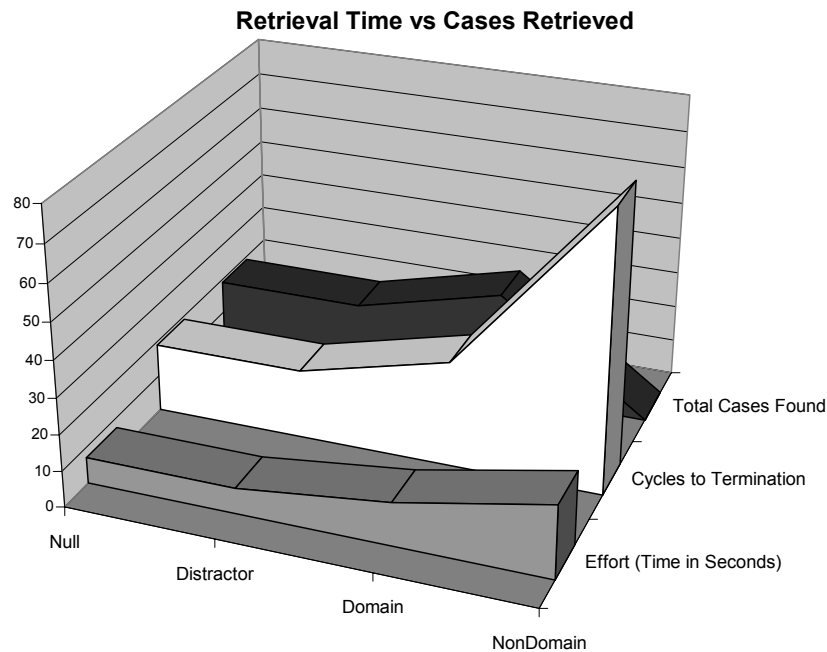
- **cycles to best guess**

First guess retrieval was almost identical for all of the knowledge bases: in the Null, Distractor and Domain conditions the system was able to retrieve a first guess in 3 retrieval cycles. In the NonDomain condition, the large number of irrelevant distractors prevented the system from finding any relevant retrievals using the amount of retrieval effort it was allowed.

---

<sup>25</sup> Note Figure 10.1 displays the number of cycles to successful retrieval, displaying failure in the NonDomain condition as zero cycles; the total number of cycles expended on retrieval (failure or success) is displayed in Figure 10.3.





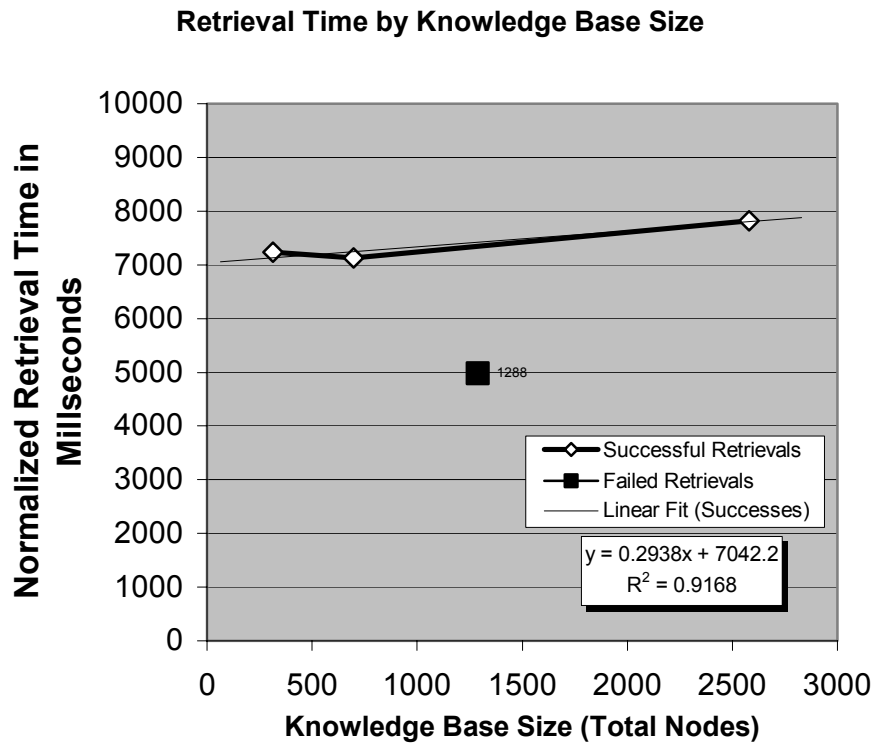
**Figure 10.5. Impact of Cases Retrieved on Total Retrieval Time**

- **quality of best guess**

The quality of the first guess retrieval was almost identical for the Null, Distractor and Domain conditions: in each, Nicole-MOORE was able to find a “high-quality” case of rank 2 from the same Stinger Missile domain as the problem statement. Retrieval broke down in the NonDomain condition, in which the large number of irrelevant distractors prevented the system from finding any relevant retrievals using the amount of retrieval effort it was allowed.

- **total cases retrieved**

The number of cases retrieved was identical for the Null and Distractor condition: Nicole-MOORE was able to retrieve the same number of cases even when the knowledge base size was doubled from 350 to 700 items by seeding in



**Figure 10.6. Scaleup of Normalized Retrieval Time**

information from another task. In the Domain condition, the number of cases increased accordingly because there were many more cases to retrieve. Again, retrieval broke down in the NonDomain condition, in which the large number of cases from other problem domains distracted the system from finding any cases from its target domain given the amount of retrieval effort it was allowed.

- **cycles to max cases**

The number of cycles to maximum retrieval was identical for the Null and Distractor condition, and increased in the Domain condition as the system harvested more cases. In contrast, in the NonDomain condition retrieval continued for many more cycles until the system gave up its attempt to find

information from the knowledge base. It is important to note that Nicole-MOORE was configured to retrieve cases serially — one at a time, exactly as retrieval was conducted in Nicole-MPA —rather than in parallel. It is possible that harvesting multiple cases per cycle could have decreased the number of cycles required to maximize number of cases retrieved in this circumstance, although Evaluation 10.9 appears to show that this would not be true in all circumstances.

- **total effort expended**

The total number of nodes expended was largely constant for all four knowledge base conditions, approximately 250 nodes. Figure 10.4 illustrates this more closely, plotting the number of nodes expended against the size of the knowledge base in each condition.

- **total time expended**

The time expended for retrieval varied primarily based on the number of cases retrieved (Figure 10.5). The total number of cycles spent in retrieval increases as additional cases are found and increases still higher when the system searched fruitlessly to find cases in the NonDomain condition. Because of this dependent relationship, it is not a fair comparison to count cycles to termination as the total retrieval time; instead, to measure the relative amount of effort spent by each approach we can normalize the retrieval time by the number of cycles each expended in retrieval. Figure 10.6 displays time to retrieval figure against

knowledge base size, normalized against the number of cycles to retrieval in the Null condition. This normalized value shows a very low linear growth of time to retrieval as knowledge base grows in size by an order of magnitude. Further experiments reinforce this measure.

Overall, the results of this experiment showed that the system provided fast best-guess retrieval to the limits of the quality of the retrieval specifications, and that asynchrony improved comprehensiveness of retrieval.

**Heterogeneity Results.** The experiment furthermore showed that while large number of “non-domain cases” degraded retrieval effectiveness, simply adding bodies of knowledge related to other tasks did not degrade performance.

The Null, Distractor and Domain conditions showed little change in knowledge base recall or performance, showing that adding new irrelevant domains of knowledge to the knowledge base, such as large chunks of the ISAAC knowledge base contained in the Distractor condition, did not affect Nicole-MPA’s ability to retrieve planning cases.

However, adding new planning domains with large numbers of irrelevant cases, as in the Null Condition, did degrade the system’s ability to retrieve planning cases. This result was interesting enough to serve as the subject for a separate experiment, Evaluation 10.7.

### **10.8.3. Evaluation 10.3: Evaluation of Scaleup II: Interactions between Platform and Scaleup.**

In addition to this experiment, larger knowledge bases were examined during the course of this research, including semantic maps of over 50,000 items in Nicole-IRIA and a large case base with over 40,000 automatically generated plans and partial plans in Nicole-MPA. A number of interesting phenomena were observed during initial exploratory tests of these knowledge bases.

**Method.** Nicole-IRIA was used for several months to serve an actual workgroup (n=3-5) as part of informal user interface studies. During these tests, conducted on a variety of Intel Pentium processor machines running Red Hat Linux 6.0 and Allegro Common Lisp 4.3, Nicole-IRIA served several users continuously for days or weeks at a time, accumulating knowledge slowly and ultimately increasing knowledge base size to more than an order of magnitude over those formally tested with Nicole-MPA. In some of these tests Nicole-IRIA's knowledge base was also "seeded" with a database of initial resources, ranging in size from a few hundred to a few thousand resources (each resource represented by an average of 30-50 nodes).

A similar test was also conducted with Nicole-MPA, in which the knowledge base was seeded with a large number of artificially generated plans and the performance of the system was monitored over time.

**Results.** These tests revealed several interesting interactions between total cost of retrieval and software platform, operating system and machine configurations.

Retrieval cost in both Nicole-IRIA and Nicole-MPA increased slowly but linearly for a period of time, then rapidly degraded. This steep “elbow” in the performance curve occurred when the host system’s main memory was exhausted and it was forced to use virtual memory. It is unsurprising that adding more memory extends the period of useful performance. It is more significant that the time of retrieval continued to grow in a linear fashion up until the memory limit was reached, no matter how much memory was available.

Garbage collection was also a factor. In both slow accumulation or and pre-seeding conditions the Nicole-IRIA knowledge base could grow in two orders in magnitude in size, from a few hundred to tens of thousands of items. Adding ~10,000 items to Nicole-IRIA’s knowledge base led to a slowdown in presentation of recommendations by a factor of three, which was judged to be unacceptable to end users. However, the bulk of this cost was attributable to frequent garbage collection as the amount of memory consumed by the system grew. Forcing Lisp to allocate more memory initially alleviated most of the problems in the accumulation condition; in the pre-seeding condition pre-emptive garbage collection improved performance considerably to the point that users were again satisfied with its performance.

While I did not conduct extensive empirical evaluations of the original Nicole-IRIA system's performance during the accumulation tests, user evaluations of the system indicated that system accuracy remained high and response time roughly constant as the knowledge base grew, until the knowledge base size ultimately overwhelmed the server's main memory and forced it to start using swap space. Furthermore, the amount of effort the system expended on retrieval was continually monitored during Nicole-IRIA's operation: the results of these tests showed that the amount of spreading activation effort the system expended on each user request was relatively constant and was independent of knowledge base size.

**Heterogeneity Results.** Nicole-IRIA successfully retrieved information of interest to a user requests without being distracted or degraded by chunks of the ISAAC and Nicole-MPA knowledge bases contained within the experience store.

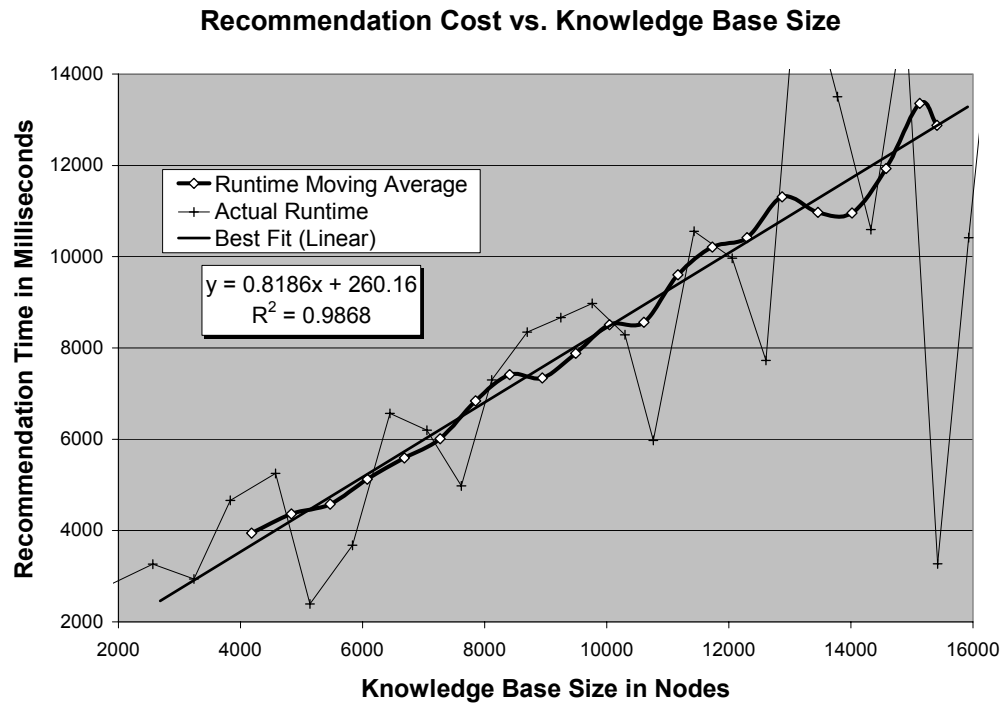
#### **10.8.4. Evaluation 10.4: Evaluation of Scaleup III: Simulated Accumulation Tests**

To follow up on these findings, I conducted an empirical evaluation that simulated the behavior of Nicole-IRIA under the accumulation test conditions. The goal of this experiment was to determine how increases in knowledge base size impacted recommendation cost as measured by both runtime and spreading activation effort. Based on Nicole-IRIA's cost control policy I expected to find that the amount of spreading activation effort expended on recommendations would remain constant; based

on the results of Evaluations 10.2 and 10.3, I suspected that the overall runtime of recommendations would increase with knowledge base size.

**Method.** The evaluation simulated a user running a series of queries and selecting the first displayed resource on each query to receive a set of recommendations. In this evaluation, Nicole-IRIA was initialized in the Null knowledge base state and was presented a series of 27 queries drawn from the Information Retrieval Problem Set. After each query's harvest cycle completed entering a new set of results in the knowledge base, the knowledge base size was recorded; then, a recommendation cycle was executed by cueing on the first result listed under the query. This recommendation cycle was timed, and then the recommendation queue was cleared and the activation state of the system cleared in preparation from the next cycle of query, harvest and recommendation.





**Figure 10.7. Increasing Recommendation Cost with Knowledge Base Size**

**Results.** The results of the experiment showed that recommendation cost was dependent on the particular query, recommendation spreading activation effort was variable but had a fixed upper bound, and recommendation time increased with knowledge base size. To reduce the impact of system factors such as garbage collection and other running processes, the experiment was run twice and the resulting times averaged; the average difference between runs was 1208 milliseconds. Recommendation cost is displayed as a function of knowledge base size in the jagged thin line in Figure 10.7.

Recommendation cost was highly dependent on the particular query. Even when Nicole-IRIA had accumulated a knowledge base of over 15,000 nodes after 20 queries, recommendation cost could drop to close to 3,000 milliseconds — or rise to almost 20,000 milliseconds.. This variance in cost can be attributed to a variety of factors: how many resources were harvested for a query, the particular recommendation cued on and its connection to other resources, and system factors such as garbage collection.

Similarly, spreading activation effort depended on the particular query and recommendation. On each recommendation cycle activation spreads from cues found on a selected resource, such as words in the title or description: because the number of cues can vary so can the amount of spreading activation. Despite this different number of cues, overall effort remained capped around 1200 nodes expanded by Nicole-MOORE's cost control policy. Note that Nicole-IRIA used a far lower activation cutoff threshold than Nicole-MPA and thus many more nodes are activated per individual cue.

Despite the variability in individual searches and recommendations, a definite trend can be seen in runtime cost. Overall recommendation cost grew from under 3000 milliseconds to just under 19,000 milliseconds over the course of the 27 queries. To reduce the impact that variation in individual queries had on visualizing the trend, I computed a moving average over 7 queries, displayed as the thick smoothed line Figure 10.5. Attempts to find a best-fit curve revealed that the relationship between knowledge base size and recommendation time can best be modeled by a linear growth curve or a polynomial curve with a low second-order constant. One such best-fit curve is displayed

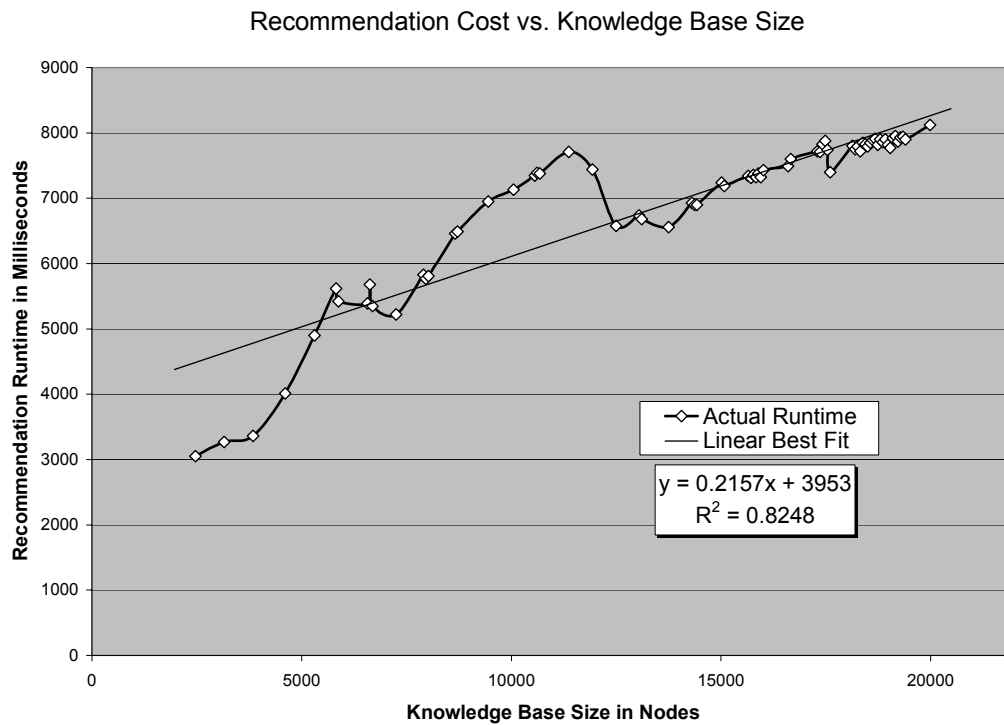
as the thin unmarked line fitting the moving average data in Figure 10.7; the equation for that curve is also displayed. Best-fit curves for both the original and moving average data were similar.

This experiment revealed that knowledge base size had an impact on recommendation cost and thus spreading activation costs; follow-on experiments are being planned with Nicole-IRIA and its successors to more precisely quantify these costs and identify their source.

#### **10.8.5. Evaluation 10.5: Evaluation of Scaleup IV: Followup Accumulation Tests**

To follow up on this follow-up, I conducted two further simulations of the accumulation tests. The goal of these tests were to reduce the impact variation in individual queries had on the accumulation results and to extend these results to a more ecologically valid set of queries.

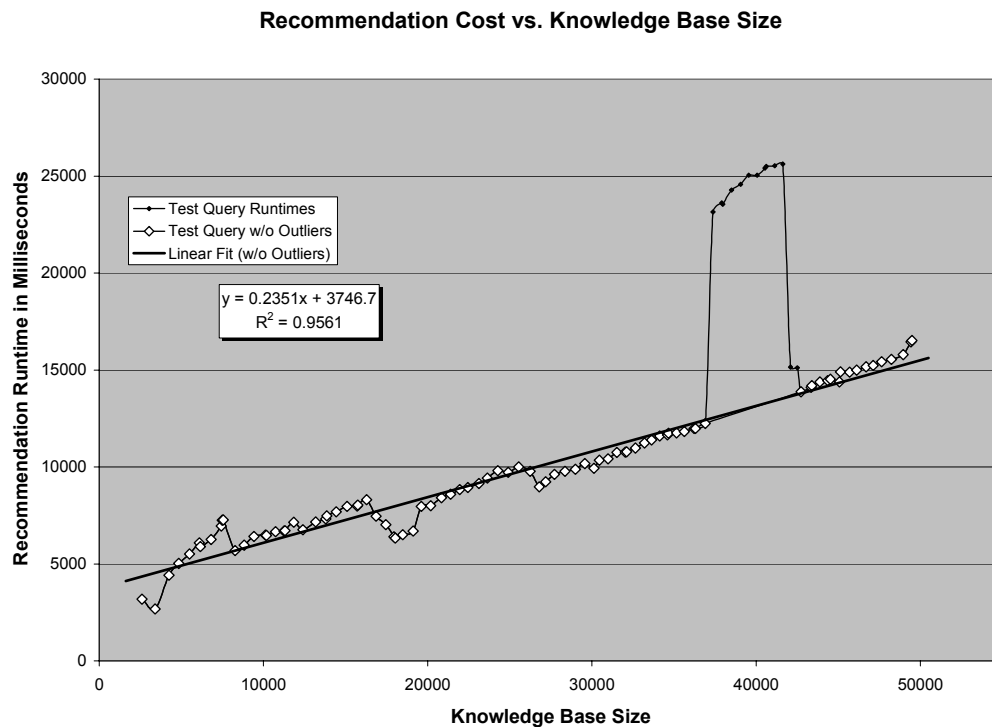
**Method.** The follow-up evaluations were similar to the previous accumulation tests with two primary differences. As before, the tests simulating a user running a series of queries and receive a set of recommendations. After each “source query” from the data set was executed, the recommendation queue was cleared and the activation state of the system cleared; then, a “test query” and recommendation was executed and timed. Because the test query and the resource cued on were fixed, measurement of the test



**Figure 10.8. Scaleup of Recommendation Cost: Randomized IR Data Set**

provided a more stable measure of the recommendation cost as knowledge base size increased because of the source queries.

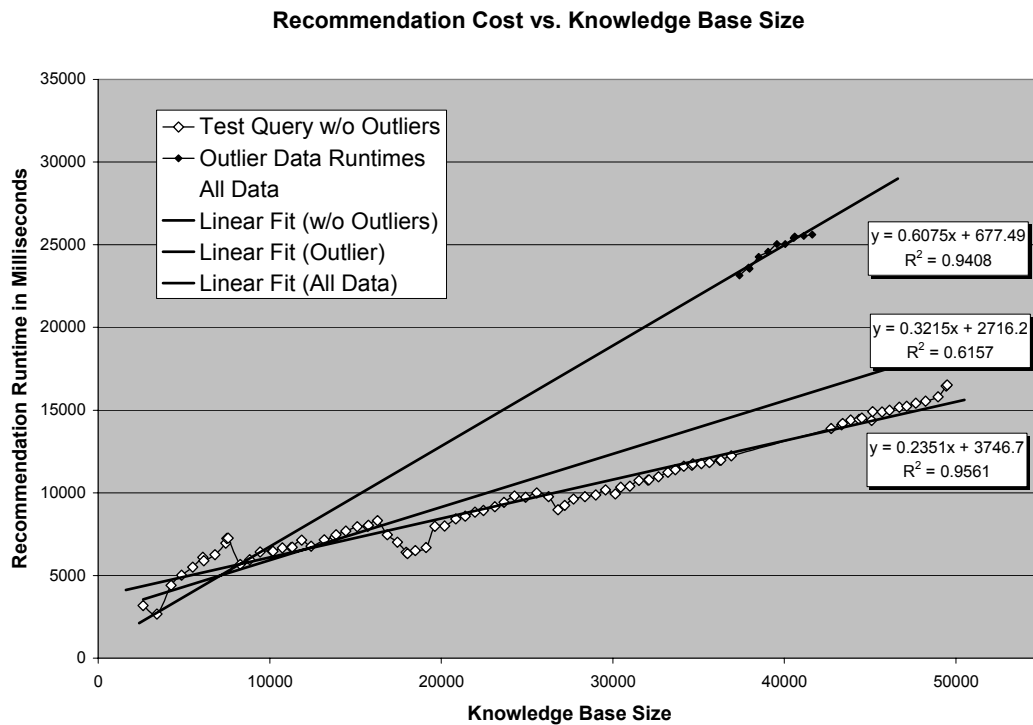
In both evaluations, Nicole-IRIA was initialized in the Null knowledge base state. In the first evaluation, Nicole-IRIA was presented with a series of 75 “source” queries drawn at random from the 27 problems in the Information Retrieval Problem Set; the test query was “centaur”. In the second evaluation, Nicole-IRIA was presented with a sequence of 100 unique source queries from the MetaSpy Problem Set; the test query was again “centaur”.



**Figure 10.9. Scaleup of Recommendation Cost: MetaSpy Data Set**

**Results.** Both evaluations showed a slow linear increase in recommendation runtime as knowledge base size increased. This growth curve was less severe than the curve detected in Evaluation 10.4 (Figure 10.7) and was similar to the growth curve found in Evaluation 10.2 (Figure 10.6). These evaluations also revealed interesting properties of both the interaction of platform properties with a system and in properties of the relevant data sets.

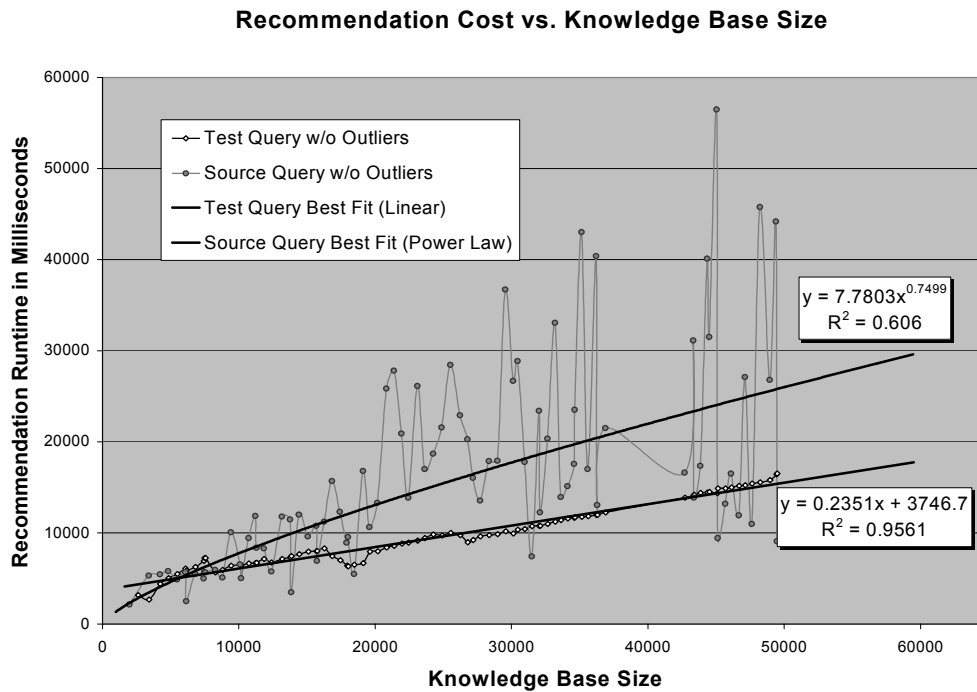
For the Information Retrieval data set, recommendation cost grew slowly until Nicole-IRIA had been presented all source queries. At that point, continued harvesting of information yielded no new resources and the size of the knowledge base essentially



**Figure 10.10. Comparison of Scaleup of Outlier and NonOutlier MetaSpy Data**

stopped growing, causing recommendation cost to similarly flatten out. Figure 10.8 displays this growth curve, along with the linear best fit trend.

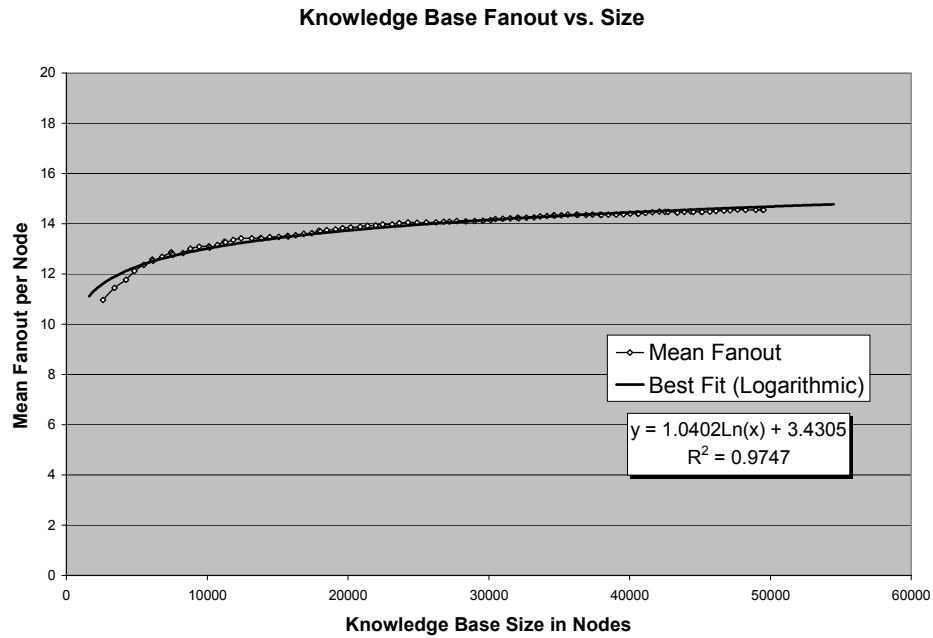
The MetaSpy problem set presented a more realistic profile of information retrieval. The size of the knowledge base continued to grow with each new query, and recommendation cost grew apace. One interesting phenomenon occurred: queries 77 through 89 showed a markedly higher cost than all other queries in the set, but nevertheless showed a similar growth curve. After query 89, the recommendation cost returned to “normal.” I hypothesized that this increased cost was due to a system process, such as a backup operating during the experimental run. Attempts to fit data to these “outliers” and “non-outliers” showed that a better fit could be achieved by considering



**Figure 10.11. Comparison of Test and Source Recommendation Cost Scaleup**

these data sets separately, and so I removed these data points from the analyses. Figure 10.9 displays the data from the MetaSpy data set, displaying normal values in white and outliers in black. The linear best fit to non-outlier data is also displayed. Figure 10.10 displays the comparative fit of the outlier, non-outlier and combined data sets.

I also recorded the recommendation runtimes for the source queries, which were considerably more variable. Figure 10.11 displays the comparative cost of recommendations of the source MetaSpy query recommendations with the test query recommendation. Note that a power law provided a better fitting curve to the MetaSpy query data than a linear fit.



**Figure 10.12. Scaleup of Knowledge Base Fanout: MetaSpy Data Set**

Because a fixed query appears to obey a linear increase whereas a naturalistic series of queries obeys a power law, I hypothesize that the power law may be a property of the data set. Like many information retrieval data sets, the Web obeys Zipf's and Heaps' laws (Baeza-Yates & Riberio-Neto 1999). Zipf's Law models the frequency of words as an inverse power law (that is, the  $i$ th most frequently occurring word has a frequency of  $Z/i^\theta$  where  $\theta > 1$ ) and Heaps' Law models vocabulary size as an increasing power law (that is, a text of size  $n$  has a vocabulary of  $Kn^\beta$  words where  $\beta < 1$ ). Heaps' law suggests that the total vocabulary of the knowledge base will increase as a power law of the number of queries, whereas Zipf's law suggests the number of unique terms in each query seen will decrease according to an inverse power law. Taken together, these regularities suggest that the amount of effort spent on recommendations over a series of

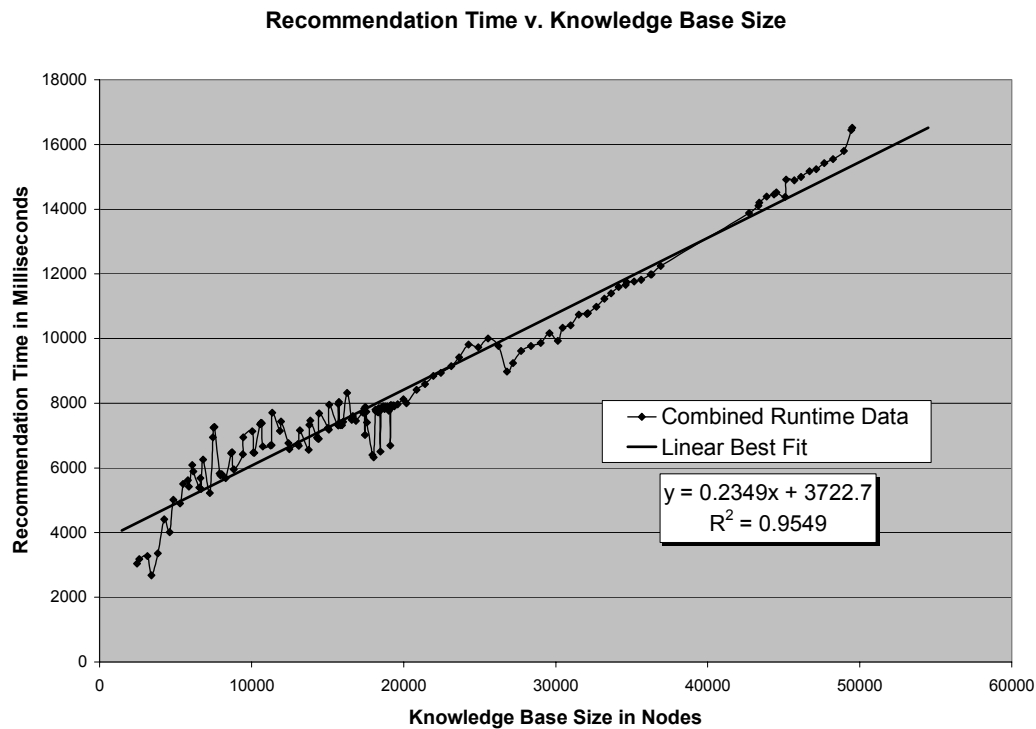


queries will increase at a decreasing rate. Other explanations are possible; for example, recommendation cost might be a function of knowledge base fanout (Figure 10.12). I am currently conducting a more detailed study to diagnose the cause of the increasing costs of recommendations and to test whether these intuitions about the properties of the system and the data set are correct.

As a further analysis of scaleup, I combined the data from the Information Retrieval and MetaSpy data sets. As shown in Figure 10.13, the two data sets show a similar increase in cost; attempting to fit a curve to the data shows a linear fit with low-constant growth similar each data set considered individually.

#### **10.8.6. Evaluation 10.6: Evaluation of Scaleup V: Effect of Seed Knowledge Bases**

The primary affect of adding knowledge to Nicole-IRIA's knowledge base appeared to be an increase in recommendation time without any noticeable change in recommendation quality. However, adding "seed" knowledge bases could affect Nicole-IRIA's recommendation performance in both positive and negative ways.



**Figure 10.13. Scaleup of Recommendation Cost: Combined Data Set**

In an exploratory series of tests, seed knowledge bases containing a variety of items were loaded into the knowledge base and used to generate “related” recommendations. This was equivalent to conducting two searches at the same time, drawing context from browsing one search to recommend information from another search.

**Method.** Two seed knowledge bases were tested for this “related recommendation” paradigm. First, a knowledge base containing approximately 900 information resources related to a single topic; second, a knowledge base containing approximately 1000 information resources related to a variety of topics. For each seed knowledge base, a set of searches was conducted from the Internet. On each search, a sequence of resources

was selected in a naturalistic browsing setting. At each click, the list of “related recommendations” was inspected to detect both changes in recommendations and the presence of relevant information.

**Results.** On the first seed knowledge base, user evaluations of quality of search revealed that the quality of related recommendations was poorer than recommendations on other searches for information. For some sets of cues Nicole-IRIA was unable to find any useful recommendations on this data set, in the worst case failing to reorganize the related recommendations list at all.

The second, structured knowledge base, showed far better search results. Under these conditions, adding additional information to the knowledge base could actually improve the quality and quantity of search results retrieved for both the related and original recommendations.

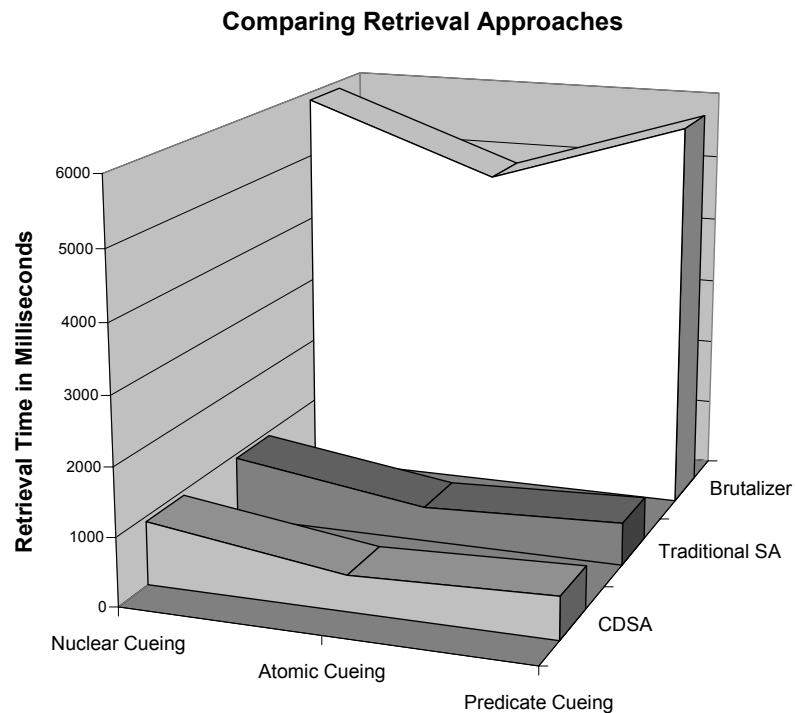
**Discussion.** An examination of the flow of activation in the knowledge base and a survey of the items stored in the knowledge base suggested two possible causes for these results. First, the addition of new items added new connections between related items, changing the patterns of flow of activation in the knowledge base; second, the addition of larger databases of items changed the available items for recall, improving the system’s ability to find some related reminding.

The results of this exploratory studies were promising, but more extensive studies are clearly required to examine in more detail the impact on larger and larger bodies of knowledge upon the scaleup of the system and the quality of recommendations. A series of experiments examining these questions is being planned on IRIA's successor system at this time.

#### **10.8.7. Evaluation 10.7: Comparative Performance Analyses**

Evaluating the performance of a context-sensitive asynchronous memory in absolute terms is difficult, given that a context-sensitive asynchronous memory has a unique set of capabilities that do not exist in any single competing system. However, while context-sensitive asynchronous memory cannot be directly compared as a whole to any other existing system, some of its novel components can.

For evaluation purposes one key component of the system, the context-directed spreading activation process, was compared with other memory search strategies, including the Brutalizer brute-force comprehensive memory search process and a traditional spreading activation parameterization of Nicole-MOORE. This section, Evaluation 10.6, addresses the brute-force comparison; the following section, Evaluation 10.7 compares CDSA with traditional spreading activation.



**Figure 10.14. Comparison of Brute-Force, Traditional and Contextual Search**

**Motivation for Evaluation.** Any spreading activation process incurs considerable space and time overhead in maintaining its active node list and spreading activation to continue its search; moreover, spreading activation is not guaranteed to return a correct result if one exists. Therefore, it is useful to compare the context directed spreading activation approach not only with other spreading activation approaches but also approaches like the Brutalizer which do not incur these overheads and which have other benefits, such as guarantees of correct results.

**Method.** The efficiency experiment tested the context-sensitive asynchronous memory approach using the same asynchronous retrieval manager and three separate

search processes: the Brutalizer, a traditional spreading activation parameterization of Nicole-MOORE, and a CDSA parameterization of Nicole-MOORE. The experiment tested each of these systems' abilities to retrieve a best-guess retrieval from the Null library for problems drawn from the X2 problem set.

The experiment used a 3x3 design, testing each of these retrieval approaches on 3 different knowledge structure strategies: predicate, atomic, and nuclear cueing. The experiment measured the following variables:

- **success rate of retrieval:**

For how many problems out of the X2 problem set was the retrieval approach able to successfully find a relevant case?

- **retrieval speed:**

What was the absolute time to find the first relevant retrieved item?

- **number of retrieval cycles:**

How many retrieval cycles (iterations of the asynchronous memory retrieval system) did it take to get a successful retrieval?

**Results.** All approaches were successfully able to retrieve cases quickly and with a high success rate; however, the Brutal approach took much longer than either CDSA or traditional SA.

As expected for the Null knowledge base and Inexperienced library, in all conditions brute-force search, spreading activation and CDSA were able to retrieve cases for all problems from the X2 library within 2 retrieval cycles. Knowledge structure strategy did not affect success rate in this condition.

However, the Brutal approach consumed much more running time than the other two approaches. Running times of each system on each knowledge structure strategy are displayed in figure 10.14. Note that the knowledge structure strategy, which includes not only different plan tagging strategies but also different matching algorithms, affected the runtime of all three approaches.

Overall, context-sensitive asynchronous retrieval produced results of the same quality as the Brutalizer algorithm, matching its performance on number of cycles and guarantee of retrieval while producing those results 600% to 900% faster than exhaustive matching. There was no statistical difference between the running time of traditional and context-directed spreading activation.

**Discussion.** These results were not a property of large knowledge base sizes. A series of exploratory tests showed that even on the smallest knowledge base available to Nicole, both traditional spreading activation and context-directed spreading activation achieved similar retrieval accuracy for almost an order of magnitude less cost than the Brutalizer.

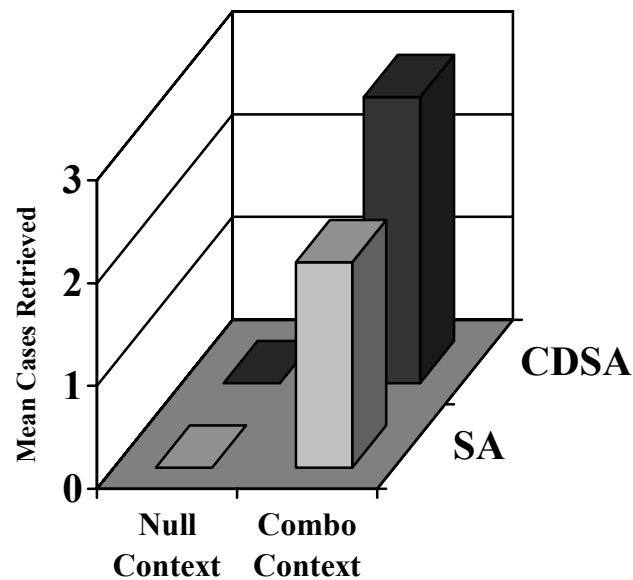
### 10.8.8. Evaluation 10.8: Contribution of Context I

While spreading activation and CDSA performed similarly on the Null knowledge base, traditional spreading activation fails to perform well on more challenging conditions such as the NonDomain knowledge base discussed in Evaluation 10.2. As discussed in chapter 3, when a knowledge base contains a large number of irrelevant distractors, we should expect that priming the system with objects found in a problem or environment and relationships relevant to the kind of task it is performing should aid the system's retrieval. A retrieval experiment was conducted to test this.

**Method.** The context experiment tested the ability of the context-sensitive asynchronous memory approach to retrieve cases for case-based planning from the heavily populated NonDomain knowledge base. This knowledge base contained many planning domains with large numbers of cases and was shown in Evaluation 10.1 to be a condition in which the system was unable to retrieve relevant planning cases without contextual guidance.

The priming of the knowledge base generated by the knowledge structure strategies was disabled for the purpose of these experiments, and priming based on the experimental contexts was simulated by propagating activation from the knowledge items in specially generated "context sets".





**Figure 10.15. Benefits of CDSA over traditional spreading activation**

The experiment used a 2×2 design, comparing retrieval for 14 problems drawn from the X2 problem set on two contextual conditions (Null Context and Objects+Relations context)<sup>26</sup> and two different algorithms (Traditional Spreading Activation and CDSA). The experiment measured the number of cases each algorithm successfully found in each contextual condition.

**Results.** The results of this experiment showed that some contextual information was required for retrieval and that CDSA was better at exploiting this information than

---

<sup>26</sup> Actually, all contextual conditions listed earlier (Null, Objects, Generic Concepts, Relations, and Objects+Relations) were tested, but only Objects+Relations differed significantly from the Null condition.

traditional spreading activation. Figure 10.15 illustrates this phenomenon. As in Evaluation 10.1, neither CDSA and SA were able to retrieve cases for X2 problems from the NonDomain knowledge base in the Null Context condition. However, when an Relations+Objects context set was provided, both CDSA and SA were able to find relevant cases, with CDSA retrieving 39% more cases than traditional spreading activation, a statistically significant difference ( $p \ll 0.01$ ). All conditions consumed the same amount of resources, consistent with the cost control policy employed by the retrieval engine.

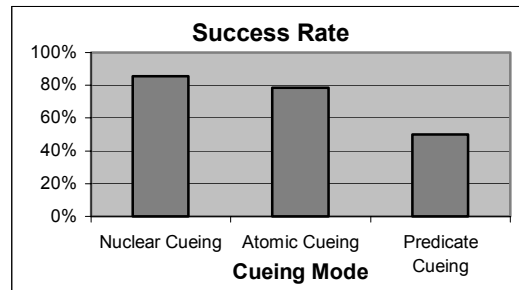
The results of the experiment showed that in a situation in which it is difficult to retrieve cases — a knowledge base containing a large number of irrelevant distractors — providing a full context (objects relating to the problem plus relationships appropriate for the domain) provided clear improvements in retrieval performance over no context and over traditional spreading activation. Exploratory evaluations showed that contexts containing merely objects or merely relations, or containing irrelevant information, were not effective in improving performance. Traditional spreading activation performance was improved by context, but not as much as CDSA: the combination of context-directed spreading activation and an appropriate context maximized the retrieval from a knowledge base.

### 10.8.9. Evaluation 10.9: Contribution of Context II

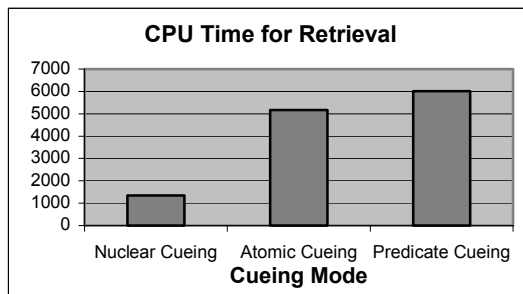
Another factor impacting the effectiveness of CDSA is the structure of the experience store. How knowledge is structured in the experience store determines how activation can spread and how context can affect that spread of activation. Nicole-MPA's knowledge structure strategies, which alter how plans are tagged in a knowledge base without altering their content, provide a way to test the impact of knowledge base structure on retrieval.

**Method.** To test the affect of knowledge base structure on retrieval, I performed another context experiment on the X2 problem set using the Inexperienced Library and the Null knowledge base. This experiment tested the ability of Nicole-MOORE to retrieve cases under different knowledge structure strategies. Unlike the last experiment, the default priming strategies of each of these knowledge structure strategies was used. This default strategy was similar to the Objects+Relations context set.

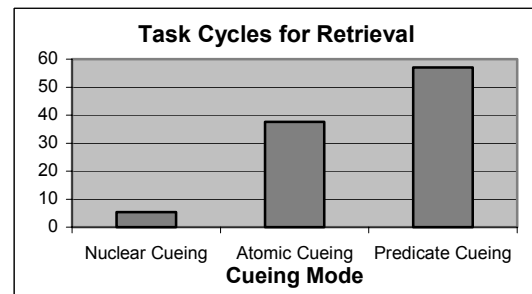
The experiment used a 3-condition design, comparing retrieval for 14 problems drawn from the X2 problem set on three knowledge structure strategy conditions: Predicate, Atomic and Nuclear. The experiment measured three variables: the number of problems for which the approach could find relevant cases, total retrieval time, and total retrieval cycles.



(a)



(b)



(c)

**Figure 10.16. Impact of Tagging Strategy on Recall, Time and Cycles**

**Results.** The Nuclear cueing strategy, which had the richest knowledge base structure of all three strategies, proved to be the most effective retrieval strategy as measured by success rate, time and cycles. Atomic was second and Predicate was third.

Nicole-MOORE was able to find relevant cases using the Nuclear knowledge structure strategy for 12 out of the 14 problems, whereas with Atomic it found only 11 and with Predicate only 7 (Figure 10.16a). The Nuclear strategy was also the least expensive in both the number of cycles necessary for retrieval and overall time for retrieval (Figures 10.16 b and c).

### **10.8.10. Evaluation 10.10: Tests of Cost Control**

A variety of tests and experiments showed that this kind of comprehensive cost control policy is necessary to ensure that performance remains roughly constant as increasing numbers of retrieval requests are added. Eliminating individual elements of the cost control policy, such as the limits on active cues or active retrievals, led to either unacceptably slow performance or poor retrieval quality.

For example, the necessity of limiting cues was demonstrated in exploratory assays of both Nicole-MPA and Nicole-IRIA. Both Nicole-MPA and Nicole-IRIA accumulate more and more contextual information about their problems as they work. In Nicole-MPA this is in the form of intermediate goal states that may provide cues about what plans are relevant; in Nicole-IRIA this is in the form of a history of user browsing actions that provide cues to relevant resources. Tests with Nicole-MPA showed allowing retrieval requests to build up an unbounded number of cues could seriously degrade retrieval performance on larger problems, and as a result Nicole-MPA adopted an explicit policy to limit the number of cues per request. Tests with Nicole-IRIA showed a similar result: allowing retrieval requests to build up an unbounded number of contextual cues both degraded performance and caused poor recommendations when user's interests changed rapidly. Based on these two experiences, Nicole's retrieval system itself was modified to limit the number of active cues per request by discarding old cues.

The necessity for limiting the number of outstanding retrieval requests was established in the exploratory accumulation tests described in Evaluation 10.3. As Nicole-IRIA continued to handle user searches, it accumulated more and more active retrieval requests. This was deliberate in that it enabled users to resume searches, conduct several searches on the same topic, and collaborate with other users; however, because the Web-based interface did not provide a way for the server to know whether a user was really done with a search or had simply browsed away for a moment with an intent to resume, Nicole-IRIA had no clear signal of when to stop processing a request. Unsurprisingly, this caused Nicole-IRIA to consume more and more resources over time. The solution was to limit the number of actively refreshed retrieval requests to some arbitrary number, reinstantiating a request if a user returned to it later.

#### **10.8.11. Evaluation 10.11: Analysis of Asynchronous and Incremental Search**

While the asynchronous and incremental properties of Nicole-MOORE could not be directly tested through ablation tests, a series of experiments and evaluations were conducted with Nicole-MPA and Nicole-IRIA to test the impact of incremental and anytime retrieval on system performance.

In addition to testing Nicole-MOORE's scaleup with respect to knowledge base size and content, Evaluation 10.2 provides an indirect way to examine incremental and anytime retrieval. The experiment tested retrieval of a planning case in Nicole-MPA and measured (a) how quickly the memory could retrieve an item (b) how good that retrieval

was and (c) whether giving the memory additional time would improve its retrieval performance. The results of the evaluation showed that Nicole-MOORE could quickly return a good-quality best guess and that the memory could retrieve more items when given additional cycles to search. The quality of retrieved items was measured by the degree of absolute match between the match specifications and the retrieved item, as well as by the ranking of that item when compared with the degree of match to the query of all other items in the knowledge base. However, this experiment did not directly evaluate the usefulness of retrieved cases with respect to the planning problem for the reasons discussed in the Case Study on Planning.

Further evaluations were conducted with Nicole-IRIA. As mentioned, Nicole-IRIA searches its knowledge base to find additional resources stored as the result of past or concurrent searches. This search occurs over many cycles, using activation from the user's ongoing browsing activity to focus search upon relevant cases. The results of the original accumulation tests showed that the initial search Nicole-IRIA's conducts in the experience store results in the retrieval of few additional cases because the search is based only on the cues available in the query. While receiving additional contextual information from users, Nicole-IRIA continued its incremental knowledge base search. Guided on the basis of cues collected over the history of several user browsing events, Nicole-IRIA was often able to find several new resources per cycle.

These evaluations of incremental and asynchronous search are encouraging but very limited: additional studies and tests are being planned with Nicole-IRIA and its successor

system to evaluate the impact of asynchronous search on the quality of information retrieval.

## **10.9. Explanation and Analysis**

Overall, these experiments showed a number of encouraging and a few discouraging results. Nicole-MOORE showed the capacity to retrieve useful information for a variety of domains; it showed the ability to exploit contextual information to improve retrieval, an ability in part dependent on its asynchronous, incremental knowledge base search. Contextual information was shown to affect the quantity and quality of retrieval and was shown to enable efficient use of spreading activation effort to find useful information even under circumstances difficult for traditional approaches.

Nicole-MOORE's cost control policy was shown to be effective in terms of the system variables that it did control — spreading activation effort — but analyses of runtime showed a significant increase in cost caused by factors outside the scope of the cost control policy. Nevertheless, these experiments and the case studies show that Nicole-MOORE's performance is good enough to support at least two different reasoning tasks and that its performance scales well up until platform limits such as available memory are reached.

To understand the impact of these results more clearly, I will now examine each of the desiderata for memory systems in turn, illustrating how the results of these



evaluations and the evaluations in the case study weigh in for the context-sensitive asynchronous memory approach as a general memory approach for intelligent agents.

### 10.9.1. Desideratum 1: Able to store a wide variety of information

A reified, grounded, bidirectional semantic network is sufficient to represent knowledge for a variety of tasks.

The range of information an architecture can store depends on the knowledge representation it uses — including both theoretical properties such as logical expressive power and practical properties such as programmability. Abstract properties of the system aside, however, the ultimate test of a knowledge representation is whether it can be used to represent information in service of a variety of tasks.

The context-sensitive asynchronous memory approach specifies a reified, grounded, bidirectional semantic network architecture to support knowledge representation. As discussed in Chapter 3 this architecture has the logical expressive power of Nth-order logic and is sufficient to represent a wide variety of information. The CRYSTAL knowledge representation instantiates the context-sensitive asynchronous memory approach's reified, grounded, bidirectional semantic network memory representation.

**Status of the Hypothesis.** Evaluation 10.1 summarizes evidence of the successful use of CRYSTAL in the case studies as well as from other work. These results demonstrate directly, through use of this knowledge representation for a variety of intelligent systems

and reasoning modules, the viability of the reified grounded semantic network approach to knowledge representation implemented in CRYSTAL. This successful use of the same knowledge representation system across a variety of tasks provides evidence that the context-sensitive asynchronous memory approach to knowledge representation is domain and task independent.

### **10.9.2. Desideratum 2: Provides a general method for accessing information**

A context-sensitive asynchronous memory can retrieve knowledge in service of a wide variety of reasoning tasks in a wide variety of domains.

Representation goes hand in hand with process. For a knowledge representation, logical expressive power is determined not by its data structures but by the methods it provides to create and access that knowledge; for example, in the CRYSTAL knowledge representation it is not possible to create a relationship between two objects that does not instantiate a higher-order relationship, even if that relationship must be defined on the fly.

For a memory retrieval system, process determines how the information in working memory can be used to find information in the memory store. Earlier, I argued that the unified nature of the experience store, the general-purpose mechanisms of context-sensitive search, and the rich matching language of asynchronous memory made context-

sensitive asynchronous memory a general approach to memory retrieval. How does this stack up against the empirical evidence?

**Status of the Hypothesis.** Evaluation 10.1 summarizes evidence of the successful use of Nicole-MOORE in the case studies and other work. These results demonstrate directly, through its support of task performance in several different applications, the viability of the general-purpose matching language implemented in Nicole-MOORE. This successful use of the same memory retrieval system across a variety of tasks provides evidence that the context-sensitive asynchronous memory approach is domain and task independent.

### **10.9.3. Desideratum 3: Scales to large multifunction knowledge bases**

Desideratum 3 can be decomposed into two closely related desiderata: that context-sensitive asynchronous memory can perform effectively on two kinds of knowledge bases: the large, and the multifaceted.

Essentially, I claim that context-sensitive asynchronous memory “scales gracefully”, retrieving knowledge guided by context in a such a way that its performance characteristics are not significantly degraded by increases in knowledge base size or addition of new domains of knowledge — where “significantly degraded” includes but is not limited to exponential time complexity, cost increases that outweigh expected benefits, or simple failure to retrieve knowledge.

#### 10.9.3.1. Large knowledge bases

Context-sensitive asynchronous memory can effectively perform retrieval on large knowledge bases.

The claim that context-sensitive asynchronous memory scales to large knowledge base sizes is based on the properties of the algorithms used for memory search, which were chosen to make their performance characteristics largely independent of knowledge base size. Results of the experiments showed that while the cost control policy was capable of controlling factors such as spreading activation effort, the actual cost of retrieval as measured by running time increased. This increase was nevertheless acceptable according to the metrics just outlined: a low linear growth up until system limits were reached.

**Status of the Hypothesis.** Evaluation 10.2 provided the first tests of scaleup: by varying the size of the knowledge base, adding more cases in the planning domain as well as adding additional bodies of knowledge in other domains such as story understanding. The evaluation varied the knowledge base in size from 300 to 2500 items, or roughly an order of magnitude. The results of this evaluation showed that the number of nodes visited was roughly constant with knowledge base size — between 246 and 254 nodes across all problems and retrieval strategies. Small variations were found when additional planning cases were added to the system, which is to be expected given that the retrieval request in question was requesting planning cases and adding cases changes the structure of the portion of the knowledge base searched; however, cost

control policies continued to operate automatically and as cases were added the number of nodes tended to slightly decrease to about 248 nodes. Adding knowledge outside the planning domain had no significant effect on the number of nodes retrieved; however, increases in knowledge base size did cause an increase in the absolute time cost of retrieval, in a roughly linear fashion with a low constant.

Evaluation 10.3 elaborated on these results on knowledge bases up to 10,000 nodes in size, showing that absolute time cost was affected by many factors, including machine performance, operating system and platform features (such as garbage collection, the size of the Lisp image, and use of virtual memory), and the implementation of the system itself.

Evaluation 10.4 began the process of formalizing the results of Evaluation 10.2's exploratory work, extending these results to knowledge bases of roughly 15,000 nodes. These results confirmed the effectiveness of the cost control policy on controlling spreading activation effort, while actual runtime increased at a low linear growth rate. Evaluation 10.4 also showed that cost of retrieval is highly dependent on the particular query and context at hand.

Evaluation 10.5 further unpacked the intuitions generated in Evaluation 10.2 on knowledge bases of 50,000 nodes in size, or roughly two orders of magnitude greater in size than the Null condition. This evaluation reaffirms that the growth rate of the running time of retrieval was low linear, and reinforcing through comparison of fixed and

variable retrieval costs the finding that retrieval effort was highly dependent on the particular query even with an operating cost control policy.

Evaluation 10.6 examined the effect of knowledge base content on retrieval quality. This evaluation showed that when rich and varied bodies of knowledge were added to the system, retrieval performance was good or even improved; however, adding many very similar items to a knowledge base could degrade retrieval performance, a result consistent with Evaluation 10.2's NonDomain condition.

When Nicole is compared against the list of desirable performance characteristics outlined earlier, it scales well up to the limits of the operating system and performance platforms upon which it operates. Accuracy on these retrievals was high: Nicole-MOORE consistently retrieved information with high relevance rankings as measured by the specifications it was given. Retrieval cost was either constant or low-ranked linear depending on the measure. Most importantly, the cost of retrieval was acceptable with respect to the expected benefit of retrieving the relevant information — Nicole-MOORE was able to find relevant cases and relevant resources in relatively small amounts of time, with less time cost than the benefit of applying a retrieved case and less time cost than manually scanning a list of resources for a given answer.

**Discussion.** The results of these tests are promising; however, further tests examining in more detail the impact of adding knowledge are clearly required. Profiling studies conducted during the original accumulation tests showed that the bulk of the increase in

memory retrieval cost in Nicole was in knowledge base access time as opposed to spreading activation time; this cost in turn was tied to the properties of the underlying Lisp implementation. While the slow linear increase in access time is potentially a concern, exploratory tests with a Java-based reimplementations of Nicole-IRIA designed to overcome the limits of the Lisp implementation have showed that space costs can be as significant or more significant than time costs for a running system — and the results of the evaluations to date seem to indicate space costs can in turn lead to time costs. A variety of tradeoffs are possible between extreme memory parsimony, constant time or linear time memory access, and flexibility. Experiments are underway on Nicole-IRIA's successor to more precisely quantify these findings.

#### **10.9.3.2. Heterogeneous knowledge bases**

Context-sensitive asynchronous memory can effectively perform retrieval from heterogeneous multi-task knowledge bases.

Context-sensitive asynchronous memory's algorithms are designed to be robust as new domains of knowledge or new items are added to the knowledge base, and the quality and quantity of retrieval should not degrade simply because new knowledge is added.

**Status of the Hypothesis.** Evaluations 10.2 and 10.6 provide evidence for the claim that the approach can easily scale to include more and more bodies of knowledge and that increased structure within large subsets of knowledge can improve performance. Evaluation 10.2 demonstrated that simply adding information outside a problem's task

domain did not affect the quality of retrieval; Evaluation 10.6 shows that if new items added within a task domain are sufficiently different, quality of retrieval is similarly unaffected. The successful use of the same retrieval mechanism for different tasks, as reviewed in Evaluation 10.1, provides indirect evidence for this claim.

**Discussion.** There are limits to how well context-sensitive asynchronous memory scales as new knowledge is added. While the CDSA algorithm does not “leak” activation to irrelevant concepts in the knowledge base, activation may flow to an “irrelevant” piece of knowledge if it is heavily interconnected with an existing domain of knowledge and uses many of the same relationships. Similarly, the context-sensitive asynchronous memory’s algorithms for memory search sacrifice comprehensiveness for near-constant-time access, using context to guide search to the appropriate portion of the knowledge base; while the anytime/incremental nature of search is robust as new items are added to the knowledge base, because of cost control search may fail when the knowledge base is full of “irrelevant distractors” — items which match the cues of a query but fail to match its specification for knowledge.

Therefore, a context-sensitive asynchronous memory should respond to a query successfully when relevant items can be found within the search space as guided by available context, and should fail in two classes of situations: first, when the matching items are very far from the terms of the query in the knowledge base and the provided context cannot effectively refocus the search scope to reach them, or second, when large numbers of irrelevant but similar items exist within the search scope and the provided



context does not provide a means to focus search on relevant items. Evaluations 10.2 and 10.6 verify this assertion, showing a retrieval quality and quantity degradation precisely when large numbers of very similar items are added to a knowledge base.

Overall, these theoretical and empirical results show that simply adding new domains of knowledge or new items to a knowledge base should not degrade a context-sensitive asynchronous memory's ability to respond to queries, except in those circumstances where adding that knowledge structures or populates the knowledge base in such a way that available context does not discriminate the new knowledge from the old. However, more extensive studies are clearly required to examine in more detail the impact on larger and larger bodies of knowledge upon the scaleup of the system. A series of experiments examining these questions is being planned on IRIA's successor system at this time.

#### **10.9.4. Desideratum 4: Manages cost of retrieval**

Problems of scalability are not limited to large knowledge bases or multiple tasks. A memory system can just as easily be overwhelmed by a large number of active retrievals, individual retrievals with complex matching questions, or too many active cues. Managing the cost of retrieval means that a system provides efficient performance under normal circumstances and has a policy in place which ensures that efficiency is maintained under unusual circumstances.

The evaluations and case studies show that context-sensitive asynchronous memory satisfies both of these desiderata: it is efficient enough to serve as a general memory

retrieval system, is superior to approaches which do not combine all of its elements, and includes a comprehensive cost control policy that ensures that retrieval remains efficient no matter how large the load is on memory.

#### 10.9.4.1. Raw Efficiency and Effectiveness

Context-sensitive asynchronous memory is efficient enough to serve the retrieval needs of a typical performance task.

Validating this claim requires examining the performance of a context-sensitive asynchronous memory with respect to the needs of actual tasks, independently from the memory's comparative performance with other approaches.

**Status of the Hypothesis.** The results of the case studies, particularly Evaluations 8.5 and 9.1, validate these claims, as do Evaluations 10.2 through 10.7. Evaluation 8.5 shows that Nicole-MOORE performs well enough to support case-based performance improvement on planning tasks in Nicole-MPA, and Evaluation 9.1 shows that it performs well enough to satisfy user desired response time in Nicole-IRIA. Evaluations 10.2 through 10.6 show that this retrieval performance degrades slowly with increased knowledge base size, meaning that as the system will continue to provide “good enough” performance as it learns. Nicole-MOORE performed well on other metrics also: for example, the quality of retrieved items was sufficient to support the reasoning of the ISAAC story understanding system.

#### 10.9.4.2. Comparative Efficiency

Context-sensitive asynchronous memory performs as well as or superior to existing approaches that do the same task (in other words, competing memory systems with the same functionality).

**Status of the Hypothesis.** Evaluations 10.7 and 10.8 showed that Nicole-MOORE could perform as well as or better than memory retrieval systems using existing approaches to memory search but designed to provide the same capabilities. Evaluation 10.7 showed the context-sensitive asynchronous memory approach to be more efficient than brute force search despite spreading activation overhead, and showed that context-sensitive asynchronous memory and traditional spreading activation are roughly equivalent in cost. Evaluation 10.8 showed that context-sensitive asynchronous memory could provide improved performance over traditional spreading activation given appropriate contextual information.

#### 10.9.4.3. Effectiveness of the cost-control policy

Context-sensitive asynchronous memory's cost control policy ensures that memory retrieval performance remains efficient and effective even when a system is faced with large or numerous retrieval requests.

Nicole-MOORE employs a variety of cost control policies along the lines of those discussed in Chapter 5, including limits on the complexity of the cues of any individual retrieval ( $R_{cuemax}$ ), limits on the number of propagating nodes ( $cycle_{total}$ ), and of course the standard CDSA limits on the number of nodes that can be activated by any individual cue

or specification. While support exists in Nicole-MOORE to place limits on the number of active retrievals ( $R_{max}$ ), controlling the number of active retrievals in practice has been done through the implemented reasoning tasks.

**Status of the Hypothesis.** Evaluations 10.2 through 10.5 showed that while Nicole-MOORE's cost control policy was successful at controlling the explicit costs of retrieval that it managed, the cost of retrieval as measured by absolute runtime nevertheless increased with increasing knowledge base size. While this increased cost was a low linear growth acceptable for many applications, it nonetheless shows two things: first, when designing a cost control policy it is important to explicitly account for all the types of cost that must be controlled and to devise a policy to deal with them; second, it shows that the design of the implementation of a system can have a critical effect on the performance that can impact a cost control policy.

#### **10.9.5. Desideratum 5: Preserves accuracy of retrieval**

Managing the cost of retrieval is important to prevent difficult or numerous questions from causing excessive resource *consumption*, but it does not directly address the problem of coping with tight resource *bounds*. One way to deal with this problem is to trade off efficiency and effectiveness.

Ideally, the best way to cope with tight resource bounds is to ensure that effective performance remains efficient no matter how large or complex the knowledge base becomes. However, raw efficiency is not always possible when given incomplete

specifications of knowledge and large knowledge bases to search. A system that effectively manages retrieval should be able to act in an anytime fashion, returning possibly inaccurate best guesses quickly or better retrievals given additional processing effort.

Context-sensitive asynchronous memory meets this requirement; it performs knowledge base search incrementally as it is provided resources and is capable of trading efficiency and effectiveness in an anytime fashion.

Context-sensitive asynchronous memory can search a knowledge base incrementally, expending resources in parallel with other reasoning processes.

Both case studies provide evidence for this claim: Nicole-MPA processed asynchronous memory retrievals in parallel with task processing and Nicole-IRIA processed information retrieval requests in parallel with user browsing of resources.

Context-sensitive asynchronous memory can act as an anytime retrieval system, improving quality and/or quantity of retrieval over time while at any time retaining the ability to return the best result it has found so far.

**Status of the Hypothesis.** Evaluations 10.2, 10.9 and 10.11 provide evidence for this claim. Evaluation 10.2 shows a direct relationship between asynchronous retrieval and quantity of information retrieved; however the timing of retrieval was directly tied to the

number of serial retrieval cycles. Evaluation 10.9 shows that Nicole-MOORE's asynchronous retrieval ability is not simply a factor of its serial retrieval policy but instead that asynchronous retrieval can extend over many cycles based on the particular way activation spreads into a knowledge base. Evaluation 10.7 shows that the performance of a context-sensitive asynchronous memory can approach an optimal brute-force approach on an appropriate knowledge base; Evaluation 10.8 shows that the quantity and quality of search is superior traditional spreading activation. Finally, Evaluation 10.11 reviews evidence showing that asynchronous search can help Nicole-IRIA improve the comprehensiveness of its responses to user queries.

#### **10.9.6. Desideratum 6: Exploits task/environmental information**

Context-sensitive asynchronous memory can heuristically adjust its search based on context (information not contained within the content of a query) to improve its retrieval performance.

**Status of the Hypothesis.** Evaluations 10.8, 10.9 and the information retrieval case study provide the most direct evidence of Nicole-MPA's ability to exploit task and environmental information to improve performance. Evaluation 10.8 directly shows that providing additional contextual information can improve a context-sensitive asynchronous memory's performance over both traditional spreading activation and non-contextual conditions. The information retrieval case study backs up these claims, showing the contribution of context to improve retrieval in Evaluation 9.1 and also showing the contribution of context from multiple users to find additional relevant

information in Evaluation 9.2. Evaluation 10.9 reinforces these results, showing that the combination of contextual information and knowledge base structure can also improve retrieval over conditions with impoverished context or structure.

#### **10.9.7. Desideratum 7: Exploits extra resources if available**

Context-sensitive asynchronous memory can improve the comprehensiveness and accuracy of retrieval if given additional resources to continue search.

The flip side of the coin of preserving accuracy in the face of resource bounds is the ability to exploit additional resources to improve retrieval.

**Status of the Hypothesis.** Evaluations 10.9, 10.2, 8.6 and 9.3 provide evidence that a context-sensitive asynchronous memory can use additional resources when available.

Evaluations 10.2 and 10.6 show that Nicole-MOORE can return more planning cases when given additional cycles to search. Nicole-MOORE was able to retrieve good matching items in memory in a few cycles and given additional time to allow anytime/incremental retrieval to complete could retrieve a large proportion of all matching cases from a heterogeneous knowledge base containing a wide variety of relevant and irrelevant items. Moreover, the amount of time taken for retrieval was far less than the potential savings afforded by using the retrieved cases, an analysis that was backed up by an actual performance improvement demonstration. Evaluation 8.6 shows

that exploiting additional resources to obtain additional retrievals was essential for the performance improvement demonstration, which depends on the incremental memory search to find additional relevant cases in the knowledge base.

Furthermore, Evaluations 9.2 and 9.3 show that Nicole-IRIA can exploit additional resources and contextual information to find additional relevant information resources.

#### **10.9.8. Desideratum 8: Potentially interleavable with reasoning**

Context-sensitive asynchronous memory can detect when a suitable retrieval is found and alert reasoning without being explicitly polled.

In order for a memory system to be interleavable with reasoning, it needs to be able to operate independently: it must have the capability to identify what it believes to be a good retrieval and alert memory when they are found. This function is the job of the “alert mechanism,” a joint component of the memory and agent architecture. The reasoning modules for the context sensitivity and performance improvement experiments were configured to await responses from memory delivered via the alert system without ever explicitly polling.

**Status of the Hypothesis.** The alert system was indirectly tested the case studies. In Nicole-MPA; if the memory’s alert system is disabled, a variety of tasks the system performs simply fail to function, including the context sensitivity and performance



improvement demonstrations. The system was exploited only indirectly in Nicole-IRIA, which deliberately searched for any new retrievals on each user browsing cycles.

#### **10.9.9. Desideratum 9: Provides guidelines for reasoning integration**

Experience-based agency provides guidelines for constructing reasoners that interoperate with context-sensitive asynchronous memory and can integrate asynchronous retrievals into their current processing state.

Experience-based agency provides guidelines on how to build agents that incorporate reasoners and memory retrieval systems that operate in parallel. Nicole-IRIA and Nicole-MPA are the two major systems that integrate asynchronous retrievals into their current processing state along these provide guidelines provided by experience-based agency. However, it is difficult to evaluate how well these guidelines satisfy the desideratum because of the large proportion of task-specific processing and knowledge necessary to actually implement a system following these guidelines. Ultimately, the properties of the task and the qualities of the system constructed to solve that task are the largest determinants of the effectiveness of reasoning integration strategies, and therefore to adequately test the guidelines provided by EBA for reasoning integration it would be necessary to construct a wide variety of systems and conduct extensive evaluations. This is the approach taken by the Soar group (Newell 1990), which continues its efforts to test and extend Soar to new environments (e.g., Gratch 2000).

**Status of the Hypothesis.** Reasoning integration was tested in the case studies. Nicole-MPA includes an extensive integration mechanism which under certain conditions can successfully integrate asynchronous retrievals to solve problems otherwise unsolvable by normal planning processes. Reasoning integration was tested to a lesser extent in Nicole-IRIA, which uses portions of the integration mechanism framework to eliminate irrelevant retrievals and improve recall precision. The results of these studies showed that the most critical part of reasoning integration is retrieval evaluation: in both Nicole-MPA and Nicole-IRIA required examining retrievals for fitness and excluding unworthy retrievals in order to achieve good performance. The studies also show that when reasoning tasks have complex constraints on what can and cannot be incorporated, robust relevance determination and integration processing are necessary to ensure that retrievals are correctly integrated into the current reasoning state.

## **10.10. Analysis of Sources of Power**

The results of the evaluations conducted to date provide strong evidence that context-sensitive asynchronous memory approach satisfies the desideratum for a general memory retrieval system for intelligent agents that can exploit task or environmental information to efficiently find good answers from large knowledge bases in response to poor questions.

However, simply demonstrating an implementation that displays an element of functionality that satisfies a desideratum is not enough to show that the approach the

implementation instantiates is responsible for that success. The possibility always exists that the implementation achieves the functionality in a different way, implements the functionality successfully only for a few test cases, or that the test cases themselves have properties which are responsible for the success.

Ideally, to verify that an approach is a successful one, an implementation that instantiates the approach should be analyzed to determine how the system achieves its functionality, in order to identify the sources of power of the implementation and verify that the implementation behaves as the model predicts.

There are a number of important sources of power in the context-sensitive asynchronous memory approach, including:

- **Structural/functional constraints on the knowledge base:**

The context-sensitive asynchronous memory model predicts that properties of the experience store, such as groundings and bidirectional relations, are key sources of power for enabling memory retrieval and that ablating them will degrade retrieval.

- **Content of the knowledge base:**

The model also predicts that the specific content of the experience store, particularly the number and type of connections between knowledge items, will also affect retrieval. Rich connections between observable features and target

retrieval items based on relationships which are specific to particular tasks performed will enable more effective retrieval.

- **Content of the context:**

Furthermore, the model predicts that the particular contextual information provided will also affect retrieval. Contexts which specify objects seen and relations which connect those objects to potential targets should be most effective at guiding activation to relevant targets.

- **Context-directed spreading activation:**

The model relies on the CDSA algorithm to exploit structure, content and context to improve retrieval. Ablating this algorithm should degrade retrieval performance.

- **Asynchrony:**

The model relies on asynchrony to enable CDSA to continue to search the knowledge base, reacting to contextual information as it arrives to help find relevant retrievals.

- **Interleavability:**

The model also relies on interleavability to harvest from reasoning tasks the contextual information that CDSA needs to guide its search.

- **Integration mechanisms:**

Finally, the model relies on elements in reasoning tasks which can respond

appropriately to asynchronously retrieved items, only responding to items which are relevant to the current reasoning context.

The following sections review the evaluations conducted so far, examining what these evaluations reveal about the contributions of these sources of power.

### **10.10.1. Structural/functional constraints on the knowledge base**

One key source of power in experience-based agency is the structure of the experience store. At its core, the theory of retrieval in context-sensitive asynchronous memory is a content-addressable memory without explicit indexing. While the context-sensitive asynchronous memory approach can support knowledge bases containing indices to support specific tasks,<sup>27</sup> the general-purpose retrieval algorithm depends on tracing backwards from features and concepts in a query to target concepts which may contain or are related to them, without depending on any explicit indexing structure.

Thus, the context-sensitive asynchronous memory approach predicts that two key properties of the experience store —bidirectional links and grounded concepts — are key sources of power for memory retrieval. If the experience store does not support

---

<sup>27</sup> For example, the ISAAC system, built on top of Nicole's memory system, explicitly tags items in its knowledge base with functional indices. Also, if the important content of a knowledge item is opaque, as in the case of the planning cases in Nicole-MPA, knowledge items may be tagged with information which act like the labels used in an indexing scheme; however, these tags are used to retrieve information in a content-addressable fashion.

bidirectional links, the memory system cannot use information in the current context back to find items in the knowledge base that refer to similar information. Even if bidirectional links are supported, if the experience store is not grounded in concepts likely to be found in the current environment, then the memory will be unable to find any information in the current context useful for retrieving prior knowledge.

To test this hypothesis, I conducted a trial set of retrieval experiments using an early version of a knowledge base for the ISAAC system using a simple knowledge representation scheme. In these tests, the memory retrieval system failed to retrieve any information for simple queries even when several targets were present in the knowledge base. Analysis revealed that the cause of this problem was that the knowledge representation did not enforce bi-directional links. Because of this flaw, activation could not spread backwards along links from targets to the concepts which those targets referenced, effectively making content-addressable retrieval impossible. This confirmed that a knowledge representation sufficient for performing a reasoning task was not sufficient for the memory retrieval strategy proposed by this approach; additional constraints were required.

Based on these initial trials, I developed the improved knowledge representation now called CRYSTAL, which automatically maintains the bi-directional links and concept grounding necessary for content addressable memory and the reified relationships necessary for context-directed spreading activation, without need for intervention by a reasoning task designer or the reasoning task itself.

Because CRYSTAL's link maintenance system is tightly integrated into the knowledge representation and ablating it would break virtually all reasoning tasks which use the new representation, I did not ablate the link maintenance system in an explicit experiment. However, the CRYSTAL knowledge representation has been successfully used as part of a wide range of retrieval experiments, including a repeat of the original experiments which defeated the original memory retrieval system using the original representation. Using the new representation and a retrieval algorithm essentially identical to the original, Nicole was able to successfully retrieve information that was impossible to retrieve using the original representation. All the evaluations discussed in these and other sections depend on this updated representation.

The placeholder system is also tightly integrated into the knowledge representation, so I did not ablate grounded concepts in any explicit experiment. However, a similar trial set of experiments were conducted on an early version of the CRYSTAL system in which the placeholder system was not yet operational. While some data could be retrieved using this representation, the memory system was only able to find frames based on the abstract relationships of one node to another and not based on "perceptual" features such as numerical or character data. While frames could be matched based on low-level data, the memory retrieval algorithms could not use this information to activate frames because this low-level information was essentially invisible to the memory retrieval system. Subsequent development of the placeholder system, which automatically creates knowledge-level wrappers for program-level data, effectively provided the "grounding"

necessary to allow the memory retrieval system to retrieve information based on any data referred to by a frame, rather than just the abstract structure of knowledge within the knowledge base.

While the reified relations system is also tightly integrated into the knowledge representation, making explicit tests of reified relations by ablation difficult, the contribution of reified relations is expressed in retrieval through the CDSA algorithms, which can be parameterized to explore the contribution of reified relations explicitly. Evaluation 10.8 shows that CDSA can exploit gated spreading activation propagating along active reified relations to find objects in memory more effectively than traditional spreading activation can given the same amount of retrieval effort.

### **10.10.2. Content of the knowledge base**

One major focus of this thesis has been that using additional information from the task or environment to improve retrieval performance is an effective way to perform retrieval under resource bounds. This is a heuristic strategy: there is no guarantee that it will result in improved performance. Therefore, it is fair to ask under what conditions is this context-sensitive memory search strategy likely to result in benefits? In other words, for what kinds of knowledge base structure will providing an appropriate context would improve retrieval performance, and what would that context look like?

The evaluations show that the structure and content of a knowledge base is a key source of power; without both relevant cases and data that relates typical content to cases



the memory system cannot retrieve information. This has been demonstrated through several evaluations of Nicole-MPA and Nicole-IRIA:

- **Nicole-MPA: Knowledge labels for opaque content necessary for retrieval:**

Because plans in Nicole-MPA are opaque data structures, it is not possible for the context-sensitive asynchronous memory to retrieve them based on their initial or goal conditions without “plan tags” which make that information visible at the knowledge level. Evaluation 10.9 showed that richer structure in plan tags enabled more effective information retrieval.

- **Nicole-IRIA: Appropriate word structure necessary for retrieval:**

To Nicole-IRIA, a URL is almost as opaque as a plan is to Nicole-MPA; IRIA cannot “read” a URL and guess its meaning. In order for IRIA to be reminded of a page by user activity, considerable structure in the knowledge base is necessary, including summaries of pages stored in the knowledge base and the labels on links that users click on. Without this knowledge, the system is unable to find and recommend useful pages and simply cannot function.

- **Nicole-IRIA: Appropriate content in knowledge base needed:**

Furthermore, the content of this structure must make appropriate connections between context and pages to enable retrieval. Evaluations 9.1 and 10.6 showed Nicole-IRIA’s capability to retrieve information varies with different domains of knowledge, showing that the content of the knowledge base affects quality of

retrieval. Evaluations 10.4 and 10.5 further illustrate the impact that the varying content of resources and result sets can have, this time on Nicole-IRIA's speed.

### **10.10.3. Content of the context**

The evaluations also show that the content of the context generated by a system is a key source of power; without the right data in the context the memory system cannot retrieve information even if the right data and data structure exists.

Evaluations 10.8 and 10.9 show that the right context is critical for resource-bounded retrieval of knowledge under difficult circumstances in Nicole-MPA. The most effective context was one that combined both objects observed in the environment and relations which might connect those objects to answers to questions; the combination of these contextual items served to channel the flow of spreading activation to potentially relevant items in the knowledge base.

Evaluation 9.1 showed that within individual domains of knowledge the particular concepts users were interested in — and hence the content of their context histories — affected the quality and quantity of relevant pages Nicole-IRIA could find. More work is necessary to precisely quantify the properties of concepts and contexts that affect retrieval both positively and negatively.

#### 10.10.4. Context-directed spreading activation algorithms

Another source of power is the context-directed spreading activation algorithm itself: while traditional spreading activation provides some context-sensitivity, context-directed spreading activation can provide more sensitivity and efficiency.

- **Nicole-MPA:** Evaluation 10.8 showed that the use of context-directed spreading activation over traditional spreading activation could improve the quantity of retrieved items by up to 30% in difficult retrieval circumstances on Nicole-MPA. Using a strong CDSA parameterization and providing an objects+relations context enabled both CDSA and traditional spreading activation to retrieve relevant cases out of a large number of irrelevant distractors, but CDSA was more effective at exploiting this context.
- **Nicole-IRIA:** Evaluation 9.5 showed that altering the parameters of context-directed spreading activation to increase its context gating improved both the quality and speed of retrieval for large knowledge bases on Nicole-IRIA. Increasing context gating enabled Nicole-IRIA to focus its spreading activation effort more closely on the representations of information resources most relevant to user's information needs and provided higher quality recommendations.

#### 10.10.5. Asynchrony

Context sensitivity in turn depends on asynchrony to provide the contextual information necessary to improve retrieval. Asynchronous retrieval also enables other

capabilities in an experience-based agent, such as improved retrieval performance using extra resources and new styles of interaction between reasoning and memory.

- **Nicole-MPA:** In Evaluation 8.5, the performance improvement demonstration, solving the challenge problem depended on asynchronous retrieval enables the memory system to collect all the necessary cases.
- **Nicole-IRIA:** Evaluation 10.11 reviews work that asynchronous retrieval enables the memory system to improve its performance by searching memory more exhaustively. Evaluation 9.1 demonstrates the contribution of asynchrony to context, enabling users to continuously update the retrieval context. Evaluation 9.2 showed that asynchronous retrieval enabled new retrieval modes such as multi-user collaboration.

**Discussion.** It is possible that other features of the system are more responsible for the system's performance in these two instances than pure asynchrony; an interesting approach would somehow disable the asynchronous retrieval system to more fully test its contribution.

Exploratory tests were conducted with ablating asynchrony in Nicole-MPA; however, it was difficult to develop any meaningful evaluation of the performance of asynchronous or anytime retrieval based on ablation studies. The reason is that the easiest ways available to disable the anytime/incremental retrieval system — stopping retrieval immediately after the first cycle, or retrieving everything at once until activation in the

memory system propagates to quiescence — break most of the tasks implemented in Nicole.

The “single-cycle” option undermines the ability of the memory to provide the information it needs and thus simply breaks most tasks that use the memory system. For example, limiting retrieval to one cycle in Nicole-MPA would undermine the context sensitivity and performance improvement demonstrations discussed in later sections. The performance improvement demonstration in particular depends on the incremental memory search to find relevant cases in the knowledge base and breaks if this capability is eliminated. Nicole-IRIA also exploits incremental retrieval; its ability to combine results from multiple searches or multiple concurrent users depends on the incremental search capability.

The “search until quiescence” option also has problems, causing the system to revert to a more “traditional” mode of memory retrieval which is more expensive; this too breaks the assumptions underlying most of the applications implemented in Nicole. For example, search to quiescence would undermine the performance improvement demonstration because of the additional costs of repeatedly querying the memory each time that a new goal opens.

#### **10.10.6. Interleavability**

Asynchrony requires that the overall memory system be interleavable with other reasoning processes. Interleavability is difficult to evaluate because it is not a “module”

that can be ablated: it cuts across a wide variety of aspects of the system from the global working memory to the interface of the memory retrieval request system to the stepping functions of the Nicole memory cycle. However, while these features cannot easily be “turned off”, they can be traced through the system. Interleavability is required in Nicole-MPA to enable the planning process to communicate with context-directed spreading activation, and in Nicole-IRIA for both interaction between a user’s browser and memory and for IRIA’s multi-user collaboration mode.

#### **10.10.7. Integration mechanisms**

Finally, making use of spontaneous retrievals depends on integration mechanisms that can incorporate them into the current reasoning state.

- **Nicole-MPA:** In Nicole-MPA, integration mechanisms enable the planner to incorporate additional information, putting cases found after initial retrieval together into a solution for improved performance. Evaluation 8.5 depends on the reasoning integration mechanisms to provide its improvement.
- **Nicole-IRIA:** In Nicole-IRIA, in contrast, integration mechanisms enable an information retrieval system to reject information, eliminating recommended sites which do not match user’s search criteria. Exploratory work in Evaluation 9.2 and Evaluation 10.11 showed that eliminating highly active but poorly matching results was key to the presentation of useful information in IRIA.

## 10.11. Lessons Learned

These evaluations, taken together with the results of the case studies, teach several important lessons about how to apply a context-sensitive asynchronous memory. The most important of these lessons are:

- **context-sensitive asynchronous memory can be applied to many tasks :**

The evaluations presented here and in the case studies showed that the context-sensitive asynchronous memory approach could provide effective retrieval for many different tasks. The rich knowledge representation and flexible matching and retrieval system of context-sensitive asynchronous memory, which were designed from the outset to be general, are the key enabling factors which make the approach easy to apply to many different problems.

- **rich knowledge base structure is necessary for context-sensitive retrieval:**

As several different evaluations show, a rich representation of knowledge within the store, including many different connections of different types between pieces of knowledge, is a critical factor enabling the context-sensitive asynchronous memory approach to effectively exploit context to provide efficient, effective, quality retrieval.

- **context should be designed to exploit gated spreading activation:**

The CDSA algorithm depends on not only a rich structure in a knowledge base but also appropriate contextual information that can help it guide its search to the correct items. Any approach to providing context, either explicit or implicit, must

make a commitment about the content of that context, whether contextual cueing happens explicitly as in Nicole-MPA and Nicole-IRIA or implicitly as a result of situation assessment processes and an active working memory. For context to be most effective, it must mirror the structure of the knowledge base relevant to the current task, activating both observed items in the task or reasoning context *and* the relations that exist between those observed items and potentially relevant past information.

- **many similar but potentially irrelevant items can degrade retrieval:**

Adding large numbers of items to a context-sensitive asynchronous memory system by themselves will not degrade retrieval quality; however, if those items are poorly discriminated, the resource-bounded nature of context-sensitive asynchronous memory retrieval will cause retrieval performance to degrade. This occurs for two reasons: first, because too many irrelevant items are available, causing relevant retrievals to decay before the memory finds them; second, because the fanout from cue and specification terms becomes too high, damping activation before it has a chance to activate relevant items.

- **context sensitivity can help combat the effects of many similar items:**

Despite the fact that many similar items can degrade retrieval, an appropriate context operating over a well-structured knowledge base can enable CDSA to still find relevant items, although it will find fewer items than it can when items in the knowledge base are better discriminated.



Finally, there was one important lesson learned about how to construct context-sensitive asynchronous memories:

- **cost control policies must take into account system factors to be effective:**

The cost control policy employed by Nicole-MOORE limited the number of active cues, the amount of spreading activation effort, and even the computation performed during spreading activation. However, the time cost of retrieval continued to increase as knowledge bases grew in size. While the ultimate cause of this increase is still being investigated, it suggests two things. First, an effective cost control policy should not be restricted to controlling theoretical properties of the architecture but instead should account for and attempt to control system functions such as time cost, perhaps by further limits on theoretical factors. Second, the design of a context-sensitive asynchronous memory should be done carefully to examine the impact design decisions have on factors such as time and space cost, in an attempt to eliminate in advance potential sources for increasing costs.

## **10.12. Conclusion**

The results of the feasibility study, along with the results of the case study, show that context-sensitive asynchronous memory is an effective approach to getting good answers from large knowledge bases given limited resources and poor questions. The results

further show that the experience-based agent approach to constructing systems that exploit context-sensitive asynchronous memories is an effective one.

The results reveal caveats about the context-sensitive asynchronous memory approach, particularly in performing retrieval with many similar distracting items, controlling costs over large knowledge bases, and fully exploiting context sensitivity and asynchronous retrievals when reasoning integration is complex. However, these caveats do not outweigh the benefits of the approach, which proved itself to be general, flexible, and capable of exploiting small amounts of naturally occurring information to improve retrieval quality and quantity.

# PART IV

## IMPACT

---

### *The Contributions of the Theory and its Future Prospects*

Context-sensitive asynchronous memory has many benefits. It leverages limited amounts of information to find more useful information. It enables reasoners to improve their performance. Most importantly, it is an approach which can be readily applied to many different tasks.

Chapter 11: Conclusions discusses these benefits, reviewing the primary claims of the research, the lessons learned during the evaluation process, and the technical contributions the context-sensitive asynchronous memory model makes to artificial intelligence and other fields. The chapter concludes by pointing the way for future research, suggesting not only how context-sensitive asynchronous memory can be applied or extended but also how the philosophy of this research program can be continued towards the ultimate prize, constructing a complete general intelligent agent.

# CHAPTER XI.

## CONCLUSIONS

---

*What this research adds to the field.*

This dissertation presented context-sensitive asynchronous memory, a model of memory retrieval that solves the problem of retrieving useful answers from large knowledge bases given under-specified questions. This problem is important because it arises in the construction of general intelligent agents and many other tasks; my solution is significant because it exploits information naturally present in the task or environment that generated the question, and does so in a general way applicable to many tasks.

While the context-sensitive asynchronous memory solution requires no additional information about the task, it does require additional processing on the part of reasoning systems that use context-sensitive asynchronous memories. As a template for the construction of reasoners based on context-sensitive asynchronous memories, this dissertation presented experience-based agency, an agent architecture built to enable the productive exchange of information between autonomous memory and reasoning systems.

To achieve these benefits, context-sensitive asynchronous memory and experience-based agency contain a suite of cooperating functional components, each of which has novel features which enable the approach or enhance its functioning, such as the reified

retrieval request queue processor that enables asynchronous memory retrieval and the context-directed spreading activation algorithm which enhances context-sensitive search on large knowledge bases.

The value of the approach was demonstrated by applying it to a variety of reasoning tasks. Most importantly, the Nicole-IRIA system showed that the approach is useful to real tasks, while Nicole-MPA and ISAAC demonstrated the sources of power and generality of the approach. Taken as a whole, these experiments, applications and case studies confirm the usefulness of the context-sensitive asynchronous memory approach and the sources of power behind the theory, but both confirmed some claims about context-sensitive asynchronous memory and how it could be applied while disconfirming others. The lessons learned during these experiments illuminated the strengths and weaknesses of the approach, help outline the contributions the approach makes to artificial intelligence research, and suggest avenues for future research.

The following sections discuss the impact of the approach and what the approach suggests for others trying to apply this research to their own problems by reviewing the scope and structure of the model proposed in this work, the experiments conducted and their results, and then discussing the lessons this research teaches.

## 11.1. Scope of the Approach

The fundamental claim of the context-sensitive asynchronous memory approach is a claim about the problem: a claim that in certain kinds of environments finding good answers to vague questions from large knowledge bases can be done by interleaving reasoning and memory to enable memory to exploit feedback from reasoning to guide retrieval.

The experimental results described in this thesis demonstrate that at least some environments satisfy the assumptions of this claim, enabling a wide variety of problems to be solved using a context-sensitive asynchronous memory and experience-based agent approaches. A wide variety, but not all; systems based on these approaches need certain properties in the environments they are deployed in to provide maximum benefit.

Most important of these environmental properties is the relevance of context to experience: an experience-based agent is most effective when similar problems reoccur over time in the context of cues which enable a context-sensitive asynchronous memory system recommend them when they are useful.

The context-sensitive asynchronous memory approach builds on this relationship between current observables and past experience, and can be most successfully applied to environments which support a rich representation of the relationship between questions and answers, to tasks which generate information during task performance, and to

task/environment combinations in which individual items can be readily discriminated from each other.

This kind of memory is of most use to a complete experience-based agent when the fundamental problems the agent faces are hard, requiring past experience, but when the integration problem is easy, enabling that past information to be easily incorporated to aid current processing.

Summarizing earlier chapters' discussions, the environmental properties which make the approach the most applicable include:

- **Context Assumption:** In a regularity-rich environment, an agent that accumulates experience can improve its ability to recall relevant experiences by exploiting information in its current situation not explicitly specified as part of its information needs.
- **Interleaving Assumption:** In a resource-constrained environment, an agent that incrementally searches for information can reduce the cost impact of retrieval by interleaving memory search with action and reasoning and can improve the comprehensiveness of retrieval by continuing to search as reasoning progresses.
- **Monitoring Assumption:** In an information-poor environment, an agent that combines incrementally searches and experiential storage can improve the comprehensiveness or accuracy of retrieval by exploiting additional context or specifications generated by reasoning, action or environmental events.

- **Integration Assumption:** In a challenging environment with nearly decomposable problems, an agent that combines the relevant portions of past experiences can solve difficult problems more quickly and effectively than attempting to solve them from scratch.

## 11.2. The Model

The context-sensitive asynchronous memory approach exploits these fundamental features of an environment to solve the problem of finding good answers to bad questions. To do so, context-sensitive asynchronous memory and its sister model experience-based agency contain a suite of cooperating functional components, each of which has novel features which enable the approach or enhance its functioning:

- **novel features of context-sensitive asynchronous memory:**

Context-sensitive asynchronous memory is a novel approach to memory architecture, made possible by three components working together: an asynchronous retrieval manager controlling a context-sensitive search process incrementally searching a content addressable store. Each of these components in turn has specific features that enable or enhance their functioning:

- **novel features of asynchronous retrieval management:**

Asynchronous retrieval management is made possible by a novel data structure called a *reified retrieval request* and a corresponding novel process called a *reified retrieval request queue processor*; together, these components



enable the incremental, anytime, asynchronous processing of memory retrieval requests and the spontaneous retrieval of candidate results.

- **novel features of context-sensitive search:**

Context-sensitive asynchronous memory is possible with traditional spreading activation, but it is enhanced by the use of a novel *context-directed spreading activation* process which uses context to alter the flow of activation and effectively reduce the branching factor of memory search.

- **novel features of content addressable memory:**

While a context-sensitive asynchronous memory system could potentially use a variety of different content addressable memories, the particular memory proposed as part of this approach, the *experience store*, enhances the memory retrieval process by providing the knowledge structure sufficient for context directed spreading activation and by automatically maintaining links between items entered into the knowledge base and potentially relevant contextual cues.

- **requirements on reasoning:**

To intelligently process spontaneous retrieval, reasoners must be able to generate feedback, accept spontaneous retrievals, judge their utility and incorporate them into current processing. The context-sensitive asynchronous memory approach proposes a novel model of *integration mechanisms* which enable reasoning to satisfy these constraints.

- **novel features of agent architecture:**

To build a complete agent based on context-sensitive asynchronous memory principles, intelligent agents need an infrastructure and protocol which enable memory and reasoning to communicate. The context-sensitive asynchronous memory approach's companion, the experience-based agent approach, provides a uniform working memory and task architecture which enable agents to achieve these functions:

- **Novel features of working memory:**

To enable memory and reasoning to communicate, experience-based agency incorporates a novel model of working memory which can automatically provide cues to a context-sensitive search process.

- **Novel features of task control:**

To enable memory and reasoning to interoperate, experience-based agency incorporates a novel hierarchical task control architecture which is integrated with the working memory structures which tasks use to communicate and which incorporates synchronization and parallel processing primitives which enable multiple tasks to be naturally interleaved or executed in parallel.

### **11.3. Evaluations and Results**

The fundamental claim about the context-sensitive asynchronous memory approach can be unpacked into a series of subsidiary claims:

- **Claim 1: Interleaving Memory with Reasoning and Action is Useful**

Interleaving memory with reasoning is useful for finding good answers from large knowledge bases with limited information and resources. Interleaving enables memory systems to exploit feedback from reasoning to guide their search to more relevant answers. Furthermore, interleaving enables anytime and asynchronous retrieval, making it possible to trade off comprehensiveness of memory retrieval and resource consumption.

- **Claim 2: Context-sensitive Asynchronous Memory Enables Interleaving**

Context-sensitive asynchronous memory enables interleaving through its asynchronous retrieval management and context-sensitive search process. Asynchronous retrieval management is a model of memory retrieval capable of incremental search of memory which can be interleaved with other reasoning or environmental processes; context-sensitive search in turn enables an asynchronous retrieval manager to exploit feedback generated by these other processes.

- **Claim 3: Rich Representation Makes Context-sensitive Asynchronous Memory Domain-Independent**

A grounded, bidirectional semantic network with reified relations is a knowledge representation with both the features necessary to support context-sensitive asynchronous memory and the representational power to support a wide range of tasks and domains.

- **Claim 4: Communication, Integration and Control are Required to Exploit Context-sensitive Asynchronous Memory**

To profitably interleave reasoning and memory, some control system must exist that orchestrates the execution of the reasoning and memory processes; furthermore, a reasoning task must have the ability to communicate its needs for information to the context-sensitive asynchronous memory and must have the capacity to integrate those results as they arrive.

This dissertation presented a variety of experimental results that validated these claims.

Experiments with Nicole-IRIA demonstrated the first claim, showing that context-sensitive asynchronous memory can provide benefit to real-world problems by finding useful answers to vague questions using task and environmental information. These experiments show that a context-sensitive asynchronous memory can find relevant resources in response to vague web queries using only the browsing context.

Experiments with Nicole-MPA demonstrated the second and fourth claims, showing that the key sources of power of the implemented context-sensitive asynchronous memory approach were consistent with that predicted by its model. These experiments show that context sensitivity, asynchrony, and interleaving all contribute to context-sensitive asynchronous memory's performance and that retrieval evaluation, relevance

determination and integration processing are all necessary to exploit a context-sensitive asynchronous memory.

The use of Nicole's memory in the ISAAC system demonstrated the third claim, showing the generality of the context-sensitive asynchronous memory approach for memory retrieval through its use in a radically different context. This usage, along with the case studies, shows that the context-sensitive asynchronous memory approach applies to tasks as varied as information retrieval, planning, and story understanding.

Finally, experiments with the Nicole system itself supported the second, third and fourth claims, serving as a proof of concept of the experience-based agency approach and a testbed for experimental and theoretical analyses of the feasibility of the approach. These experiments verify the contributions of context, knowledge base structure and cost control policies to a context-sensitive asynchronous memory's performance and demonstrate that a context-sensitive asynchronous memory is a viable memory retrieval approach.

## **11.4. Benefits**

These results show that context-sensitive asynchronous memory enables users performing real tasks to find answers to questions which would otherwise be unanswerable, and that experience-based agency enables designers to build artificial

intelligence systems that exploit context-sensitive asynchronous memories for improved performance.

By using feedback from the environment and task performance, a context-sensitive asynchronous memory can find answers which would not have otherwise been found. When answers are plentiful, a context-sensitive asynchronous memory can bring the most relevant answers to the forefront, ensuring that the best answers are not overlooked. This was demonstrated in Nicole-IRIA through its ability to use context to find useful recommendations out of hundreds or thousands of potential items, and demonstrated in Nicole-MPA when it was able to use cueing information to retrieve cases from larger case bases that would otherwise have been overlooked.

Context-sensitive asynchronous memory enables agents to efficiently balance resources between task processing and memory retrieval. By incrementally searching the knowledge base, a context-sensitive asynchronous memory supports anytime retrieval of the best answer found so far and thus enables agents to satisfice, terminating the search with the first suitable answer found. This satisficing was observed in both Nicole-IRIA and Nicole-MPA. Both systems could quickly find some relevant satisficing answers; both also demonstrated anytime retrieval, in Nicole-MPA through studies showing that it could improve its retrieval over time, and in Nicole-IRIA's through studies in which the context of a user search enabled Nicole-IRIA to incrementally find more relevant results from its knowledge base.

Context-sensitive asynchronous memory retrieval furthermore enables agents easily use additional information to improve retrieval. By explicitly recording and maintaining the state of a search, a context-sensitive asynchronous memory supports incorporating new retrieval specifications, thus enabling agents to refine their questions in the hope of getting better answers. This was demonstrated in Nicole-IRIA, where continue updating of the cues of a search powered the identification of relevant recommendations.

From the end-user perspective, these capabilities enable an intelligent system based on a context-sensitive asynchronous memory to exhibit flexible, accurate, human-like performance: rather than simply returning fixed answers in response to questions, a system based on context-sensitive asynchronous memory can work with a user, unobtrusively watching the user's performance on the task to help it reorganize its presentation of information or explicitly taking hints from the user to help it find better answers — all without discarding the work it has already performed.

## **11.5. Lessons Learned**

Taken as a whole, these experiments, applications and case studies confirm the usefulness of the context-sensitive asynchronous memory approach and the sources of power behind the theory, and demonstrate what is necessary to fully and effectively exploit a context-sensitive asynchronous memory. The lessons learned during these experiments illuminated the strengths and weaknesses of the approach, help outline the

contributions the approach makes to artificial intelligence research, and suggest avenues for future research.

Some of these lessons learned have been reviewed in the chapters discussing the case studies and feasibility studies; here, the most important of these lessons are reviewed and additional overarching lessons are distilled.

### **11.5.1. Lessons Learned in Memory Retrieval**

The case studies and experiments revealed that context-sensitive asynchronous memory works well on a variety of knowledge bases and with a variety of feedback sources. However, the approach has a few requirements to function correctly:

- **Knowledge must be transparently represented:**

Items to be retrieved must not be opaque atoms; instead, their important content must be exposed to the memory system either directly or through annotation tags.

- **Context should contain both observable features and relationships to targets:**

When context is explicitly supplied by the reasoner, its content is important; activating both relevant content in the environment and the relationships that are likely to connect that content to target items will improve retrieval, as will increasing the gating relationships. However, when all of this information is not completely provided, under certain circumstances the context-directed spreading activation algorithm will automatically activate the appropriate relationships and context items, albeit at reduced intensity.



- **Cost control policies should be comprehensive:**

The application of the context-sensitive asynchronous memory approach to information retrieval highlighted the need for a comprehensive cost control policy. In order to make each cycle of the asynchronous retrieval process effectively constant time, a cost control policy must not rely solely on the limits inherent in spreading activation but must also place bounds on the number of active queries, the complexity of each active query, the complexity of matching specifications, and the complexity of connectivity computations on large knowledge bases.

- **Appropriate settings of CDSA parameters can improve performance:**

Experimentation with Nicole-IRIA also revealed the benefits of careful parameterization of the context-directed spreading activation process. On the planning domain, a variety of parameters were tested for planning and were found to be equivalent. These default parameters were “good enough” to produce good recommendations on information retrieval; however, further experimentation revealed that other settings of the parameters, which made context sensitivity more extreme, were even more effective without causing a degradation in planning. Adjusting the size of the candidate buffer also improved retrieval.

Experimentation with Nicole-IRIA also pointed to a useful property of relation leakage: automatic context focusing. Under certain conditions, relation leakage could automatically focus spreading activation search on links related to the problem at hand.

It remains an interesting topic of future research to see how far this automatic context focusing can be extended.

### **11.5.2. Lessons Learned in Reasoning and Task Architecture**

Experiments with both Nicole-MPA and Nicole-IRIA taught lessons about what reasoning task needs to fully exploit a context-sensitive asynchronous memory:

- **Memory retrievals must be integrated deliberately:**

A context-sensitive asynchronous memory may provide memory retrievals asynchronously, but how and when memory retrievals are integrated into the current reasoning state must be done deliberately on the part of the reasoner and not reactively as items are retrieved. In both Nicole-MPA and Nicole-IRIA systems, spontaneous retrieval could contribute to task performance but failure to carefully screen spontaneous retrievals for values relevant to the current reasoning context could produce poor behavior.

These investigations also revealed that while complex tasks could be implemented at a fairly fine grain level using the task language and working memory structures, larger-grained tasks could be just as effective at integrating reasoning and memory while being far easier to program. However, fairly complex task synchronization primitives, including iteration, blocking, and race conditions, were necessary with all but the simplest reasoning tasks.

### 11.5.3. Lessons Learned in Planning

The case study in planning revealed that multiple plans can be combined in an experience based fashion and that context-sensitive asynchronous memory can support planning, even though the implemented system did not completely exploit all of the potential of the context-sensitive asynchronous memory approach.

- **Multi-plan integration must be based on the needs of the planning process:**

The key to exploiting context-sensitive asynchronous memory was the deliberate control of the integration of spontaneous retrievals, along with the deliberate control of features of the planning process which could undermine the use of asynchronous integration, such as combinatorial explosions resulting from ordering and binding problems in the merging of cases.

These experiments taught lessons directly about planning as well:

- **Controlling the complexity of partial plan states enables dynamic merging:**

The dynamic clipping and splicing model of reasoning integration in Nicole-MPA was most applicable to state-space or limited partially-ordered planning domains which control the ordering and binding problems that can arise in the least-commitment planning process. Partial plans in unconstrained least-commitment planning can have many conflicts between unordered steps and potential bindings of variables; when two complex plans are merged, ordering and binding conflicts in the merged plan can require a combinatorial amount of adaptation effort which swamps the expected benefits. A constrained least-commitment planning domain

which reduces binding conflicts, or alternatively a state-space domain which controls both ordering and binding problems, allows larger and more complex merged plans to be more effectively adapted.

- **Metaplanning enables effective dynamic multiple-plan merging:**

Control of the planning process was also key. Attempting to merge cases into the current planning state as “extended operators” can cause an explosion of partial plans in the search space and create challenges for the effective design of search heuristics. In contrast, the case studies showed that use of a metaplanning model, which explicitly manages operator and case merging search spaces separately and which makes choices about which search space to pursue, proved far more effective at combining multiple plans to solve problems more rapidly.

#### **11.5.4. Lessons Learned in Information Retrieval**

The success of the Nicole-IRIA system in the variety of areas to which it has been applied vindicates the basic idea of context-sensitive asynchronous memory as well as demonstrates its generality across domains within a task. Nicole-IRIA showed that a system using context-sensitive asynchronous memory can exploit feedback to give users qualitatively and quantitatively superior results to a variety of other approaches. But it also showed limits to the approach when used as an applications framework.

- **Cognitive flexibility must be traded off against applications efficiency.**

There are tradeoffs between cognitive flexibility and applications efficiency. Nicole implements many components of the context-sensitive asynchronous

memory approach faithfully at the expense of efficiency. For example, obtaining the ordering of items in the experience store normally requires harvesting data items from the candidate buffer and querying them individually for activation. The deployed version of Nicole-IRIA short-circuits this process, using the weighting of knowledge in the activation space directly rather than using the equivalent but slower candidate buffer interface. The results of the two approaches are the same — a list of nodes ranked by their current spreading activation weights — but the speed of processing is much faster in the short-circuited version, enabling Nicole-IRIA to display results more quickly. In another example, the Nicole-MOORE subsystem recomputes its spreading activation connection tables on each cycle to ensure that spreading activation is in sync with knowledge base changes; a more efficient implementation would use static or cached connection tables.

Most importantly, information retrieval demonstrates that the idea of a “large knowledge base” is very much in the eye of the beholder. As the terms are used in this research, “large” knowledge bases contain thousands of items, and “very large” knowledge bases contain tens to hundreds of thousands of items. Some trials of Nicole-IRIA have built knowledge bases containing fifty thousand items. But in the information retrieval domain, a precompiled knowledge base of 10 million items can be considered small, and the World Wide Web itself contains over a billion publicly *indexable* pages, not to mention an inestimable sea of more publicly *accessible* pages. Knowledge bases

of this size shame the term “very large”: they are *vast*. The lesson learned from the information retrieval domain is this:

- **Vast knowledge bases require new techniques for memory retrieval.**

While the results of the information retrieval case study show that the context-sensitive asynchronous memory approach can extend to vast knowledge bases, Nicole-IRIA itself extends memory retrieval with a harvesting process. Basic algorithms for knowledge access, inheritance and spreading activation must now deal with the possibility that individual nodes may contain tens of thousands or millions of connections to other pieces of knowledge. As discussed in Chapter 3 and Chapter 9, even context directed spreading activation has limits dealing with these kinds of fanouts; if spreading activation is used at all alternative approaches must be considered, such as the harvesting approach already used in Nicole-IRIA, hierarchical knowledge bases which spilt activation, strengthened primed spreading activation approaches which “lazily” access the knowledge base, link filters which reduce branching factors by eliminating links rather than weighting them, or automatic knowledge organization techniques (Kolodner 1983) that provide the structure activation needs to propagate.

## **11.6. Contributions**

The strengths of the context-sensitive asynchronous memory approach rest upon its novel features. Context-sensitive asynchronous memory provides the benefits it does

through novel technical contributions to several areas of artificial intelligence research, including memory, knowledge representation, agent systems, task control, planning, and information retrieval, as well as through novel combinations of techniques from all of these areas.

### **11.6.1. Contributions to Memory**

The most important contribution of this research is the context-sensitive asynchronous memory approach itself. Developing context-sensitive asynchronous memory required a variety of technical advances, including not only novel models, methods, algorithms, languages and representations but also technical analyses of these components as well as implementations used for testing these components. These contributions include:

- **model: context-sensitive asynchronous memory**

Context-sensitive asynchronous memory is a novel method for information access which is general, scalable, operates in parallel with reasoning, controls the cost of retrieval, and exploits contextual information to improve performance. A context-sensitive asynchronous memory is built upon an asynchronous retrieval manager, a context sensitive search process, and a content addressable store. These components contain novel methods and algorithms in their own right:

- **method: asynchronous memory retrieval**

Asynchronous memory retrieval is a novel method for memory retrieval that simultaneously enables anytime retrieval and spontaneous reminders while

supporting the revision of retrieval specifications or the input of contextual information. An asynchronous memory retrieval process accepts memory retrieval requests, processes them incrementally, accepts revised specifications when provided, provides answers when demanded, and proactively alerts reasoning when a result is found. This kind of process can be implemented using a reified retrieval request queue manager and request definition language.

- **algorithm: reified retrieval request queue processor**

A reified retrieval request queue processor is an algorithm which can achieve asynchronous memory retrieval by treating information requests as first-class objects and processing them incrementally in concert with a context-sensitive search process. This differs from existing methods of memory retrieval by decoupling memory from reasoning and enabling asynchronous processing.

- **language: request definition language**

The communication between asynchronous memory and a reasoning task depends on a request definition language, a novel method for explicitly communicating memory retrieval requests. This language differs from existing methods of communicating between memory and reasoning by explicitly providing a method to communicate a desired change to an outstanding request as it is being processed.

- **method: context-sensitive memory search**

Context-sensitive memory search is a novel model of memory search, working



hand in hand with an asynchronous memory, that improves the precision of memory search. Context-sensitive search uses cues beyond the questions it has been asked to guide search, focusing the search effort on the portion of the knowledge base most likely to yield useful answers. This approach enables the priming effect of human memory retrieval, as modeled in cognitive architectures such as ACT\*, to be applied to more general intelligent systems. Context sensitive search can be achieved using an algorithm called context-directed spreading activation.

- **algorithm: context-directed spreading activation**

Context-directed spreading activation algorithm is a novel algorithm for memory search that enables spreading activation to scale up to larger, more complex knowledge bases. Context-directed spreading activation uses the existing set of active nodes in a knowledge base to change the weights of connections dynamically, altering the flow of spreading activation. This has the effect of reducing the effective size and branching factor of a knowledge base and enables the same knowledge base to be used in a variety of ways. Through relation leakage, CDSA can in some circumstances focus itself on relevant context without explicit cueing. Context-directed spreading activation has also been shown to improve the quality of retrieved items within the same task.

- **knowledge representation: experience store**

Another important component of a context-sensitive asynchronous memory is a content-addressable store, implemented here in the experience-store knowledge representation. The experience store enables the context-sensitive asynchronous memory approach to be applied to a variety of tasks. By providing a rich knowledge representation capable of representing highly complex knowledge which supports the features required for the context-directed spreading activation algorithm, the experience store provides a task-independent basis for context-sensitive search.

- **theoretical analyses:**

Part of the contribution of the context-sensitive asynchronous memory approach is an analysis of its applicability. This includes both an analysis of the kinds of domains to which the theory applies, an analysis of the kinds of structure of knowledge that the system requires, analyses of the computational properties of CDSA, and an analysis of the properties of representations necessary to enable the context-sensitive asynchronous memory approach.

- **implementation: Nicole-CRYSTAL and Nicole-MOORE**

Finally, this research contributes an example implementation of this approach capable of use in a variety of contexts. This includes a knowledge representation suitable for context-sensitive asynchronous memory (the CRYSTAL knowledge

representation) as well as a full-fledged context-sensitive asynchronous memory system (the Nicole-MOORE system).

### 11.6.2. Contributions to Agent Systems

Building on the context-sensitive asynchronous memory approach, the experience-based agent architecture enables designers to construct agents with context-sensitive asynchronous memories. Developing experience-based agents also required technical advances, including overall agent architecture, working memory and task control (discussed in the next section), and specifications for constructing reasoning tasks to work with context-sensitive asynchronous memories. These contributions include:

- **model: experience-based agency**

Experience-based agency is a novel, general, memory-centered agent architecture that supports a context-sensitive asynchronous memory. Experience-based agents are a novel combination of context-sensitive asynchronous memory, working memory and task control systems that, along with a specification of how to construct reasoning tasks, provides a framework for building complete systems which incorporate both context-sensitive asynchronous memory principles and real performance tasks.

- **analysis: areas of applicability**

This research has contributed theoretical and practical analyses of each of the components of the experience-based agent framework, providing a way to predict the classes of tasks and situations in which the approach will provide benefits,

enabling users and designers to evaluate the utility of the approach for their task and applications.

- **implementation: Nicole**

This research also contributed an example implementation in the Nicole agent architecture.

### **11.6.3. Contributions to Task Control**

One of the key advances necessary to developing experience-based agency was the development of a unified task communications and control system that supported both parallel task processing and complex task synchronization:

- **model: parallel hierarchical task decomposition:**

The parallel hierarchical task definition system developed in this research is a novel model of task control that unifies task definitions, working memory definitions, and parallel task synchronization. This model is designed to facilitate the integration of memory and reasoning systems that are asynchronously connected. By coupling a task decomposition system that supports complex task logic with a parallelism and synchronization language which supports complex task coordination, the parallel hierarchical task network enables the construction of reasoners which can provide feedback to a parallel memory process and accept proactive retrievals from those processes.

- **analysis: areas of applicability**

The features of the task control system make it particularly suitable for interleaving memory and reasoning in an intelligent agent:

- capable of dynamically interleaving multiple reasoning tasks
- synthesizes elements of reactive and deliberative approaches
- is suited for both complex reasoning and reactive processes
- is especially suited for memory/task interleaving

- **implementation: Nicole-Taskstorm**

This research also contributes an example implementation of this approach (Nicole-Taskstorm) that provides a language for writing the control structure of an artificial intelligence system.

#### **11.6.4. Contributions to Planning**

As part of the case studies, the context-sensitive asynchronous memory approach was applied to first planning, then information retrieval. In the case studies, novel techniques had to be developed in each area to apply the approach. In planning, more specifically case-based planning, these contributions include:

- **model: multi-plan adaptation**

The multi-plan adaptor algorithm enables dynamic recombination of multiple planning cases. By providing a way to clip the relevant parts of a case and splice them into an active planning process, the MPA algorithm provides a method for

dynamically combining multiple planning cases to improve performance. a method for dynamically combining multiple planning cases by clipping and splicing relevant subcases which can provide improved performance over single-case case-based planning

- **analysis: areas of application**

Experimental and theoretical applications of the multi-plan adaptation algorithm showed its range of applicability. Fully taking advantage of the algorithm requires using a metacontroller which explicitly manipulates merging and adaptation; it also applies most strongly to limited plan-space or to state-space planning domains.

- **implementation: Nicole-MPA**

This research also contributed an example implementation in the Nicole-MPA system.

#### **11.6.5. Contributions to Information Retrieval**

The case studies also contributed to the field of information retrieval.

- **model: context-sensitive information retrieval and presentation**

Representing information resources in an experience store and mapping user queries and user browsing to information requests and contextual feedback, a context-sensitive asynchronous memory information retrieval system can find

information in memory relevant to the user's goals and rank it according to a user's current interests.

- **analysis: areas of application**

Analysis of experiments with the Nicole-IRIA system demonstrated that the context-sensitive information retrieval approach applies to a wide variety of domains, but has limits when applied to in-memory search of vast knowledge bases that can be overcome through an information harvesting approach.

- **implementation: Nicole-IRIA**

This research also contributed an example implementation in the Nicole-IRIA system that demonstrated the flexibility of the context-sensitive information retrieval approach.

## **11.7. Future work**

There is a great deal of future work which should be done to follow up on the work described in this dissertation. There are important research questions about context-sensitive asynchronous memory and experience-based agency that need to be answered, limitations to the implementations which need to be addressed in order to explore those research questions, and many areas of pure future research that build on the successful results of this study.

### 11.7.1. Open Research Questions

In memory, while experiments and tests showed that the context-sensitive asynchronous memory approach could provide benefits in a variety of circumstances, these tests themselves have limits. Further work needs to be done to evaluate the approach in a wider range of conditions. Furthermore, some of the less central features of the approach have not been fully tested or evaluated. Some of these limitations include:

- Retrieval has been effectively tested only on queries whose semantic distance to target is limited, primarily as a consequence of the structure of the knowledge base and the types of queries afforded by the case-based planning performance task used in the experiments. As the semantic distance of relevant targets increases — and as a consequence the strength of spreading activation decreases — the experience-based agent approach may need an additional “strategic” search component which directs search through the search space, such as the “beacon” search of the KDSA approach (Wolverton 1994).
- Automatic priming — generating the context for context-sensitive memory using automatic propagation of activation from working memory — has been implemented but not tested. Evaluations of the mechanisms of context-sensitive memory indicate that they are sound, but automatically activating these mechanisms made the behavior of the performance task harder to predict and



debug. When a more robust planning or reasoning architecture is in place, the impact of automatic priming should be thoroughly investigated.

- The original plan for the retrieval architecture (Francis 1994) included an adaptive storage component which has been only partially implemented and not tested. The complete design and implementation of the adaptive storage component depends on examination and evaluation of a pattern of retrieval requests from a fully functioning agent. When a more robust planning or reasoning architecture is in place, an investigation of adaptive storage should be conducted.

Furthermore, while context-sensitive asynchronous memory has been tested on knowledge bases with information from multiple tasks, it has yet to be tested on the kind of full multi-task AI system for which it was designed. Context-sensitive asynchronous search was also designed to assist an agent operating in an environment, and it has not yet been tested operating in a real-world environment. The approach should be applied to these kinds of artificial intelligence systems and then should be extensively investigated to demonstrate whether the predictions of the theory hold.

### **11.7.2. Extending the Implementations**

There are also limits on the implementations constructed to date, including the implementations in the case studies as well as in the Nicole architecture itself. Extending

these systems to overcome these limitations will allow a fuller analysis of the context-sensitive asynchronous memory and experience-based agent approaches.

In Nicole-MPA, both the metaplanning system and the plan merging algorithm have limitations:

- the new metaplanning system which supplanted the original state-spaced planner is brittle, causing the performance improvement demonstration to fail in certain conditions. More sophisticated metaplanning techniques are needed, including a wider array of metaplanning operators and the ability to backtrack in the metaplanning space.
- The partially-ordered plan integration mechanisms are limited and brittle, providing performance improvement primarily for small problems or restricted domains. Two solutions are possible: a more sophisticated merging algorithm based on a goal-case table could provide guidance on when to merge plans, eliminating some of the search explosion; alternatively (or concurrently) the merging algorithms could be adapted to a state-space search approach which would limit the binding/ordering search explosion caused during plan-space merging.

Nicole's architectural infrastructure also has limitations:

- A limitation of the task controller is the power and consequent complexity of the task language; it can be difficult to write large programs using fine-grained task decomposition.
- The knowledge representation is “agnostic” about features such as truth maintenance and probabilities — that is, it enables the implementation of these features but does not mandate them. This was deemed necessary to enable the agent architecture to support a wide range of tasks, but a complete agent architecture would need to make commitments about the native representation of this kind of information.
- The overall agent controller has a task decomposition system, but lacks a goal or motivational system that could direct the systems behavior or cause the activation of a task.

### 11.7.3. Future Work

This research has established the basic feasibility of the context-sensitive asynchronous memory approach but has not scratched the surface of future research directions or potential applications based on this idea. Some of these ideas include:

- **research in agent systems:**

A missing element in the control architecture of an experience-based agent is a motivational system: a set of top-level goals or themes which could direct and organize the agent’s activity. This was partially deliberate: experience-based

agency views the memory system as an “essential organ” of mind which is always active; one no more decides to use it than one decides to use one’s heart. However, performance tasks in real agents are performed on the basis of more basic motivations — themes, goals, drives, or emotions. An account of these processes is necessary to make the experience-based agency theory a more complete model of an intelligent agent, and thus to ease the application of context-sensitive asynchronous memory to a wider variety of systems.

- **research topics in memory:**

Context-sensitive asynchronous memory is a model of “automatic” memory search: requests are made and the memory algorithm attempts to satisfy them quickly and without deliberation. Strategic memory search occurs on the part of a reasoning task when it examines retrieved items, weighs them with respect to its goals, and generates new cues or specifications for retrieved items in the hope of getting better results from memory. While some strategic memory processes may be entirely task-specific, it would be interesting to see what common operations exist across tasks in an attempt to build a standard strategic memory module. Systems such as CYRUS (Kolodner 1983) and IDA (Wolverton 1994) provide a start in this direction. Other interesting areas of future research including investigating other forms of context focusing in addition to primed and gated spreading activation, such as activation gated based on the types of concepts. Also, the role of structure in the context-sensitive search process should be

explored more thoroughly, including not only knowledge domains with richer structure but also automatically learned structure that could improve retrieval.

- **research in information retrieval:**

A variety of future research directions exist in applying the context sensitive asynchronous memory approach to information retrieval. The approach could be used to guide an automatic web spidering process that incrementally searched for more relevant pages on the web, not just in memory. This could tie into an interesting proposal by Kenneth Moorman to apply the ISAAC system to reading the content of web pages directed by the context sensitive asynchronous memory search process. The organization and presentation of results is another area of future research; mining retrieved results for categories or other information could both improve the presentation of information to users and could add structure to the knowledge base that could improve retrieval. The Nicole-IRIA toolkit has also been applied to developing computer-based tools to aid problem based learning (Barrows 1998, P. Ram 1999, Francis et al. 2000c, P. Ram et al. 2000); similar educational applications remain an interesting area for future research.

- **research in problem solving:**

The section on limitations hinted at the need for more extensive merging or metaplanning algorithms. One such avenue of research would be a merging system based on a “goal-case table” which maps retrieved cases against goals in the plan, tracks whether those goals are satisfied, and estimates the potential

usefulness of a retrieved case. Using a goal-case table could potentially enable a reasoner to discard useless cases or alternatively to discard bad work on a plan in favor of a superior retrieved case. Another avenue of research would be a more sophisticated metaplanning system with a wider range of operators, perhaps combined with an approach like a goal-case table which would enable the metaplanner to consider not only merging and generative problem solving but also backtracking and other advanced planning operations.

- **applications to creativity research:**

One particular thread of interest involves application of the context-sensitive asynchronous memory approach to creativity. The *Aha! phenomenon* occurs when a creative person receives a flash of insight when working on a problem. One possible model for the Aha! phenomenon is that a reified retrieval request for a solution to a high-priority problem may become dormant if it is not satisfied immediately. If this request is retrieved and satisfied at a later date, its importance could interrupt processing at many levels of the task network. Darwin's notes on his discovery of the theory of evolution provide a well-documented and analyzed example of the Aha! phenomenon. An interesting research project would be to attempt to model Darwin's reasoning process using an experience-based agent and match the predictions of the theory with the trace recorded in the notebooks. Another interesting project would be to combine experience-based agency with a model of invention, such as enterprise-directed reasoning (Simina 1999).

- **applications to cognitive psychology:**

The original inspiration of the context-sensitive asynchronous memory approach was human memory (Francis 1994). The approach was designed to be consistent with a wide range of results on human memory retrieval and can be considered a child of such theories as ACT (Anderson 1983). Many “hooks” exist in the Nicole system to adjust its parameters, enabling the modeling of human cognitive phenomena. An interesting project would be to attempt to model a human cognitive phenomena with Nicole to determine its fitness as an architecture for cognition.

Finally, as already discussed, the context-sensitive asynchronous memory approach should be more thoroughly tested and applied to other domains. Other directions include an experience based agent version of a design system like JULIA, or multi-task or large-domain extensions of route planning systems ROUTER. Another direction might be applications of context-sensitive asynchronous memory to agents embodied in real or simulated environments, which could be used to help test the automatic priming and reactive control components of the theory.

## **11.8. Conclusion**

Context-sensitive asynchronous memory is an approach to information retrieval grounded in the philosophy that the ultimate goal of artificial intelligence is to create systems with human level performance. The instantiation of context-sensitive

asynchronous memory in the experience-based agent architecture, its applications to tasks as varied as planning, information retrieval and story understanding, and the approach with which individual components of the system were constructed were guided by this overarching philosophy.

To both apply and continue this research, I advocate keeping our eyes on the prize — focusing on how researchers in artificial intelligence can continue to extend the performance, capabilities, and flexibility of intelligent systems towards that possessed by human beings. To this end, I advocate:

- build intelligent systems components with cognitive agents in mind, rather than restricting the problem to a subset which cannot be extended
- build intelligent systems which process difficult tasks in an interleaving / incremental / anytime fashion, enabling them to either react quickly to situations or to kick back and exploit the time and resources available to them
- build intelligent systems which deal with large amounts of information in a context-sensitive fashion

Context-sensitive asynchronous memory and experience-based agency are instantiations of this idea applied to managing information access to large, multifunction knowledge bases. As a result of applying this idea, the resulting approach is general, efficient and enables the construction of more efficient, flexible intelligent systems which can opportunistically exploit novel information or additional resources, and therefore



constitutes an advance in the state of the art of the tools available to construct cognitive agents.

# APPENDICES

---

*Domains and Problem Sets*

# APPENDIX A

## PLANNING DOMAINS

---

*Data structures used in the evaluation of Nicole-MPA*

### A.1 The Simple Travel Domain

```
(create-domain travelworld
:documentation "The Travel Domain"
:nicknames (travel)
:literals (human
           kinkos postoffice
           Hometown USA World
           Luggage Passport
          )
:loader load-travelworld-ops
:operators
(
;;-----;;
;; STEP go ;;
;;-----;;
;; NOTES: Define a step for going someplace within ;;
;; a town. ;;
;;-----;;
(defstep
:action '(go ?person ?place)
:precond '((at ?person ?start)
           (location ?place Hometown)
           (location ?start Hometown)
          )
:postcond '((at ?person ?place)
            (not (at ?person ?start))
           )
:equals '(not (?place ?start))
)

;;-----;;
;; STEP get ;;
;;-----;;
;; NOTES: Define a step for getting an object ;;
;;-----;;
(defstep
:action '(get ?person ?object)
:precond '((at ?person ?place)
           (in ?object ?place)
          )
:postcond '((in ?object luggage)
            (not (in ?object ?place))
           )
:equals '()
)

;;-----;;
;; STEP put ;;
;;-----;;
;; NOTES: Define a step for putting an object ;;
```

```

;;-----;;
(defstep
  :action '(put ?person ?object)
  :precond '((at ?person ?place)
             (in ?object luggage)
            )
  :postcond '((in ?object ?place)
              (not (in ?object luggage))
             )
  :equals '(
            )
  )

;;-----;;
;; STEP fly-generic ;;
;;-----;;
;; NOTES: Define a step for an international flight ;;
;;-----;;
(defstep
  :action '(fly-generic ?person ?finish)
  :precond '((at ?person ?start)
             (location ?start ?place1)
             (location ?finish ?place2)
            )
  :postcond '((at ?person ?finish)
              (not (at ?person ?start))
             )
  :equals '(
            (not (?finish ?start))
            (not (?place1 ?place2))
            )
  )
)
)

```

## A.2 The Stinger Missile Domain

```
(create-domain stinger
:documentation "The Stinger Missile Domain"
:nicknames (missile)
:literals (human
           kinkos postoffice
           Hometown USA World
           Luggage Passport
           )
:loader load-stinger-ops
:operators
(
;; *****
;; Personal Movement Operators
;; *****
;; -----
;; STEP go
;; -----
;; NOTES: Define a step for going someplace within
;; a town.
;; -----
(defstep
:action '(go ?person ?start ?finish ?city)

:precond '((at-person ?person ?start)
           (location ?finish ?city)
           (location ?start ?city)
           )

:postcond '((at-person ?person ?finish)
            (not (at-person ?person ?start))
            )

:equals '((not (?finish ?start))
          )
)

;; -----
;; STEP fly-domestic
;; -----
;; NOTES: Define a step for an international flight
;; -----
(defstep
:action '(fly-domestic ?person
                       ?start ?place1
                       ?finish ?place2
                       ?country1)

:precond '((at-person ?person ?start)
           (legal ?person)
           (airport ?start)
           (airport ?finish)
           (location ?start ?place1)
           (location ?finish ?place2)
           (country ?place1 ?country1)
           (country ?place2 ?country1)
           )

:postcond '((at-person ?person ?finish)
            (not (at-person ?person ?start))
            )

:equals '((not (?finish ?start))
          (not (?place1 ?place2))
          )
)

;; -----
;; STEP fly-international
;; -----
;; NOTES: Define a step for an international flight
;; -----
(defstep
```

```

:action  '(fly-international ?person
                        ?start ?place1 ?country1
                        ?finish ?place2 ?country2
                        )
:precond  '((at-person ?person ?start)
            (legal      ?person)
            (holding    ?person passport)
            (airport    ?start)
            (airport    ?finish)
            (location   ?start ?place1)
            (location   ?finish ?place2)
            (country    ?place1 ?country1)
            (country    ?place2 ?country2)
            )
:postcond  '((at-person ?person ?finish)
             (not (at-person ?person ?start))
             )
:equals    '((not (?finish ?start))
             (not (?place1 ?place2))
             (not (?country1 ?country2))
             )
)

;; *****;;
;; Object Movement Operators                               ;;
;; *****;;
;;-----;;
;; STEP get                                                ;;
;;-----;;
;; NOTES: Define a step for getting an object              ;;
;;-----;;
(defstep
:action  '(get ?person ?object ?place)
:precond  '((at-person ?person ?place)
            (at-object ?object ?place)
            )
:postcond  '((holding ?person ?object)
             (not (at-object ?object ?place))
             )
:equals    '(
)
)

;;-----;;
;; STEP put                                                ;;
;;-----;;
;; NOTES: Define a step for putting an object              ;;
;;-----;;
(defstep
:action  '(put ?person ?object ?place)
:precond  '((at-person ?person ?place)
            (holding ?person ?object)
            )
:postcond  '((at-object ?object ?place)
             (not (holding ?person ?object))
             )
:equals    '(
)
)

;; *****;;
;; Luggage Movement Operators                             ;;
;; *****;;
;;-----;;
;; STEP pick-up                                            ;;
;;-----;;
;; NOTES: Define a step for picking up luggage            ;;
;;-----;;
(defstep
:action  '(pick-up ?person ?luggage ?place)
:precond  '((at-person ?person ?place)

```

```

        (at-luggage ?luggage ?place)
      )
      :postcond '((porting ?person ?luggage)
                  (not (at-luggage ?luggage ?place)))
      )
      :equals '()
    )

;;-----;;
;; STEP put                                     ;;
;;-----;;
;; NOTES: Define a step for putting an object  ;;
;;-----;;
(defstep
  :action '(put-down ?person ?luggage ?place)
  :precond '((at-person ?person ?place)
              (porting ?person ?luggage)
              )
  :postcond '((at-luggage ?luggage ?place)
              (not (porting ?person ?object)))
  )
  :equals '()
)

;;-----;;
;; STEP stow                                     ;;
;;-----;;
;; NOTES: Define a step for getting an object  ;;
;;-----;;
(defstep
  :action '(stow ?person ?object ?luggage)
  :precond '((holding ?person ?object)
              (porting ?person ?luggage)
              )
  :postcond '((in-luggage ?object ?luggage)
              (not (holding ?person ?object)))
  )
  :equals '()
)

;;-----;;
;; STEP put                                     ;;
;;-----;;
;; NOTES: Define a step for putting an object  ;;
;;-----;;
(defstep
  :action '(retrieve ?person ?object ?luggage)
  :precond '((porting ?person ?luggage)
              (in-luggage ?object ?luggage)
              )
  :postcond '((holding ?person ?object)
              (not (in-luggage ?object ?luggage)))
  )
  :equals '()
)

;;*****;;
;; Contraband Movement Operators                ;;
;;*****;;
;;-----;;
;; STEP obtain                                     ;;
;;-----;;
;; NOTES: Define a step for obtaining an object ;;
;;-----;;
(defstep
  :action '(obtain ?person ?contraband ?place)
  :precond '((at-person ?person ?place)
              (at-contraband ?contraband ?place)
              )

```

```

        (legal          ?person)
      )
:postcond '((carrying      ?person ?contraband)
            (not (at-contraband ?contraband ?place))
            (not (legal ?person)))
      )
:equals  '(
      )
)

;;-----;;
;; STEP deliver                                     ;;
;;-----;;
;; NOTES: Define a step for deliveriting an object ;;
;;-----;;
(defstep
:action  '(deliver ?person ?contraband ?place)
:precond '((at-person ?person ?place)
            (carrying ?person ?contraband)
            )
:postcond '((at-contraband ?contraband ?place)
            (legal          ?person)
            (not (carrying ?person ?contraband)))
      )
:equals  '(
      )
)

;;-----;;
;; STEP hide                                       ;;
;;-----;;
;; NOTES: Hide contraband and make person legal. ;;
;;-----;;
(defstep
:action  '(hide ?person ?contraband ?luggage)
:precond '((carrying ?person      ?contraband)
            (porting  ?person      ?luggage)
            )
:postcond '((hidden   ?contraband ?luggage)
            (legal     ?person)
            (not (carrying ?person ?contraband)))
      )
:equals  '(
      )
)

;;-----;;
;; STEP unhide                                     ;;
;;-----;;
;; NOTES: Unhide contraband and make the person   ;;
;;         illegal for flight purposes             ;;
;;-----;;
(defstep
:action  '(unhide ?person ?contraband ?luggage)
:precond '((legal      ?person)
            (porting ?person      ?luggage)
            (hidden   ?contraband ?luggage)
            )
:postcond '((carrying   ?person ?contraband)
            (not (legal  ?person))
            (not (hidden ?contraband ?luggage)))
      )
:equals  '(
      )
)

;;*****;;
;; Contraband Action Operators                     ;;
;;*****;;
;;-----;;
;; STEP destroy                                     ;;

```



```

;;-----;;
;; NOTES: Destroy a location using a contraband      ;;
;;          missile.                                ;;
;;-----;;
(defstep
  :action '(destroy ?person ?location ?missile)
  :precond '((at-person ?person ?location)
             (missile ?missile)
             (carrying ?person ?missile)
             )
  :postcond '((destroyed ?location)
              (not (carrying ?person ?missile))
              )
  )

;;*****;;
;; Presentation Action Operators                      ;;
;;*****;;
;;-----;;
;; STEP present                                      ;;
;;-----;;
;; NOTES: Destroy a location using a set of slides  ;;
;;          and a boring presentation.                ;;
;;-----;;
(defstep
  :action '(present ?person ?presentation ?location)
  :precond '((at-person ?person ?location)
             (presentation ?presentation)
             (holding ?person ?presentation)
             )
  :postcond '((presented ?person ?presentation ?location)
              )
  )
)
)

```

## A.3 The Abstract-3 Domain

```

;; *****;;
;; SPA Domain Abstract 3;;
;; *****;;
;; (a3-make-domain :column-pairs '((a null) (b a) (c b));;
;; :shadow-pairs '((z (a b c)));;
;; :n 3;;
;; :m 3);;
;; *****;;
(create-domain abstract-3
:documentation "The Abstract-3 Domain"
:nicknames ()
:literals ()
:loader load-abstract-3-ops
:operators
(
;; *****;;
;; Abstract-3 Chaining Operators;;
;; *****;;
(DEFSTEP
:ACTION '(TRANSFORM-SOURCE-A-SINK-NULL-TO-MATRIX-ENTRY-A)
:PRECOND '((SOURCE-A) (SINK-NULL))
:POSTCOND '((MATRIX-ENTRY-A) (NOT (SOURCE-A)) (NOT (SINK-NULL)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ENTRY-A-TO-MATRIX-ELEMENT-A-1-1)
:PRECOND '((MATRIX-ENTRY-A))
:POSTCOND '((MATRIX-ELEMENT-A-1-1) (NOT (MATRIX-ENTRY-A)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ENTRY-A-TO-MATRIX-ELEMENT-A-1-2)
:PRECOND '((MATRIX-ENTRY-A))
:POSTCOND '((MATRIX-ELEMENT-A-1-2) (NOT (MATRIX-ENTRY-A)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ENTRY-A-TO-MATRIX-ELEMENT-A-1-3)
:PRECOND '((MATRIX-ENTRY-A))
:POSTCOND '((MATRIX-ELEMENT-A-1-3) (NOT (MATRIX-ENTRY-A)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ELEMENT-A-1-1-TO-MATRIX-ELEMENT-A-2-1)
:PRECOND '((MATRIX-ELEMENT-A-1-1))
:POSTCOND '((MATRIX-ELEMENT-A-2-1) (NOT (MATRIX-ELEMENT-A-1-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ELEMENT-A-1-1-TO-MATRIX-ELEMENT-A-2-2)
:PRECOND '((MATRIX-ELEMENT-A-1-1))
:POSTCOND '((MATRIX-ELEMENT-A-2-2) (NOT (MATRIX-ELEMENT-A-1-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ELEMENT-A-1-1-TO-MATRIX-ELEMENT-A-2-3)
:PRECOND '((MATRIX-ELEMENT-A-1-1))
:POSTCOND '((MATRIX-ELEMENT-A-2-3) (NOT (MATRIX-ELEMENT-A-1-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ELEMENT-A-1-2-TO-MATRIX-ELEMENT-A-2-1)
:PRECOND '((MATRIX-ELEMENT-A-1-2))
:POSTCOND '((MATRIX-ELEMENT-A-2-1) (NOT (MATRIX-ELEMENT-A-1-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ELEMENT-A-1-2-TO-MATRIX-ELEMENT-A-2-2)
:PRECOND '((MATRIX-ELEMENT-A-1-2))
:POSTCOND '((MATRIX-ELEMENT-A-2-2) (NOT (MATRIX-ELEMENT-A-1-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION '(TRANSFORM-MATRIX-ELEMENT-A-1-2-TO-MATRIX-ELEMENT-A-2-3)
:PRECOND '((MATRIX-ELEMENT-A-1-2))
:POSTCOND '((MATRIX-ELEMENT-A-2-3) (NOT (MATRIX-ELEMENT-A-1-2)))
:EQUALS 'NIL)

```

```

(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-1-3-TO-MATRIX-ELEMENT-A-2-1)
:PRECOND     '((MATRIX-ELEMENT-A-1-3))
:POSTCOND    '((MATRIX-ELEMENT-A-2-1) (NOT (MATRIX-ELEMENT-A-1-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-1-3-TO-MATRIX-ELEMENT-A-2-2)
:PRECOND     '((MATRIX-ELEMENT-A-1-3))
:POSTCOND    '((MATRIX-ELEMENT-A-2-2) (NOT (MATRIX-ELEMENT-A-1-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-1-3-TO-MATRIX-ELEMENT-A-2-3)
:PRECOND     '((MATRIX-ELEMENT-A-1-3))
:POSTCOND    '((MATRIX-ELEMENT-A-2-3) (NOT (MATRIX-ELEMENT-A-1-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-1-TO-MATRIX-ELEMENT-A-3-1)
:PRECOND     '((MATRIX-ELEMENT-A-2-1))
:POSTCOND    '((MATRIX-ELEMENT-A-3-1) (NOT (MATRIX-ELEMENT-A-2-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-1-TO-MATRIX-ELEMENT-A-3-2)
:PRECOND     '((MATRIX-ELEMENT-A-2-1))
:POSTCOND    '((MATRIX-ELEMENT-A-3-2) (NOT (MATRIX-ELEMENT-A-2-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-1-TO-MATRIX-ELEMENT-A-3-3)
:PRECOND     '((MATRIX-ELEMENT-A-2-1))
:POSTCOND    '((MATRIX-ELEMENT-A-3-3) (NOT (MATRIX-ELEMENT-A-2-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-2-TO-MATRIX-ELEMENT-A-3-1)
:PRECOND     '((MATRIX-ELEMENT-A-2-2))
:POSTCOND    '((MATRIX-ELEMENT-A-3-1) (NOT (MATRIX-ELEMENT-A-2-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-2-TO-MATRIX-ELEMENT-A-3-2)
:PRECOND     '((MATRIX-ELEMENT-A-2-2))
:POSTCOND    '((MATRIX-ELEMENT-A-3-2) (NOT (MATRIX-ELEMENT-A-2-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-2-TO-MATRIX-ELEMENT-A-3-3)
:PRECOND     '((MATRIX-ELEMENT-A-2-2))
:POSTCOND    '((MATRIX-ELEMENT-A-3-3) (NOT (MATRIX-ELEMENT-A-2-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-3-TO-MATRIX-ELEMENT-A-3-1)
:PRECOND     '((MATRIX-ELEMENT-A-2-3))
:POSTCOND    '((MATRIX-ELEMENT-A-3-1) (NOT (MATRIX-ELEMENT-A-2-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-3-TO-MATRIX-ELEMENT-A-3-2)
:PRECOND     '((MATRIX-ELEMENT-A-2-3))
:POSTCOND    '((MATRIX-ELEMENT-A-3-2) (NOT (MATRIX-ELEMENT-A-2-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-2-3-TO-MATRIX-ELEMENT-A-3-3)
:PRECOND     '((MATRIX-ELEMENT-A-2-3))
:POSTCOND    '((MATRIX-ELEMENT-A-3-3) (NOT (MATRIX-ELEMENT-A-2-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-3-1-TO-MATRIX-EXIT-A)
:PRECOND     '((MATRIX-ELEMENT-A-3-1))
:POSTCOND    '((MATRIX-EXIT-A) (NOT (MATRIX-ELEMENT-A-3-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-3-2-TO-MATRIX-EXIT-A)
:PRECOND     '((MATRIX-ELEMENT-A-3-2))
:POSTCOND    '((MATRIX-EXIT-A) (NOT (MATRIX-ELEMENT-A-3-2)))
:EQUALS      'NIL)
(DEFSTEP

```

```

:ACTION      '(TRANSFORM-MATRIX-ELEMENT-A-3-3-TO-MATRIX-EXIT-A)
:PRECOND     '((MATRIX-ELEMENT-A-3-3))
:POSTCOND    '((MATRIX-EXIT-A) (NOT (MATRIX-ELEMENT-A-3-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-EXIT-A-TO-SINK-A)
:PRECOND     '((MATRIX-EXIT-A))
:POSTCOND    '((SINK-A) (NOT (MATRIX-EXIT-A)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-SINK-A-TO-SIDE-EFFECT-A)
:PRECOND     '((SINK-A))
:POSTCOND    '((SIDE-EFFECT-A))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-SOURCE-B-SINK-A-TO-MATRIX-ENTRY-B)
:PRECOND     '((SOURCE-B) (SINK-A))
:POSTCOND    '((MATRIX-ENTRY-B) (NOT (SOURCE-B)) (NOT (SINK-A)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ENTRY-B-TO-MATRIX-ELEMENT-B-1-1)
:PRECOND     '((MATRIX-ENTRY-B))
:POSTCOND    '((MATRIX-ELEMENT-B-1-1) (NOT (MATRIX-ENTRY-B)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ENTRY-B-TO-MATRIX-ELEMENT-B-1-2)
:PRECOND     '((MATRIX-ENTRY-B))
:POSTCOND    '((MATRIX-ELEMENT-B-1-2) (NOT (MATRIX-ENTRY-B)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ENTRY-B-TO-MATRIX-ELEMENT-B-1-3)
:PRECOND     '((MATRIX-ENTRY-B))
:POSTCOND    '((MATRIX-ELEMENT-B-1-3) (NOT (MATRIX-ENTRY-B)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-1-TO-MATRIX-ELEMENT-B-2-1)
:PRECOND     '((MATRIX-ELEMENT-B-1-1))
:POSTCOND    '((MATRIX-ELEMENT-B-2-1) (NOT (MATRIX-ELEMENT-B-1-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-1-TO-MATRIX-ELEMENT-B-2-2)
:PRECOND     '((MATRIX-ELEMENT-B-1-1))
:POSTCOND    '((MATRIX-ELEMENT-B-2-2) (NOT (MATRIX-ELEMENT-B-1-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-1-TO-MATRIX-ELEMENT-B-2-3)
:PRECOND     '((MATRIX-ELEMENT-B-1-1))
:POSTCOND    '((MATRIX-ELEMENT-B-2-3) (NOT (MATRIX-ELEMENT-B-1-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-2-TO-MATRIX-ELEMENT-B-2-1)
:PRECOND     '((MATRIX-ELEMENT-B-1-2))
:POSTCOND    '((MATRIX-ELEMENT-B-2-1) (NOT (MATRIX-ELEMENT-B-1-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-2-TO-MATRIX-ELEMENT-B-2-2)
:PRECOND     '((MATRIX-ELEMENT-B-1-2))
:POSTCOND    '((MATRIX-ELEMENT-B-2-2) (NOT (MATRIX-ELEMENT-B-1-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-2-TO-MATRIX-ELEMENT-B-2-3)
:PRECOND     '((MATRIX-ELEMENT-B-1-2))
:POSTCOND    '((MATRIX-ELEMENT-B-2-3) (NOT (MATRIX-ELEMENT-B-1-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-3-TO-MATRIX-ELEMENT-B-2-1)
:PRECOND     '((MATRIX-ELEMENT-B-1-3))
:POSTCOND    '((MATRIX-ELEMENT-B-2-1) (NOT (MATRIX-ELEMENT-B-1-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-B-1-3-TO-MATRIX-ELEMENT-B-2-2)

```

```

:PRECOND ' ((MATRIX-ELEMENT-B-1-3))
:POSTCOND ' ((MATRIX-ELEMENT-B-2-2) (NOT (MATRIX-ELEMENT-B-1-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-1-3-TO-MATRIX-ELEMENT-B-2-3)
:PRECOND ' ((MATRIX-ELEMENT-B-1-3))
:POSTCOND ' ((MATRIX-ELEMENT-B-2-3) (NOT (MATRIX-ELEMENT-B-1-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-1-TO-MATRIX-ELEMENT-B-3-1)
:PRECOND ' ((MATRIX-ELEMENT-B-2-1))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-1) (NOT (MATRIX-ELEMENT-B-2-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-1-TO-MATRIX-ELEMENT-B-3-2)
:PRECOND ' ((MATRIX-ELEMENT-B-2-1))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-2) (NOT (MATRIX-ELEMENT-B-2-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-1-TO-MATRIX-ELEMENT-B-3-3)
:PRECOND ' ((MATRIX-ELEMENT-B-2-1))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-3) (NOT (MATRIX-ELEMENT-B-2-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-2-TO-MATRIX-ELEMENT-B-3-1)
:PRECOND ' ((MATRIX-ELEMENT-B-2-2))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-1) (NOT (MATRIX-ELEMENT-B-2-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-2-TO-MATRIX-ELEMENT-B-3-2)
:PRECOND ' ((MATRIX-ELEMENT-B-2-2))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-2) (NOT (MATRIX-ELEMENT-B-2-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-2-TO-MATRIX-ELEMENT-B-3-3)
:PRECOND ' ((MATRIX-ELEMENT-B-2-2))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-3) (NOT (MATRIX-ELEMENT-B-2-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-3-TO-MATRIX-ELEMENT-B-3-1)
:PRECOND ' ((MATRIX-ELEMENT-B-2-3))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-1) (NOT (MATRIX-ELEMENT-B-2-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-3-TO-MATRIX-ELEMENT-B-3-2)
:PRECOND ' ((MATRIX-ELEMENT-B-2-3))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-2) (NOT (MATRIX-ELEMENT-B-2-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-2-3-TO-MATRIX-ELEMENT-B-3-3)
:PRECOND ' ((MATRIX-ELEMENT-B-2-3))
:POSTCOND ' ((MATRIX-ELEMENT-B-3-3) (NOT (MATRIX-ELEMENT-B-2-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-3-1-TO-MATRIX-EXIT-B)
:PRECOND ' ((MATRIX-ELEMENT-B-3-1))
:POSTCOND ' ((MATRIX-EXIT-B) (NOT (MATRIX-ELEMENT-B-3-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-3-2-TO-MATRIX-EXIT-B)
:PRECOND ' ((MATRIX-ELEMENT-B-3-2))
:POSTCOND ' ((MATRIX-EXIT-B) (NOT (MATRIX-ELEMENT-B-3-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-B-3-3-TO-MATRIX-EXIT-B)
:PRECOND ' ((MATRIX-ELEMENT-B-3-3))
:POSTCOND ' ((MATRIX-EXIT-B) (NOT (MATRIX-ELEMENT-B-3-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-EXIT-B-TO-SINK-B)
:PRECOND ' ((MATRIX-EXIT-B))

```

```

:POSTCOND ' ((SINK-B) (NOT (MATRIX-EXIT-B)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-SINK-B-TO-SIDE-EFFECT-B)
:PRECOND ' ((SINK-B))
:POSTCOND ' ((SIDE-EFFECT-B))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-SOURCE-C-SINK-B-TO-MATRIX-ENTRY-C)
:PRECOND ' ((SOURCE-C) (SINK-B))
:POSTCOND ' ((MATRIX-ENTRY-C) (NOT (SOURCE-C)) (NOT (SINK-B)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ENTRY-C-TO-MATRIX-ELEMENT-C-1-1)
:PRECOND ' ((MATRIX-ENTRY-C))
:POSTCOND ' ((MATRIX-ELEMENT-C-1-1) (NOT (MATRIX-ENTRY-C)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ENTRY-C-TO-MATRIX-ELEMENT-C-1-2)
:PRECOND ' ((MATRIX-ENTRY-C))
:POSTCOND ' ((MATRIX-ELEMENT-C-1-2) (NOT (MATRIX-ENTRY-C)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ENTRY-C-TO-MATRIX-ELEMENT-C-1-3)
:PRECOND ' ((MATRIX-ENTRY-C))
:POSTCOND ' ((MATRIX-ELEMENT-C-1-3) (NOT (MATRIX-ENTRY-C)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-1-TO-MATRIX-ELEMENT-C-2-1)
:PRECOND ' ((MATRIX-ELEMENT-C-1-1))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-1) (NOT (MATRIX-ELEMENT-C-1-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-1-TO-MATRIX-ELEMENT-C-2-2)
:PRECOND ' ((MATRIX-ELEMENT-C-1-1))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-2) (NOT (MATRIX-ELEMENT-C-1-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-1-TO-MATRIX-ELEMENT-C-2-3)
:PRECOND ' ((MATRIX-ELEMENT-C-1-1))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-3) (NOT (MATRIX-ELEMENT-C-1-1)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-2-TO-MATRIX-ELEMENT-C-2-1)
:PRECOND ' ((MATRIX-ELEMENT-C-1-2))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-1) (NOT (MATRIX-ELEMENT-C-1-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-2-TO-MATRIX-ELEMENT-C-2-2)
:PRECOND ' ((MATRIX-ELEMENT-C-1-2))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-2) (NOT (MATRIX-ELEMENT-C-1-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-2-TO-MATRIX-ELEMENT-C-2-3)
:PRECOND ' ((MATRIX-ELEMENT-C-1-2))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-3) (NOT (MATRIX-ELEMENT-C-1-2)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-3-TO-MATRIX-ELEMENT-C-2-1)
:PRECOND ' ((MATRIX-ELEMENT-C-1-3))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-1) (NOT (MATRIX-ELEMENT-C-1-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-3-TO-MATRIX-ELEMENT-C-2-2)
:PRECOND ' ((MATRIX-ELEMENT-C-1-3))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-2) (NOT (MATRIX-ELEMENT-C-1-3)))
:EQUALS 'NIL)
(DEFSTEP
:ACTION ' (TRANSFORM-MATRIX-ELEMENT-C-1-3-TO-MATRIX-ELEMENT-C-2-3)
:PRECOND ' ((MATRIX-ELEMENT-C-1-3))
:POSTCOND ' ((MATRIX-ELEMENT-C-2-3) (NOT (MATRIX-ELEMENT-C-1-3)))

```

```

:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-1-TO-MATRIX-ELEMENT-C-3-1)
:PRECOND     '((MATRIX-ELEMENT-C-2-1))
:POSTCOND    '((MATRIX-ELEMENT-C-3-1) (NOT (MATRIX-ELEMENT-C-2-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-1-TO-MATRIX-ELEMENT-C-3-2)
:PRECOND     '((MATRIX-ELEMENT-C-2-1))
:POSTCOND    '((MATRIX-ELEMENT-C-3-2) (NOT (MATRIX-ELEMENT-C-2-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-1-TO-MATRIX-ELEMENT-C-3-3)
:PRECOND     '((MATRIX-ELEMENT-C-2-1))
:POSTCOND    '((MATRIX-ELEMENT-C-3-3) (NOT (MATRIX-ELEMENT-C-2-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-2-TO-MATRIX-ELEMENT-C-3-1)
:PRECOND     '((MATRIX-ELEMENT-C-2-2))
:POSTCOND    '((MATRIX-ELEMENT-C-3-1) (NOT (MATRIX-ELEMENT-C-2-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-2-TO-MATRIX-ELEMENT-C-3-2)
:PRECOND     '((MATRIX-ELEMENT-C-2-2))
:POSTCOND    '((MATRIX-ELEMENT-C-3-2) (NOT (MATRIX-ELEMENT-C-2-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-2-TO-MATRIX-ELEMENT-C-3-3)
:PRECOND     '((MATRIX-ELEMENT-C-2-2))
:POSTCOND    '((MATRIX-ELEMENT-C-3-3) (NOT (MATRIX-ELEMENT-C-2-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-3-TO-MATRIX-ELEMENT-C-3-1)
:PRECOND     '((MATRIX-ELEMENT-C-2-3))
:POSTCOND    '((MATRIX-ELEMENT-C-3-1) (NOT (MATRIX-ELEMENT-C-2-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-3-TO-MATRIX-ELEMENT-C-3-2)
:PRECOND     '((MATRIX-ELEMENT-C-2-3))
:POSTCOND    '((MATRIX-ELEMENT-C-3-2) (NOT (MATRIX-ELEMENT-C-2-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-2-3-TO-MATRIX-ELEMENT-C-3-3)
:PRECOND     '((MATRIX-ELEMENT-C-2-3))
:POSTCOND    '((MATRIX-ELEMENT-C-3-3) (NOT (MATRIX-ELEMENT-C-2-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-3-1-TO-MATRIX-EXIT-C)
:PRECOND     '((MATRIX-ELEMENT-C-3-1))
:POSTCOND    '((MATRIX-EXIT-C) (NOT (MATRIX-ELEMENT-C-3-1)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-3-2-TO-MATRIX-EXIT-C)
:PRECOND     '((MATRIX-ELEMENT-C-3-2))
:POSTCOND    '((MATRIX-EXIT-C) (NOT (MATRIX-ELEMENT-C-3-2)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-ELEMENT-C-3-3-TO-MATRIX-EXIT-C)
:PRECOND     '((MATRIX-ELEMENT-C-3-3))
:POSTCOND    '((MATRIX-EXIT-C) (NOT (MATRIX-ELEMENT-C-3-3)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-MATRIX-EXIT-C-TO-SINK-C)
:PRECOND     '((MATRIX-EXIT-C))
:POSTCOND    '((SINK-C) (NOT (MATRIX-EXIT-C)))
:EQUALS      'NIL)
(DEFSTEP
:ACTION      '(TRANSFORM-SINK-C-TO-SIDE-EFFECT-C)
:PRECOND     '((SINK-C))
:POSTCOND    '((SIDE-EFFECT-C))
:EQUALS      'NIL)

```

```

(DEFSTEP
  :ACTION
    '(TRANSFORM-SIDE-EFFECT-A-SIDE-EFFECT-B-SIDE-EFFECT-C-TO-TARGET-Z)
  :PRECOND  '((SIDE-EFFECT-A) (SIDE-EFFECT-B) (SIDE-EFFECT-C))
  :POSTCOND '((TARGET-Z)
                (NOT (SIDE-EFFECT-A))
                (NOT (SIDE-EFFECT-B))
                (NOT (SIDE-EFFECT-C))
                )
  :EQUALS   'NIL)
)

```



# APPENDIX B

## PROBLEM SETS

---

*Data structures used in the evaluation of Nicole-MOORE*

### B.1 The X2 Problem Set

```
(create-problem
px2-r-apd
:documentation "STINGER: at-person-domestic problem"
:domain        #!stinger
:initial        ((at-person researcher hartsfield)
                 (holding  researcher passport)
                 (porting   researcher luggage)
                 (legal     researcher)
                 (location  dulles      washington)
                 (airport   dulles)
                 (location  hartsfield atlanta)
                 (airport   hartsfield)
                 (location  athensintl athens)
                 (airport   athensintl)
                 (country   washington usa)
                 (country   atlanta     usa)
                 (country   athens      greece)
                 )
:goal           ((at-person researcher dulles))
)

(create-problem
px2-r-apf
:documentation "STINGER: at-person-international problem"
:domain        #!stinger
:initial        ((at-person researcher hartsfield)
                 (holding  researcher passport)
                 (porting   researcher luggage)
                 (legal     researcher)
                 (location  dulles      washington)
                 (airport   dulles)
                 (location  hartsfield atlanta)
                 (airport   hartsfield)
                 (location  athensintl athens)
                 (airport   athensintl)
                 (country   washington usa)
                 (country   atlanta     usa)
                 (country   athens      greece)
                 )
:goal           ((at-person researcher athensintl))
)

(create-problem
px2-r-apl
:documentation "STINGER: at-person-international problem"
:domain        #!stinger
:initial        ((at-person researcher hartsfield)
                 (holding  researcher passport)
                 (porting   researcher luggage)
                 (legal     researcher)
                 (location  dulles      washington)
                 )
:goal           ((at-person researcher athensintl))
)
```

```

        (airport    dulles)
        (location   hartsfield atlanta)
        (airport    hartsfield)
        (location   athensintl athens)
        (airport    athensintl)
        (location   big-chicken atlanta)
        (country    washington usa)
        (country    atlanta     usa)
        (country    athens      greece)
    )
:goal      ((at-person researcher big-chicken))
)

(create-problem
px2-r-edc
:documentation "The Domestic Conference Travel Problem"
:domain        #!stinger
:initial        ((at-person  Researcher  GeorgiaTech)
                 (porting    Researcher  Luggage)
                 (at-object   Presentation Kinkos)
                 (legal       Researcher)
                 (location    GeorgiaTech Atlanta)
                 (location    Kinkos      Atlanta)
                 (location    Hartsfield  Atlanta)
                 (airport     Hartsfield)
                 (country     Atlanta     USA)
                 (location    Dulles      Washington)
                 (airport     Dulles)
                 (country     Dulles      USA)
                 )
:goal          ((at-person  Researcher  Dulles)
                 (in-luggage Presentation Luggage)
                 )
)

(create-problem
px2-r-efc
:documentation "The Foreign Conference Travel Problem"
:domain        #!stinger
:initial        ((at-person  Researcher  GeorgiaTech)
                 (at-luggage Luggage    KingsTavern)
                 (at-object   Presentation Kinkos)
                 (at-object   Passport   Postoffice)
                 (legal       Researcher)
                 (location    KingsTavern Atlanta)
                 (location    GeorgiaTech Atlanta)
                 (location    Postoffice  Atlanta)
                 (location    Kinkos      Atlanta)
                 (location    Hartsfield  Atlanta)
                 (airport     Hartsfield)
                 (country     Atlanta     USA)
                 (location    AthensIntl  Athens)
                 (airport     AthensIntl)
                 (location    Acropolis   Athens)
                 (country     Athens      Greece)
                 )
:goal          ((at-person  Researcher  Acropolis)
                 (in-luggage Presentation Luggage)
                 )
)

(create-problem
px2-r-epl
:documentation "The Researcher's Presentation Travel Problem"
:domain        #!stinger
:initial        ((at-person  Researcher  KingsTavern)
                 (porting    Researcher  Briefcase)
                 (in-luggage Slides     Briefcase)
                 )
)

```

```

        (presentation Slides)
        (location KingsTavern Atlanta)
        (location GeorgiaTech Atlanta)
        (country Atlanta USA)
    )

:goal      ((presented Researcher Slides GeorgiaTech)
)

(create-problem
px2-r-hdc
:documentation "The Domestic Conference Travel Problem"
:domain      #!stinger
:initial      ((at-person Researcher GeorgiaTech)
               (porting Researcher Luggage)
               (at-object Slides Kinkos)
               (presentation Slides)
               (legal Researcher)
               (location GeorgiaTech Atlanta)
               (location Kinkos Atlanta)
               (location Hartsfield Atlanta)
               (airport Hartsfield)
               (country Atlanta USA)
               (location AAAI Washington)
               (location Dulles Washington)
               (airport Dulles)
               (country Dulles USA)
               )

:goal      ((presented Researcher Slides AAAI)
)

(create-problem
px2-r-hfc
:documentation "The HARD Foreign Conference Travel Problem"
:domain      #!stinger
:initial      ((at-person Researcher GeorgiaTech)
               (at-luggage Luggage KingsTavern)
               (at-object ECML-Slides Kinkos)
               (presentation ECML-Slides)
               (at-object Passport Postoffice)
               (legal Researcher)
               (location KingsTavern Atlanta)
               (location GeorgiaTech Atlanta)
               (location Postoffice Atlanta)
               (location Kinkos Atlanta)
               (location Hartsfield Atlanta)
               (airport Hartsfield)
               (country Atlanta USA)
               (location AthensIntl Athens)
               (airport AthensIntl)
               (location Acropolis Athens)
               (country Athens Greece)
               )

:goal      ((presented Researcher ECML-Slides Acropolis)
)

(create-problem
px2-r-hpl
:documentation "The Researcher's Hard Presentation Travel Problem"
:domain      #!stinger
:initial      ((at-person Researcher KingsTavern)
               (at-object LaserPointer GeorgiaTech)
               (at-object Emory-Slides Kinkos)
               (presentation Emory-Slides)
               (location KingsTavern Atlanta)
               (location Kinkos Atlanta)

```

```

        (location GeorgiaTech Atlanta)
        (location Emory Atlanta)
        (country Atlanta USA)
    )

:goal ((holding Researcher LaserPointer)
      (presented Researcher Slides Emory)
      )
)

(create-problem
px2-r-pack
:documentation "The Packing Travel Problem"
:domain #!stinger
:initial ((at-person Researcher KingsTavern)
         (at-luggage Luggage KingsTavern)
         (at-object Presentation KingsTavern)
         (location KingsTavern Atlanta)
         (country Atlanta USA)
         )
:goal ((in-luggage Presentation Luggage)
       )
)

(create-problem
px2-r-shop
:documentation "The Grocery Shopping Problem"
:domain #!stinger
:initial ((at-person Researcher University)
         (at-object Milk Kroger)
         (at-object Bread Kroger)
         (location University Atlanta)
         (location Kroger Atlanta)
         (location Home Atlanta)
         (country Atlanta USA)
         )
:goal ((at-person Researcher Home)
       (at-object Milk Home)
       (at-object Bread Home)
       )
)

(create-problem
px2-t-apd
:documentation "STINGER: at-person-international problem"
:domain #!stinger
:initial ((at-person terrorist hartsfield)
         (holding terrorist passport)
         (porting terrorist luggage)
         (carrying terrorist contraband)
         (location dulles washington)
         (airport dulles)
         (location hartsfield atlanta)
         (airport hartsfield)
         (location athensintl athens)
         (airport athensintl)
         (country washington usa)
         (country atlanta usa)
         (country athens greece)
         )
:goal ((at-person terrorist dulles))
)

(create-problem
px2-t-apf
:documentation "STINGER: terrorist-at-person-international problem"
:domain #!stinger
:initial ((at-person terrorist hartsfield)
         (holding terrorist passport)
         (porting terrorist luggage)

```

```

        (carrying terrorist contraband)
        (location dulles      washington)
        (airport  dulles)
        (location hartsfield atlanta)
        (airport  hartsfield)
        (location athensintl athens)
        (airport  athensintl)
        (country  washington usa)
        (country  atlanta     usa)
        (country  athens      greece)
    )
:goal      ((at-person terrorist athensintl))
)

(create-problem
px2-t-apl
:documentation "STINGER: terrorist-at-person-international problem"
:domain      #!stinger
:initial      ((at-person terrorist hartsfield)
               (holding  terrorist passport)
               (porting  terrorist luggage)
               (carrying terrorist contraband)
               (location dulles      washington)
               (airport  dulles)
               (location hartsfield atlanta)
               (airport  hartsfield)
               (location athensintl athens)
               (airport  athensintl)
               (location big-chicken atlanta)
               (country  washington usa)
               (country  atlanta     usa)
               (country  athens      greece)
               )
:goal      ((at-person terrorist big-chicken))
)

```

## B.2 The Information Retrieval Problem Set

```
;;-----;;  
;; VARIABLE *queries*                               ;;  
;;-----;;  
;; NOTES: Typical queries by actual users of Nicole-IRIA.  ;;  
;;-----;;  
(defvar *query-list*  
  '("centaur"  
    "java software development"  
    "sushi"  
    "harrison ford"  
    "barbecue"  
    "indian restaurants"  
    "temptations"  
    "doonesbury"  
    "microsoft windows"  
    "linux"  
    "lisp"  
    "yo-yo"  
    "artificial intelligence games"  
    "corba programming"  
    "COM object model"  
    "taido"  
    "dot-com"  
    "online bookbag store"  
    "design pattern"  
    "intelligent search"  
    "relevance filtering"  
    "information retrieval"  
    "klingson language institute"  
    "shakespeare"  
    "space travel"  
    "vegetarian foods"  
    "home gardens"  
  )  
)
```

## B.3 The MetaSpy Problem Set

```
;;-----;;
;; VARIABLE *metaspy-query-list*                               ;;
;;-----;;
;; NOTES: Actual queries to the MetaSearch search engine as    ;;
;;         reported by the "clean version" of the Metaspy Service. ;;
;;         Data collected: 12:32am 7/23/2000                     ;;
;;-----;;
(defvar *metaspy-query-list*
  '(
    "ebooks compiler"
    "candie evans"
    "Edith Ogden Harrison"
    "e-mail Dictionary"
    "ireland real property sale"
    ;; "tiffany taylor mpeg"                                ;; Removed due to harvester problems
    "Red Book"
    "jackyl"
    "toppless golg"
    "pokemon"
    "rites of passage"
    ;; "xtal mp3"                                           ;; Removed due to harvester problems
    "vacation rentals sqaw valley ca"
    "deep dance 66"
    "snes roms"
    "pics of lil kim"
    "replacement rifle barrels for enfield"
    "snes roms"
    "New Playstation Games Review"
    "hotels london england"
    "Pitairn Island"
    "free greeting cards"
    "Bettie Page"
    "Plotters In Canada"
    "corset training"
    "transmitters 2.4 mgz"
    "vatican"
    "flying fish express"
    "computer images"
    ;; "DUKE3D WALKTHROUGHS"                                ;; Removed due to harvester problems
    "ovarian cysts"
    "McAfee Virus updates"
    "nightmare in badham county"
    "Free Plumpers"
    "camilla wierød pettersen"
    "front mission 3"
    "watercolor"
    "bukkake"
    "Screen Saver Freeware"
    "cluster school"
    "trains in Europe"
    "crossword puzzle helper"
    "travel"
    "best air mile credit"
    "disney"
    "wealth"
    "nellie lyrics"
    "summer traffic"
    "heman"
    "eastbay"
    "glenmount ontario"
    "rugby league"
    "claim jumpers menu"
    "pronovias"
    "Internet Speed Test"
    "Coronado, CA"
    "national rugby league"
    "Kimberly Williams Actress"
    "question search engines"
```

```

"wizard works"
"ochelltree"
"hotmail"
"seattle mariner"
"the periodic table"
"eating disorders"
"l carnitine"
"canyon river blues sno apperal"
"Pro Wrestling Chat Rooms"
"sony car cd players"
"fun bitch test"
"rollercoaster tycoon trianers"
"barbara s. waincott"
"computer sites"
"Forbidden Planet London"
"Enrique Inglieis"
"cameltoe"
"quickcam webserver"
"fantasy art"
"Jeep Performance Parts"
"saalem, massachusetts"
"www.bae.ncsu.edu/people/faculty/walker/hotlist/graphics.html"
"cartoon network"
"real estate"
"smashing pumpkins mtv live"
"Tvtap"
"ewok's celebration"
"loincloth Africa"
";"linux irc clones"
"internet"
"address by phone number"
"Bamcinematek"
"African Plains"
"ichy iktestus"
"Home Plans"
"Mary-Kate and Ashley Olsen"
"bmw e36"
"health"
"this week in nascar"
"nicholas gentry"
"blasen"
"X-MEN"
"bobblehead.com"
"Download Windows Millennium"
"chat room"
"search for songs using lyrics"
)
)

```



# BIBLIOGRAPHY

---

## *References and related readings*

- Altavista. (2000). *www.altavista.com*
- Amarel, S. (1967). On Representations of Problems on Reasoning about Actions. Reprinted in Nilsson & Webber, (eds.), *Readings in Artificial Intelligence*, Tioga Press. 1967.
- Anderson, J. R. (1983a). *The Architecture of Cognition*. Cambridge, Massachusetts: Harvard University Press.
- Anderson, J. R. (1983b). A spreading activation theory of memory, *Journal of Verbal Learning and Verbal Behavior*, 22. pp.261-295. Reprinted in A. Collins & E.E. Smith, (eds.), *Readings in cognitive science: a perspective from psychology and artificial intelligence*. (1988). San Mateo: Morgan Kaufman.
- Anderson, J. R. (1990). *The adaptive character of thought*. Hillsdale, New Jersey: Lawrence Earlbaum Associates.
- Anderson, J. R. (1992). "Automaticity and the ACT\* Theory." *The American Journal of Psychology*, 105(2) p 165.
- Anderson, J.R. (1993). *Rules of the mind*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Anderson, J. R., & Reder, L.M. (1979). An elaborative processing explanation for depth of processing. In Cermak, L.S., & Craik, F.I.M. (eds.) *Levels of processing in human memory*. Hillsdale, New Jersey: Lawrence Earlbaum Associates.
- Anderson, J. R., (ed.) (1981). *Cognitive skills and their acquisition*. Hillsdale, New Jersey: Lawrence Earlbaum Associates.
- Arbib, M. (1981). "Perceptual structures and distributed motor control." In Brooks (ed.), *Handbook of Physiology - The Nervous System II*. American Physiological Society. pp. 1449-1465. 1981.
- Arbib, M. (Ed.) (1995). *The Handbook of Brain Theory and Neural Networks*. MIT Press.
- Ardö, A., & Lundberg, S. (1998). A regional distributed WWW search and indexing service – the DESIRE way. In Proceedings of the Seventh International World Wide Web Conference, Brisbane, Australia April 14-18 1998. Also see: <http://nwi.dtv.dk/www7/>; also see <http://www.lub.lu.se/desire/>
- Arkin, R.C. (1989). Motor Schema Based Mobile Robot Navigation. *International Journal of Robotics Research*, 8(4), August 1989.
- Ashley, K.D. (1990). *Modeling legal argument: Reasoning with cases and hypotheticals*. Cambridge, Massachusetts: Bradford/MIT Press.
- Ashley, K.D. (1990). Reasoning with cases and hypotheticals in Hypo. *International Journal of Man-Machine Studies* 34: pp. 753-796.
- Atkinson, R.C. & Shiffrin, R.M. (1968). Human memory: a proposed system and its control processes. In K.W. Spence & J.T. Spence, (eds.), *The psychology of learning and motivation: advances in research and theory*, vol 2. New York: Academic Press, pp. 89-195.
- Baddeley A.D. (1981). "The concept of working memory: A view of its current state and probable future development." *Cognition*, 10, 17-23.

- Baddeley, A.D. & Hitch, G. (1974). Working memory. In G. H. Bower (ed.), *The psychology of learning and motivation*, vol. 8., pp. 47-90.
- Baddeley, A.D. (1976). *The psychology of memory*. New York: Basic Books.
- Baeza-Yates, R. & Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley-Longman.
- Bareiss, E.R., Porter, B.W. and Weir, C.C. (1988). Protos: an exemplar-based learning apprentice. *International journal of man-machine studies* 29:549-561.
- Barletta, R. & Mark, W. (1988). Breaking Cases into Pieces. In Kolodner, J.L. (Ed.), *Proceedings of the DARPA Case-Based Reasoning Workshop*. Morgan Kaufmann.
- Barnhart, E.R. (Publisher). (1991). The Physician's Desk Reference 45<sup>th</sup> edition 1991. Oradelli, NJ: Medical Economics Data.
- Barret, A. & Weld, D. (1994). Partial order planning: evaluating possible efficiency gains. *Artificial Intelligence*, 67 (1), 71-112.
- Barrows, H.S. (1988). *The Tutorial Process: Revised Edition*. Springfield, Illinois: Southern Illinois University School of Medicine.
- Bhatta, S. & Goel, A. (1993). Learning Generic Mechanisms from Experiences for Analogical Reasoning. In *Proc. of the Fifteenth Annual Conf. of the Cog. Sci. Soc.*, 237--242.
- Bhatta, S. & Goel, A. (1994). From Design Experiences to Generic Mechanisms: Model-Based Learning in Analogical Design. In *Proc. of AID'94 Workshop on Machine Learning in Design*, Aug. 1994, Lausanne, Switzerland.
- Bobrow, D.G. (1968) Natural language input for a computer problem solving system. In Minsky, M. (ed.) (1968) *Semantic Information Processing*, pp. 133-215. Cambridge, MA: MIT Press.
- Bolt, R.A. (1985). Conversing with computers. *Technology Review* 88(2), pp. 35-43. Reprinted in Baecker, R.M & Buxton, W.A.S (1987) *Readings in human-computer interaction: A multidisciplinary approach*. Morgan Kaufmann.
- Bowman, C. M., Danzig, P.B., Hardy, D. R., Manber, U., & Schwartz, M.F. (1994). The Harvest information discovery and access system. In *Proceedings of the Second International World Wide Web Conference*, pp. 763-771, Chicago, Illinois, October 1994.
- Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J. (2000) Graph structure in the web. <http://www.almaden.ibm.com/cs/k53/www9.final/>
- Brachman, R. J. (1985). An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9, (1985) 171-216
- Bratman, M. (1987). *Intentions, Plans and Practical Reason*. Harvard.
- Brooks, R. (1986). "A Robust Layered Control System for a Mobile Robot." *IEEE Journal of Robotics and Automation*, 2(1), pp. 14-23, 1986.
- Bury, S. (1999). Web tools energize designers. *Electronic Publishing*, May 1999. pp.45-50.
- Byrne, M., Guzdial, M., Ram, P., Catrambone, R., Ram, A., Stasko, J., Shippey, G., & Albrecht, F. (1996). The Role of Student Tasks in Accessing Cognitive Media Types. In *Proceedings of the Second International Conference on the Learning Sciences*, Evanston, IL.
- Campbell, I. & van Rijsbergen, C.J. (1996). The ostensive model of developing information needs. In *Proceedings of CoLIS2*, pp. 251-268. Copenhagen, Denmark, October 1996.
- Chandler, T. N., & Kolodner, J.L. (1993). The Science Education Advisor: A Case-Based Advising System for Lesson Planning. In *Proceedings of the International Conference on AI and Education*, Scotland.

- Chandrasekaran, B. (1989). Generic tasks as building blocks of knowledge-based systems: the diagnosis and routine design examples. *Knowledge Engineering Review*, 3(3): 183-219, 1988.
- Charniak, E. (1983). Passing Markers: A Theory of Contextual Influence in Language Comprehension. *Cognitive Science* 7, pp 171 -190.
- Chase, William G. and Simon, Herbert A. (1973). "The Mind's Eye in Chess." In Collins, Allan and Smith, Edward. E., eds. 1988. *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*. Palo Alto, California: Morgan Kaufmann Publishers, Inc.
- Chen, I., Bastani, F.B., and Tsao, T. (1995). On the Reliability of AI Planning Software in Real-Time Applications . IEEE Transactions on Knowledge and Data Engineering Vol. 7, No. 1, February 1995
- Cheong, F. (1996). *Internet agents: spiders, wanderers, brokers, and bots*. New Riders.
- Clark, P. & Porter, B. (1996). KM - The Knowledge Machine: User's Manual. AI Lab, University of Texas at Austin. <http://www.cs.utexas.edu/users/mfkb/manuals/userman.ps>
- Clark, P. & Porter, B. (1997). Building Concept Representations from Reusable Components. In AAAI'97, pages 369-376, CA:AAAI Press, 1997.
- CL-HTTP. (2000). <http://www.ai.mit.edu/projects/iip/doc/cl-http/home-page.html>
- Cohen, P. R., Greenberg, M. L., Hart, D.M., and Howe, A. E. (1989). Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *AI Magazine*. 10(3): 32-48.
- Cohen, P.R. & Kjeldsen, R. (1987). Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management*, 23, pp.255-268.
- Cohen, W.W. (1990). "Learning approximate control rules of high utility." In *Machine Learning: Proc. 7<sup>th</sup> Int'l Conf.*, 1990.
- Collins, A. M. & Loftus, E.F. (1975). A spreading activation theory of semantic processing. *Psychological Review*, 82:407-428, 1975. Reprinted in A. Collins & E.E. Smith, (eds.), *Readings in cognitive science: a perspective from psychology and artificial intelligence*. (1988). San Mateo: Morgan Kaufman.
- Cook, S.; Dwork, C.; Reischuk, R. (1986). "Upper and lower time bounds for parallel random access machines without simultaneous writes." *SIAM Jrnl on Computing*, 15(1):87-97, 1986.
- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press, McGraw-Hill. 1990.
- Cox, M. (1993). *Introspective Multistrategy Learning*. Cognitive Science Technical Report #2, Atlanta: Georgia Institute of Technology, College of Computing.
- Craig, I.D. (1991) The Role of Formal Specification in Rule-Based Real-Time AI. *Tech Report No. CS-RR-194*. Department of Computer Science, University of Warwick, Coventry, UK, Sep 1991.
- Craik, F.I.M., & Lockhart, R.S. (1972). "Levels of processing: A framework for memory research." *Journal of Verbal Learning and Verbal Behavior*, 11, 671-684.
- Crestani, F & Lee, P.L. (1999). WebSCSA: Web Search by Constrained Spreading Activation. In *Proceedings of IEEE ADL 99 - Advances in Digital Libraries Conference*, pages 163-170, Baltimore, USA, May 1999.
- Cutting, D.R., Karger, D.R., Pedersen, J.O., Tukey, J.W. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the SIGIR '92*, pp. 318-329. ACM Press.
- CYCORG (2000). *Features of the CycL language*. <http://www.cyc.com/cycl.html>
- De Bra, P.M.E. & Post, R.D.J. (1994). Searching for arbitrary information in the WWW: the Fish-search for Mosaic. Proc. 2<sup>nd</sup>. Int. World Wide Web Conf., 1994.

- DeJong, G.F. (1979) *Skimming stories in real time: an experiment in integrated understanding*. PhD Dissertation, Yale University, 1979.
- Direct Hit. (2000). [www.directhit.com](http://www.directhit.com)
- Domeshek, E. (1992). Do the right thing: A component theory for indexing stories as social advice. *Technical Report #26*, May 1992. Institute for the Learning Sciences, Northwestern University.
- Donald, M. (1991). *Origins of the Modern Mind: Three Stages in Culture and Cognition*. Cambridge, MA: Harvard University Press.
- Doorenbos, Bob. (1993). "Matching 100,000 Learned Rules." In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, July 11-15, 1993, Washington D.C.
- Draper, D., Hanks, S., Weld, D. (1994). Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on AI Planning Systems*, p170-175, June 13-15 1994, Chicago, Illinois.
- Dublin Core Metadata Initiative. (2000). <http://purl.org/DC/>
- Eich, J.M. (1985). Levels of processing, encoding specificity, elaboration and CHARM. *Psychological Review*, 92, 1-38.
- Eich, J.M. (1995). Distortions in human memory. In Arbib, M.A. (Ed.), *The Handbook of Brain Theory and Neural Networks*, pp.321-322. Cambridge, Massachusetts: Bradford/MIT Press.
- Ellis, G. & Levinson, R. (eds.) (1992). *Proceedings of the First International Workshop on PIERCE: A Conceptual Graphs Workbench*. Las Cruces: New Mexico, July 10, 1992.
- Ericsson, K. A., & Kintsch, W. (1995). Long-term working memory. *Psychological Review*, 102, 211-245.
- Erman, L.D., Hayes-Roth, F., Lesser, V.R., Reddy, D.R. (1980). The Hearsay-II Speech Understanding System: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2), June 1980.
- Etzioni, O. (1992). An Asymptotic Analysis of Speedup Learning. In *Machine Learning: Proceedings of the 9<sup>th</sup> International Workshop*, 1992.
- Etzioni, O. (1997). Moving up the information food chain: deploying softbots on the World Wide Web. *AI Magazine*, Summer 1997, pp. 11-18.
- Falkenhainer, B., Forbus, K., and Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41, pp. 1-63.
- Fikes, Hart and Nilsson. (1971). STRIPS. *Artificial Intelligence*, 2, pp. 189-208.
- Firby, J. (1989). Adaptive Execution in Complex Dynamic Worlds Ph.D. Thesis, Yale University Technical Report, YALEU/CSD/RR #672, January 1989.
- Fodor, J. (1983). *The Modularity of Mind*. Cambridge, MA: The MIT Press.
- Forgy, C.L. (1981). *OPS5 User's Manual*. Technical Report (July). Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania. A great town.
- Francis, A.G. & Ram, A. (1993a). The Utility Problem in Case-Based Reasoning. Abstracted in *AAAI-CBR-93, the Proceedings of the 1993 Case-Based Reasoning Workshop*.
- Francis, A.G. & Ram, A. (1993b). Computational Models of the Utility Problem and their Application to a Utility Analysis of Case-Based Reasoning. In *KCSL-93: Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 48-55, University of Massachusetts at Amherst, June 30, 1993. Amherst has good bookstores.

- Francis, A.G. & Ram, A. (1994). A Comparative Utility Analysis of Case-Based and Control-Rule Learning Systems. In *Proceedings of the Workshop on Case-Based Reasoning*. AAAI, 1994. Seattle: Washington. Another great town, with both great food and great bookstores.
- Francis, A.G. & Ram, A. (1995). A Comparative Utility Analysis of Case-Based Reasoning and Control-Rule Learning Systems. In Lavrac, N. & Wrobel, S. (eds.) *Machine Learning: ECML-95. Proceedings, 8<sup>th</sup> European Conference on Machine Learning, Heraklion, Crete, Greece, April 1995*. Heraklion had great restaurants but they're somewhat hard to find. Lovely scenery.
- Francis, A.G. & Ram, A. (1997). Can your architecture do this: A proposal for impasse-driven asynchronous memory retrieval request generation. In *Proceedings of the AAAI-97 Workshop on ROBOTS, SOFTBOTS, IMMOBOTS: Theories of Action, Planning and Control*.
- Francis, A.G. (1995a). "Sibling Rivalry." *The Leading Edge: Magazine of Science Fiction and Fantasy*, 30, February 1995, p79-102.
- Francis, A.G., Ram, A., and Devaney, M. (2000a). IRIA: The Information Research Intelligent Assistant. In Arabina, H.R., (Ed.) *Proceedings of the International Conference on Artificial Intelligence IC-AI'2000*, Las Vegas, Nevada, June 26-29, 2000. CSREA Press.
- Francis, A.G., Ram, A., and Devaney, M. (2000b). *The IRIA Project: Information Research Intelligent Assistant*. Air Force Rome Labs SBIR Technical Report (report number TBA).
- Francis, A.G., Ram, A., Devaney, M., and Ram, P. (2000c). *The CAPABLE Project: Computer Aided Problem Based Learning Engine*. Department of Education Technical Report (report number TBA).
- Freeman, P. (1975). *Software systems principles: a survey*. Chicago: Science Research Associates.
- Genesereth, M.R. & Fikes, R. E. (1992). *Knowledge Interchange Format Version 3.0 Reference Manual*. Computer Science Department, Stanford University, Stanford, California.
- Gentner, D. & Forbus, K. (1991). MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Goel, A. (1989). Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving, PhD Thesis, Ohio State University.
- Goel, A., & Murdock, W. (1996). Meta-cases: Explaining case-based reasoning. In Ian Smith & Boi Faltings, (eds.), *Advances in Case-Based Reasoning: Lecture Notes in Computer Science 1168*. Springer.
- Goel, A., Ali, K.S., Donnellan, M.W., Garza, A.G., and Callantine, T.J. (1994). Multistrategy Adaptive Path Planning. *IEEE Expert (9)* 6, December 1994, pp. 57-65.
- Goel, A., Mahesh, K., Peterson, J. & Eiselt, K. (1996). Unification of language understanding, device comprehension, and knowledge acquisition. In. *Proc. 1996 Cognitive Science Conference*, San Diego, July 1996.
- Golden, K., Etzioni, O., & Weld, D. (1994). Omnipotence Without Omniscience: Sensor Management in Planning. In *Proceedings of AAAI-94*. AAAI Press/MIT Press.
- Grass, G. (1996). Reasoning about Computational Resource Allocation: An introduction to anytime algorithms. *ACM Crossroads*. <http://www.acm.org/crossroads/xrds3-1/racra.html>
- Gratch, J.M. and Dejong, G.F. (1991a) Trouble with gestalts: The composability problem in control learning. Technical Report, University of Illinois at Urbana-Champaign, April 1991. A good town for movies and local theater.
- Gratch, J.M. and Dejong, G.F. (1991b) Utility generalization and composability problems in explanation-based learning. *Technical Report*, University of Illinois at Urbana-Champaign, Illinois, August 1991.

- Gratch, J.M., (ed.) (2000) *Proceedings, Soar Workshop 2000*. <http://www.isi.edu/soar/soar-workshop/workshop00.html>
- Green, H. (1999). The Information Gold Mine. *Business Week e.biz*, July 26 1999, pp. EB17-30
- Greene, R.L. (1992). *Human Memory: Paradigms and Paradoxes*. Lawrence Erlbaum Associates.
- Guise, D. (1988) Intelligent tutoring systems for foreign language acquisition. In *Proceedings of the Asia-Pacific Conference on Computer Education (APCCE88)*, pp. 33-48. Chinese Computer Federation, Shanghai, China 1988.
- Guise, D. (1989) *KR: Constraint-based knowledge representation*. Technical Report CMU-CS-89-142, Carnegie Mellon University Computer Science Department, April 1989.
- Guise, D. (1992) *KR: Constraint-based knowledge representation*. Manual. November 1992, Carnegie Mellon University.
- Haddawy, P. (1996). Focusing Attention in Anytime Decision-Theoretic Planning. *SIGART Bulletin* 7(2): 34-40
- Hammond, K. (1989). Opportunistic Memory. In *Proceedings of the 1989 Meeting of the International Joint Committee on Artificial Intelligence*.
- Hanks, S., & Weld, D. (1992). Systematic Adaptation for Case-Based Planning. In *Proceedings of the First International Conference on AI Planning Systems*, 96-105. San Mateo, California: Morgan Kaufmann.
- Hanks, S., & Weld, D. (1995). A Domain-Independent Algorithm for Plan Adaptation. *Journal of Artificial Intelligence Research* 2, pp. 319-360.
- Hastings, P. (1997) *The Soar Coloring Book: A Tutorial*. Artificial Intelligence Lab, The University of Michigan
- Hayes, P.J. (1979). The logic of frames. Frame conceptions and text understanding. In Webber, B.L. & Nilsson, N.J., *Readings in Artificial Intelligence*, pp. 451-458. San Mateo, California: Morgan Kaufmann.
- Hayes-Roth, B. (1995). Agents on stage: Advancing the State of the Art of AI. *Knowledge Systems Laboratory Report No. KSL 95-50*. May 1995, Knowledge Systems Laboratory, Stanford University.
- Hayes-Roth, B., & Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science* (2), pp. 275-310.
- Hendler, J.A. (1989) Marker passing over micro-features: toward a hybrid symbolic/connectionist model. *Cognitive Science* 13(1).
- Heylighen F. (1999): "Collective Intelligence and its Implementation on the Web: algorithms to develop a collective mental map", *Computational and Mathematical Organization Theory* 5(3), p. 253-280.
- Heylighen F. & Bollen J. (1996): "The World-Wide Web as a Super-Brain: from metaphor to model" in: R. Trappl (ed.) (1996): *Cybernetics and Systems '96* (Austrian Society for Cybernetic Studies), p. 917.
- Hinrichs, T.R. (1992). *Problem Solving in Open Worlds: A Case Study in Design*. Lawrence Erlbaum.
- Holder, L.B.; Porter, B.W.; Mooney, R.J. (1990). The general utility problem in machine learning. In *Machine Learning: Proceedings of the Seventh International Conference*, 1990.
- Holyoak, K. & Hummel, J. (in press). The Proper Treatment of Symbols in a Connectionist Architecture. In E. Deitrich & A. Markman (Eds.), *Cognitive dynamics: Conceptual change in humans and machines*. Cambridge, MA: MIT Press.

- Holyoak, Keith J. (1991). Symbolic connectionism: Toward third-generation theories of expertise. Erikson, I K, & Smith, J. (Eds.): *Towards a General Theory of Expertise: Prospects and Limits*, Cambridge University Press.
- Horvitz, E.J., Suermondt, H.J., and Cooper, G.F. (1989). Bounded Conditioning: Flexible inference for decision under scarce resources. In *Proceedings of the 1989 Workshop on Uncertainty in Artificial Intelligence*, pp. 182-193, Windsor, Ontario, 1989.
- Howe, A.E. & Dreilinger, D. (1997). SavvySearch: A metasearch engine which learns which search engines to query. *AI Magazine*, Summer 1997, pp. 19-25.
- IBN (1998). 1999 Electronic Recruiting Index: The Industry Matures. *IBN: inerbiznet.com* 1998.
- Inktomi. (2000). *www.inktomi.com*
- Internet World (1998). Getting there, or not: Why search is so ineffective. *Internet World Online Edition*, February 23, 1998.
- Irhig, L. & Kambhampati, S. (1994a). Derivation Replay for Partial-Order Planning. In *Proceedings of AAAI-94*. AAAI Press/MIT Press.
- Irhig, L. & Kambhampati, S. (1994b). Evaluating the effectiveness of derivation replay in partial-order vs. state-space planning. In *Proceedings of the Workshop on Case-Based Reasoning*. AAAI, 1994. Seattle: Washington.
- Irhig, L. & Kambhampati, S. (1994c). On the relative utility of plan-space vs. state-space planning in a case-based framework. Technical Report 94-006, Department of Computer Science, Engineering, 1994, Arizona State University.
- Irhig, L. & Kambhampati, S. (1995). Integrating Replay with EBL to Improve Planning Performance. Submitted to ICJAI-95.
- Jacobs, N. & Shea, R. (1996). The role of Java in InfoSleuth: agent-based exploitation of heterogeneous information resources, *IntraNet96 Java Developers Conference*, April 1996.
- Joachims, T., Feitag, D., Mitchell, T. (1997). WebWatcher: a tour guide for the World Wide Web. *Proceedings of IJCAI97*, August 1997.
- Jones, J.L. & Flynn, A.M. (1993). *Mobile Robots: Inspiration to Implementation*. Wellesley, Massachusetts: A K Peters.
- Jones, R. M. & Langley, P. (1995). Retrieval and learning in analogical problem solving. In J. D. Moore & J. F. Lehmann (Eds.), *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (pp. 466-471). Hillsdale, NJ: Lawrence Erlbaum.
- Jones, R. M. (1989). *A model of retrieval in problem solving*. Doctoral dissertation. Department of Information and Computer Science, University of California, Irvine.
- Jones, R. M. (1993) Problem solving via analogical retrieval and analogical search control. In Chipman, S. & Meyrowitz, A. (eds.), *Machine learning: Induction, analogy and discovery*. Boston: Kluwer Academic.
- Jonides, J. (1993). Class discussion in PSY 7011A.
- Kambhampati, S. & Chen, J. (1993). Relative Utility of EBG based Plan Reuse in Partial Ordering vs. Total Ordering Planning. In *Proceedings of AAAI-93*. AAAI Press/MIT Press.
- Kambhampati, S. & Hendler, J. (1992). A Validation Structure Based Theory of Plan Modification and Reuse. *Artificial Intelligence*, 55, 193-258.

- Katukam, S. & Kambhampati, S. (1994). Learning EBL-based search control rules for partial order planning. In *Proceedings, AAAI-94*, 1994.
- Kettler, B.P., Hendler, J.A., Andersen, W.A., and Evett, M.P. (1993). Massively parallel support for case-based planning. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence Applications*. Washington, DC: IEEE CS Press.
- Kintsch, W. (1993). The role of working memory in comprehension. A talk at the Georgia Tech Cognitive Science Colloquium. Friday, May 21, 1993.
- Kitano, H.; Shimazu, H.; Shibata, A. (1993). Case-Method: A Methodology for Building Large-Scale Case-Based Systems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, July 11-15, 1993, Washington D.C.
- Klimesch, W. J. (1994). *The structure of long-term memory: A connectivity model of semantic processing*. LEA.
- Knoblock, C. A., Minton, S., Ambite, J.L., Ashish, N., Modi, P.J., Muslea, I., Philpot, A.G., Tejada, S. (1997) Modeling web sources for information integration. Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI 1998.
- Knuth, D.E. (1973). *The Art of Computer Programming Volume 3: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts.
- Kohonen, T. (1984). *Self-organization and associative memory*. Berlin: Springer-Verlag.
- Kolodner, J.L., & Penberthy, L. (1990). A case-based approach to creativity in problem-solving. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, 978--985. Hillsdale, NJ: Lawrence Erlbaum
- Kolodner, J.L. & Simpson, R.L. (1988). The MEDIATOR: A case study of a case-based reasoner. Georgia Institute of Technology, School of Information and Computer Science, Technical Report no. GIT-ICS-88/11. Atlanta, Georgia.
- Kolodner, J.L. & Simpson, R.L. (1989). The MEDIATOR: Analysis of an early case-based problem solver. *Cognitive Science* 13(4): 507-549.
- Kolodner, J.L. & Wills, L. (1993a). Case-based creative design. Proceedings of the AAAI Spring Symposium on AI and Creativity, Palo Alto, CA, March.
- Kolodner, J.L. & Wills, L. (1993b). Paying attention to the right thing: Issues of focus in case-based creative design. Proceedings of the Ninth Annual Conference of the Cognitive Science Society, July.
- Kolodner, J.L. (1984). *Retrieval and organization strategies in conceptual memory: a computer model*. Northvale, NJ: Lawrence Earlbaum Associates.
- Kolodner, J.L. (1985). Memory for experience. In Bower, G.H. (Ed.) *The psychology of learning and motivation, vol 19*. New York: Academic Press.
- Kolodner, J.L. (1988). Retrieving events from a case memory: a parallel implementation. In *Proceedings: Workshop on case-based reasoning (DARPA)* Clearwater, Florida. San Mateo, California: Morgan Kaufmann.
- Kolodner, J.L. (1993). *Case-based Reasoning*. Morgan Kaufmann.
- Korth, H. F. & Silberschatz, A. (1986). *Database system concepts*. New York: McGraw-Hill.
- Koton, P. A. (1989). *A Method for Improving the Efficiency of Model-Based Reasoning Systems*. Laboratory for Computer Science, MIT, Cambridge, MA. Hemisphere Publishing, 1989.



- Krushke, J.K. (1992). ALCOVE: An exemplar-based connectionist model of category learning. *Psychological Review*, 99(1). pp. 22-44.
- Landauer, T.K. (1975). Memory without organization: Properties of a model with random storage and undirected retrieval. *Cognitive Psychology*, 7, 495-531.
- Lange, T.E., & Wharton, C.M. (1994). Remind: Retrieval from Episodic Memory by INferencing and Disambiguation. In Barnden, J. & Holyoak, K. (eds). *Advances in connectionist and neural computation theory: Analogy, Metaphor and Reminding*. Norwood, NJ: Ablex.
- Langley, P. & Allen, J.A. (1991). The acquisition of human planning expertise. In Birnbaum, L.A. & Collins, G.C. (eds.), *Machine Learning: Proceedings of the Eighth International Workshop*. Los Altos, California: Morgan Kaufmann.
- Law, K., Forbus, K.D., & Gentner, D. (1994). Simulating similarity-based retrieval: A comparison of ARCS and MAC/FAC. In Ram, A. & Eiselt, K. (eds.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. Hillsdale, New Jersey: Lawrence Earlbaum Associates.
- Lawrence, S. & Giles, C.L. (1998). Searching the world wide web. *Science*, 280 pp.98-100.
- Lawrence, S. & Giles, C.L. (1999). Accessibility of Information on the Web, *Nature*, 400, pp.107-109.
- Lenat, D.B. and Guha, R.V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley Publishing Company, Inc., 1990.
- Levinson, R. & Ellis, G. (eds.) (1993). *Proceedings of the Second International Workshop on PIERCE: A Conceptual Graphs Workbench*. Quebec: Canada, August 7, 1993.
- Levinson, R. (1994). Human frontal lobes and AI planning systems. In *Proceedings of the Second International Conference on AI Planning Systems*, p170-175, June 13-15 1994, Chicago, Illinois.
- Liu, L., Pu, C., Barga, R. and Zhou, T. (1996). Differential Evaluation of Continual Queries. Extended abstract in *IEEE Proceedings of the 16th International Conference on Distributed Computing Systems*, 27-30 May, 1996, Hong Kong. The long version is available as TR95-17, Department of Computing Science, University of Alberta.
- Long, D. (Ed.) (2000). The AIPS-98 Planning Competition. *AI Magazine*, (21)2, Summer 2000, pp. 13-34.
- Maes, P. (1990). Situated agents can have goals. In P. Maes, Ed., *Designing Autonomous Agents: Theory and Practice from Biology and Engineering and Back*. MIT.
- Maes, P. (1993). "Behavior-Based Artificial Intelligence." In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*. June 18-21, 1993.
- Maida, A. S. & Shapiro, S. C. (1982). Intensional concepts in propositional semantic networks. *Cognitive Science*, 6(4):291-330, 1982. Reprinted in Brachman, R. J. & Levesque, H. J. (eds.) *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, CA, 1985, 170-189.
- Mallery, J.C. (1994). A Common LISP hypermedia server. *Proceedings of the First International Conference on the World-Wide Web*, Geneva: CERN, May 25, 1994.
- Markovitch, S. and Scott, P.D. (1989) Utilization Filtering: a method for reducing the inherent harmfulness of deductively learned knowledge. In *IJCAI-89: Proc. of the 11<sup>th</sup> International Joint Conference on Artificial Intelligence*, 1989.
- Markovitch, S. and Scott, P.D. (1993) Information Filtering: Selection Methods in Learning Systems. *Machine Learning*, 10: 113-151.
- Marr, D. (1982). *Vision*. W.H.Freeman and Co., New York.

- Marr, J. (1993). Automaticity: A Behavioral Perspective. A talk given at the Cognitive Science Colloquium Series, 1993.
- Maudlin, M.L. (1991). *Conceptual information retrieval: a case study in adaptive partial parsing*. Boston, Massachusetts: Kluwer Academic Publishers.
- McAllester, D., & Rosenblitt, D. (1991). Systematic Nonlinear Planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 634-639. AAAI Press/MIT Press.
- McClelland, J.L. & Rumelhart, D.E. (1986). A distributed model of human learning and memory. In Rumelhart, D.E., & McClelland, J.L. (eds.) *Parallel distributed processing: Explorations in the microstructure of cognition, vol 2*. Cambridge, MA: MIT Press.
- McDermott, D. (2000). The 1998 AI Planning Systems Competition. *AI Magazine*, (21)2, Summer 2000, pp. 35-56.
- McKerrow, P.J. (1991). *Introduction to Robotics*. Sydney, Australia: Addison-Wesley.
- McNamara, T.P. (1992). Priming and constraints it places on theories of memory and retrieval. *Psychological Review*, 99, 650-662.
- McNamara, T. P., & Diwadkar, V. A. (1996). The context of memory retrieval. *Journal of Memory and Language*, 35, 877-892
- McWilliams, B. (1998). Search engine to sell top positions on results lists. *PC World Online*, February 23, 1998. <http://www2.pcworld.com/news/daily/0298/9802233173204.html>
- Medin, D.L., & Ross, B.H. (1992). *Cognitive Psychology*. Fort Worth, Texas: Harcourt Brace Jovanovich.
- MetaSpy. (2000). [www.metaspj.com](http://www.metaspj.com)
- Meyer, D.E., & Schvaneveldt, R.W. (1971). Facilitation in recognizing pairs of words: Evidence of a dependence between retrieval operations. *Journal of Experimental Psychology*, 90, 227-234.
- Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. 1993. Introduction to WordNet: An On-Line Lexical Database.
- Miller, G.A. (1956). The magic number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63, 81-97.
- Minsk, B., Balakrishnan, H., and Ram, A. (1995). Structuring on-the-job troubleshooting performance to aid learning. *Proceedings of the Fourth World Conference on Engineering Education*. St. Paul, Minnesota; October 1995.
- Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, San Mateo, California: Morgan Kaufmann.
- Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3), March 1990.
- Minton, S., Bresina, J., Drummond, M. (1994). Total order and Partial-Order Planning: A Comparative Analysis. *Journal of Artificial Intelligence Research* 2, p319-360
- Moore, A.W., & Atkeson, C.G. (1995) Memory based neural networks for robot learning. *Neurocomputing*, 9(3) pp. 243-269.
- Moorman, K. & Francis, A. (1994). *Case-based reasoning and the data chunking problem in soar: Failures in the Captain's Advisory Tool, Soar Version*. Unpublished Project Report, Spring 1994.
- Moorman, K. & Ram, A. (1994). Integrating Creativity and Reading: A Functional Approach. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, GA, August 1994.

- Moorman, K. (1997) A functional theory of creative reading: process, knowledge and evaluation. Ph.D. Dissertation, Georgia Institute of Technology.
- Mouaddib, A. & Zilberstein, S. (Chairs). (1998). *ECAI-98 Workshop: Monitoring and control of real-time intelligent systems*. Held at the 13th biennial European Conference on Artificial Intelligence August 24, 1998, Brighton Centre, Brighton, UK.
- Mueller, E. T. & Dyer, M.G. (1985). Towards a computational theory of human daydreaming. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pp. 120-129. Irvine, CA.
- Musliner, D.J. (2000). *Year 2000 AAAI Spring Symposium on Real-Time Autonomous Systems*, Stanford, CA. March 20-22, 2000.
- Myers, B.A. et al. (1990) Garnet: Comprehensive support for graphical, highly-interactive user interfaces. *IEEE Computer* 23(11) Nov 1990. pp. 71-85.
- National Library of Medicine (2000) *UMLS Knowledge Sources Documentation*, 11<sup>th</sup> Edition. <http://www.nlm.nih.gov/research/umls/UMLSDOC.HTML>
- Newell, A. & Rosenbloom, P.S. (1981) Mechanisms of skill acquisition and the law of practice. In J.R. Anderson, (ed.), *Cognitive skills and their acquisition*, pp. 1-56. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Newell, A. & Simon, H.A. (1975). *Computer Science as Empirical Inquiry: Symbols and Search*. Reprinted in Haugeland, J., (ed.). *Mind Design: Philosophy, Psychology, Artificial Intelligence*, chapter 1, pp 35-66. MIT Press, 1981.
- Newell, A. (1962). Some problems of basic organization in problem-solving programs. In Yovitz, M.C., Jacobi, G.T., & Goldstein, G.D. (Eds.), *Conference on Self-Organizing Systems*. Washington, D.C.: Spartan Books.
- Newell, A. (1990). *Unified theories of cognition*. Harvard.
- Nickerson, R.S. (1977). On conversational interaction with computers. In *User-oriented design of interactive graphics systems*. New York: Association of Computing Machinery, pp. 101-113. Reprinted in Baecker, R.M & Buxton, W.A.S (1987) *Readings in human-computer interaction: A multidisciplinary approach*. Morgan Kaufmann.
- Nii, H.P. (1986). Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, Summer, 1986.
- Nilsson, N. (1980). *Principles of Artificial Intelligence*. San Mateo, California: Morgan Kaufmann.
- Nilsson, N. (1995). Eye on the prize. *AI Magazine* (16) 2, Summer 1995, pp.9-17.
- Nissen, M.J., Knopman, D.S., & Schacter, D.L. (1987). Neurochemical dissociation of memory systems. *Neurology*, 37, 789-794.
- Norvig, P. (1989). *Paradigms of AI Programming: Case Studies In Common Lisp*. San Mateo, California: Morgan Kaufmann.
- Orasanu, J. & Connolly, T. (1993). The Reinvention of Decision Making. In Klein, G. A., Orasanu, J., Calderwood, R., and Zsombok, C.E., eds., *Decision Making in Action: Models and Methods*. Ablex.
- Owens, C. (1989). Integrating feature extraction and memory search. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Northvale, NJ: Erlbaum.
- Parberry, I. (1987). *Parallel Complexity Theory*. John Wiley & Sons, 1987.

- Penberthy, J. & Weld, D. (1992). UCPOP: A sound, complete partial-order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, 103-114.
- Peot, M, and Smith, D. (1992). Conditional Nonlinear Planning. In *Proceedings of the First International Conference on AI Planning Systems*, p 189-197, June 1992.
- Peterson, J., Mahesh, K. & Goel, A. (1994). Situating natural language understanding within experience-based design. *International Journal of Human-Computer Studies* 41, pp. 881-913.
- Pirolli, P. (1997). Computational models of information scent-following in a very large browsable text collection. In *Proceedings of the Human Factors in Computing CHI '97 Conference*, Atlanta, GA, ACM Press.
- Pirolli, P. & Card, S. K. (1995). Information foraging in information access environments. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, Denver, May 1995, ACM.
- Pirolli, P. & Card, S. K. (1999). Information foraging. *Psychological Review* October 1999, Volume 106, Number 4.
- Pirolli, P., Pitkow, J. and Rao, R. (1996). Silk from a sow's ear: Extracting usable structures from the Web. In Bilger, R. Guest, S. and Tauber, M.J. (Eds.) *Proceedings of CHI-96*. ACM Press.
- Pitta, J. (1999). !&#\$.com. *Forbes*, August 23, 1999, pp76-77.
- Pollock, J.L. (1995). *Cognitive Carpentry: A blueprint for how to build a person*. MIT Press.
- Pryor, L. & Collins, G. (1994). Opportunities: a unifying framework for planning and execution. In *Proceedings of the Second International Conference on AI Planning Systems*, p170-175, June 13-15 1994, Chicago, Illinois.
- Pu, C. & Liu, L. (2000). Continual Query Project Description Page.  
<http://www.cse.ogi.edu/DISC/CQ/description.html>
- Purser, P., Faget, M. & Smith, N., Eds. (1964). *Manned Spacecraft: Engineering Design and Operation*. New York, New York: Fairchild.
- Pylyshyn, Z.W. (1991). The role of cognitive architecture in theories of cognition. In VanLehn, K. (Ed.), *Architectures for Intelligence*. Hillsdale: Lawrence Erlbaum Associates Inc.
- Quillan, M. R. (1968). Semantic Memory. In Minsky, M., (ed.) *Semantic Information Processing*, pp 227-270. Cambridge, MA: MIT Press.
- Quillian, M.R. (1962). A revised design for an understanding machine. *Mechanical Translation*, 1962, 7, pp. 17-29.
- Quillian, M.R. (1966). *Semantic memory*, Report AD-641671, Clearinghouse for Federal Scientific and Technical Information. Abridged version in Minsky, M. (ed.) (1968) *Semantic Information Processing*. Cambridge, MA: MIT Press.
- Quillian, M.R. (1967). Word concepts: a theory and simulation of some basic semantic capabilities. *Behavioral Science*, 1967, 12, pp. 410-430.
- Raaijmakers, J.G.W, & Shiffrin, R.M. (1981). Search of Associative Memory. *Psychological Review*, 88, 93-134.
- Rakeover, S.S. (1983). "In Defense of Memory Viewed as Stored Mental Representation." *Behaviorism: A Forum for Critical Discussion* 11(1):53-62.
- Ram, A. & Francis, A. G. (1996). Multi-Plan Retrieval and Adaptation in an Experience-Based Agent. In D. B. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, AAAI Press.

- Ram, A. & Hunter, L. (1992) The Use of Explicit Goals for Knowledge to Guide Inference and Learning. *Applied Intelligence*, 2(1):47-73.
- Ram, A. (1989). *Question-driven understanding: An integrated theory of story understanding, memory and learning*. Ph.D. Dissertation, YALEU/CSD/RR #70, Yale University, May 1989.
- Ram, A. (1991). A theory of questions and question asking. *Journal of the Learning Sciences*, 1(3&4): 273-318, 1991.
- Ram, A., & Francis, A. G. 1996. Multi-Plan Retrieval and Adaptation in an Experience-Based Agent, In D. B. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, AAAI Press.
- Ram, P. (1999) Problem-Based Learning in Undergraduate Instruction. A Sophomore Chemistry Laboratory. In *J. Chem. Educ.* 1999 76 1122
- Ram, P., Francis, A., Devaney, M., & Ram, A. (2000). Virtual Sherlock: an intelligent computer based PBL system for undergraduate chemistry. In *Proceedings of the Second International Conference on Problem-Based Learning in Higher Education*, Linköping, Sweden, September 17-20, 2000.
- Raphael, B. (1968) SIR: A computer program for semantic information retrieval. In Minsky, M. (ed.) (1968) *Semantic Information Processing*, pp. 33-134. Cambridge, MA: MIT Press.
- Raphael, B. (1968). SIR: A computer program for semantic information retrieval. In Minsky, M. (ed.) *Semantic Information Processing*. Cambridge, MA: MIT Press.
- Ray, E.J., Ray, D.S., and Seltzer, R. (1998). *The AltaVista Search Revolution, Second Edition*. Berkeley: Osborne/McGraw-Hill.
- Reder, L.M. (1987). Beyond associations: Strategic components in memory retrieval. In D. Gorfein & R. Hoffman (Eds.), *Memory and learning: The Ebbinghaus Centennial Conference*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 203-220.
- Reder, L.M. (1988). Strategic control of retrieval strategies. In G. Bower (Ed.), *The psychology of learning and motivation*, Vol. 22, New York: Academic Press, pp. 227-259.
- Redmond, M. (1990). Distributed cases for case-based reasoning: Facilitating use of multiple cases. In *Proceedings of AAAI-90*. Cambridge, MA: AAAI Press/MIT Press.
- Redmond, M. (1992). Learning by Observing and Understanding Expert Problem Solving. *Georgia Institute of Technology, College of Computing Technical Report no. GIT-CC-92/43*. Atlanta, Georgia.
- Reed, G. (1979). Everyday Anomalies of Recall and Recognition. In Khilstrom, J.F. & Evans, F.J. (eds). *Functional Disorders of Memory*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Rich, E. & Knight, K. (1991). *Artificial Intelligence*. 2e. New York, New York: McGraw-Hill.
- Riesbeck, C. (1993). Replacing CBR: Now What? Invited talk. *AAAI-93 Workshop on Case-Based Reasoning*, Washington, DC, July.
- Riggs, B. (1999). Knowledge finders. *Information Week*, May 24, 1999.
- Rissland, E. (1984). Ingredients of intelligent user interfaces. *International Journal of Man-Machine Studies* 21, pp. 377-388. Reprinted in Baecker, R.M & Buxton, W.A.S (1987) *Readings in human-computer interaction: A multidisciplinary approach*. Morgan Kaufmann.
- Robertson, S.E. & Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27, pp.129-146.
- Rogers, S. O. (1997). *Symbolic Performance & Learning in Continuous-valued Environments*. PhD Thesis, The University of Michigan, Dept. of Computer Science and Electrical Engineering, January 1997

- Rosenbloom, P.S. (1993). A symbolic goal-oriented perspective on connectionism and Soar. Reprinted in *The Soar Papers*, ed. J. Laird & A. Newell, 1993. Cambridge, MA: MIT Press.
- Rumelhart, D.E. & McClelland, J. L. (1986). Parallel distributed processing: Explorations in the microstructure of cognition. Cambridge, Massachusetts: Bradford/MIT Press.
- Rumelhart, D.E. & Norman, D.A. (1985) Representation of Knowledge. In Aitkenhead, A. M. & Slack, J. M. (eds.) *Issues in Cognitive Modeling*. London: Lawrence Erlbaum Associates.
- Russel, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Morgan Kaufmann.
- Salton, G. & Buckley, C. (1988). On the Use of Spreading Activation Methods in Automatic Information Retrieval. Technical Report, *TR88-907*, April 1988, Cornell University.
- Salton, G. & Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41, pp.288-297.
- Samuels, A.L. (1959) Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3, 211-229.
- Santos, E. Jr., Hguyen, H, and Brown, S.M. (2000). Medical document information retrieval through active user interfaces. In Arabina, H.R., (Ed.) *Proceedings of the International Conference on Artificial Intelligence IC-AI'2000*, Las Vegas, Nevada, June 26-29, 2000. CSREA Press.
- Schank, R. & Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- Schank, R. (1982). *Dynamic memory: a theory of reminding and learning in computers and people*. Cambridge: Cambridge University Press.
- Schubert, L. K. (1976). Extending the expressive power of semantic nets. *Artificial Intelligence* 7, pp. 163-198.
- Schubert, L. K. (1991). Semantic nets are in the eye of the beholder. In Sowa, J.F. (1991). *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pp. 95-107. San Mateo, California: Morgan Kaufmann.
- Shell, P. & Carbonell, J. (1989). *PARMENIDES: A Class-Based Frame System*. April 3, 1989. Carnegie Mellon University.
- Simina M. (1999). *Enterprise-directed Reasoning: Opportunism and Deliberation in Creative Reasoning*, Ph.D. Thesis, Georgia Tech
- Simina, M. & Kolodner, J. (1995). Opportunistic Reasoning: A Design Perspective. In *Proceedings of the 17th Annual Cognitive Science Conference*. Pittsburg, PA, July 1995.
- Simon, H.A. (1969). *Sciences of the Artificial*. Cambridge, Massachusetts: MIT Press.
- Simon, H.A. (1993). Artificial Intelligence as an Experimental Science. Invited Talk. Abstracted in *Proceedings of the Eleventh National Conference on Artificial Intelligence*, page 853, July 11-15, 1993.
- Smyth, B. & Keane, M. (1995). Remembering to Forget: A Competence-Preserving Deletion Policy for CBR Systems. In *Proceedings of IJCAI-95*.
- Sowa, J.F. (1984). *Conceptual Structures*. Reading, Massachusetts: Addison-Wesley.
- Sowa, J.F. (1991). *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. San Mateo, California: Morgan Kaufmann.
- Sparck Jones, K. & Willet, P. (Eds.) (1997). *Readings in Information Retrieval*. San Francisco: Morgan Kaufmann.

- Sparck Jones, K. (1979) Search term relevance weighting given little relevance information. *Journal of Documentation*, 35, 30-48. Reprinted in Sparck Jones, K. & Willet, P. (Eds.) (1997). *Readings in Information Retrieval*. San Francisco: Morgan Kaufmann.
- Stefik, M. (1995). *Knowledge Systems*. San Francisco, California: Morgan Kaufmann.
- Stefik (1981) Planning and metaplaning. In Nilsson and Weber, eds., *Readings in Artificial Intelligence*, pp. 272-286, Tioga.
- Sternberg, R. J. (1985). Teaching Critical Thinking: Are We Making Critical Mistakes? *Phi Delta Kappa*, November 1985, p194-198.
- Sternberg, R. J. (1986). *Intelligence Applied: Understanding and Increasing Your Intellectual Skills*. Harcourt, Brace, and Jovonovitch.
- Tambe, M. & Rosenbloom, P. S. (1994). Investigating production system representations for non-combinatorial match. *Artificial Intelligence*, 68(1).
- Tambe, M.; Doorenbos, R.; Newell, A. (1992) "The Match Cost of Adding a New Rule: A Clash of Views." *Technical Report CMU-CS-92-158*, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, June 1992.
- Tambe, M.; Newell, A.; Rosenbloom, P. S. (1990). The Problem of Expensive Chunks and its Solution by Restricting Expressiveness. *Machine Learning*, 5:299-348, 1990.
- Tannenbaum, A.S. (1984) *Structured computer organization*. Englewood Cliffs, N.J.: Prentice-Hall.
- Thagard, P., Holyoak, K., Nelson, G., & Gochfeld, D. (1990). Analogical retrieval by constraint satisfaction. *Artificial Intelligence*, 46, pp. 259-310.
- Turner, R. (1989). A schema-based model of adaptive problem solving. Ph.D. dissertation. Georgia Institute of Technology, School of Information and Computer Science Technical Report no. *GIT-ICS-89/42*. Atlanta, Georgia .
- Turner, S.R (1992). MINSTREL: A Computer Model of Creativity and Storytelling. PhD thesis, University of California, Los Angeles.
- Turner, S.R. (1994). MINSTREL. Lawrence-Erlbaum Associates.
- Turtle, H.R. & Croft, W.B. (1990). Inference networks for document retrieval. In Vidick, J.L. (Ed.), *Proceedings of the Thirteenth International Conference on Research and Development in Information Retrieval*, pp. 1-24. New York: Association for Computing Machinery. In Jones, K.S. & Willet, P. (Eds.), *Readings in Information Retrieval*. (1997). San Francisco: Morgan Kaufmann.
- Veale, T. (1995). *Metaphor, Memory and Meaning: Symbolic and Connectionist Issues in Metaphor Interpretation*. Ph.D. Thesis. School of Computer Applications, Dublin City University, Glasnevin, Dublin, Ireland.
- Vedasoftware (2000). Karnak. <http://www.karnak.com/>
- Veloso, M. & Blythe, J. (1994) Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the Second International Conference on AI Planning Systems*, p170-175, June 13-15 1994, Chicago, Illinois.
- Veloso, M. (1994). *Planning and Learning by Analogical Reasoning*. Springer-Verlag.
- Watson, I. & Perera, S. (1997) The evaluation of a hierarchical case representation using context-guided retrieval. In Leake, D.B. & Plaza, E. (Eds.) *Case-based reasoning research and development: Second International Conference on Case-Based Reasoning, ICCBR-97*, pp. 255-297. Providence, RI, USA, July 1997.

- Weibel, S., Kunze, J., Lagoze, C. and Wolf, M. (1998). Dublin Core Metadata for Resource Discovery, RFC 2413, September 1998. <ftp://ftp.isi.edu/in-notes/rfc2413.txt>
- Weizenbaum, J. (1966) ELIZA. *Communications of the Association for Computing Machinery*, 1966, 9, pp.36-45.
- Weld, D. (1994). An introduction to least-commitment planning. *AI Magazine*, (15) 4, Winter 1994, pages 27-61.
- Wickelgren, W.A. (1981). Human learning and memory. *Annual Review of Psychology*. 32, 21-52.
- Wilensky, R. (1984). Meta planning: representing and using knowledge about planning in problem solving and natural language understanding, *Cognitive Science*, 5, pp. 197-234. Reprinted in A. Collins & E.E. Smith, (eds.), *Readings in cognitive science: a perspective from psychology and artificial intelligence*. (1988). San Mateo: Morgan Kaufmann.
- Wilensky, R. (1986). Some Problems and Proposals for Knowledge Representation *Technical Report #UCB/CSD 86/294* Dept. of Electrical Engineering & Computer Science, University of California - Berkeley.
- Wilensky, R., Chin, D., Luria, M., Martin, J., Mayfield, J., and Wu, Dekai. (1988). "The Berkeley UNIX Consultant Project." *Computational Linguistics* 14(4):35-84.
- Williams, M.D. & Pottenger, W.M. (2000). The role of semantic locality in hierarchical distributed dynamic indexing. In Arabina, H.R., (Ed.) *Proceedings of the International Conference on Artificial Intelligence IC-AI'2000*, Las Vegas, Nevada, June 26-29, 2000. CSREA Press.
- Wills, L. M. & Kolodner, J.L. (1994). Towards More Creative Case-Based Design Systems. *AAAI*, Vol. 1 1994. pp. 50-55
- Winograd, G. (1992). Flashbulb memories of the San Francisco Earthquake. A talk given at the Cognitive Science Colloquium Series at Georgia Tech, April 3rd, 1992.
- Winograd, T. (1972). *Understanding Natural Language*. New York: Academic Press.
- Wolverton, M. & Hayes-Roth, B. (1993). Using controlled knowledge search to retrieve cross-contextual information. In *Working Notes of the AAAI Spring Symposium on Case-Based Reasoning and Information Retrieval — Exploring the Opportunities for Technology Sharing*, pp. 133-139, March 1993.
- Wolverton, M. (1994) *Retrieving semantically distant analogies*. Ph.D. Dissertation, Stanford University.
- Woods, W.A. (1975). What's in a Link: Foundations for Semantic Networks. In Bobrow, D.G. & Collins, A.M., (eds.), *Representation and Understanding: Studies in Cognitive Science*, pp. 35-82. New York: Academic Press.
- Wooldridge, M. & Jennings, N. (1995) Intelligent Agents: Theory and Practice. Submitted to *Knowledge Engineering Review* October 1994, Revised January 1995.
- Wu, W.X. & Dai, H. (2000). Discovering documents associations through growing cell structures. In Arabina, H.R., (Ed.) *Proceedings of the International Conference on Artificial Intelligence IC-AI'2000*, Las Vegas, Nevada, June 26-29, 2000. CSREA Press.
- Yahoo. (2000). [www.yahoo.com](http://www.yahoo.com)
- Young, R.M., Pollack, M.E., & Moore, J. D. (1994). Decomposition and causality in partial-order planning. In *Proceedings of the Second International Conference on AI Planning Systems*, p170-175, June 13-15 1994, Chicago, Illinois.
- Zamir, O. & Etzioni, O. (1998). Web document clustering: a feasibility demonstration. SIGIR-98.



Zilberstein, S. & Russel, S.J. (1993). Anytime sensing, planning and action: A Practical model for robot control. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, pp. 1402-1407, Chambéry, France.

Zilberstein, S. (1993). Operational Rationality through Compilation of Anytime Algorithms. Ph.D. dissertation, Computer Science Division, University of California at Berkley.

# CURRICULUM VITAE

## Dr. Anthony G. Francis, Jr.

**Origin: February 10, 1969, Greenville, South Carolina, USA, Earth**

### Primary Address

5 King's Tavern Place  
Atlanta, GA 30332-0280  
Phone (404) 351-4574

### College of Computing

Georgia Institute of Technology  
Atlanta, GA 30332-0280  
(404) 894-1077, fax 894-9846

### Internet

centaur@cc.gatech.edu  
[http://www.cc.gatech.edu/  
ai/students/centaur/](http://www.cc.gatech.edu/ai/students/centaur/)

## Objectives

To design and construct agent-based artificial intelligence systems, to study the impact of human memory and emotion upon the theory of human intelligence and the design of intelligent agents, and to pass on the fruits of this research program through teaching and the development of useful software artifacts.

### Research Areas

Artificial intelligence, specializing in memory, information retrieval, planning, agency, knowledge representation, and natural language understanding; cognitive psychology; emotion; animal cognition; robotic control; Internet search technologies; and human-computer interaction.

## Education

Georgia Institute of Technology

### Ph.D. in Computer Science (summer 2000)

- Thesis: "Context-Sensitive Asynchronous Memory"  
A theory of memory retrieval that achieves performance by interleaving retrieval and reasoning and using feedback from reasoning to guide memory search.
- Thesis Supervisor: Ashwin Ram
- Thesis Committee: Janet Kolodner, Ashok Goel, Kurt Eiselt, Nancy Nersessian
- Minor: Cognitive Psychology; Certificates: Cognitive Science

### Masters of Science in Computer Science (fall 1996)

### B.S. in Information and Computer Science (spring 1991)

- With Highest Honor

## Honors and Awards

### U.S. Air Force Laboratory Graduate Fellow (1992-1995)

- 3-year fellowship awarded by the Air Force Office of Scientific Research
- Sponsored by Human Resources Laboratory, Brooks AFB, Texas.
- Mentor: Dr. Mike Spector, AL/HRTI, Brooks AFB.

**National Merit Scholarship****(1987-1991)****Dow Chemical Corporation Sophomore Award****(1988)**

- Awarded to the outstanding sophomore student in Chemistry.

**Professional Experience**

Enkia Corporation

Atlanta, Georgia

**VP of Research and Development****fall 1998-present**

- Inventor of IRIA, an intelligent information management and recommendation system based on my Ph.D. work in context-sensitive asynchronous memory
- Principal Investigator on SBIR Phase I and II grants with Air Force Research Labs and Department of Education totaling \$850,000
- As part of management team closed angel and Series A investment rounds totaling \$2,250,000.
- Developed original IRIA core in Lisp and participated in development of commercial IRIA engine in cross-platform Java
- Managed research and development efforts, including proposal, paper and patent writing
- Managed intellectual property, legal, and corporate secretarial issues
- Co-Founded Enkia with Ashwin Ram (CEO) and Mark Devaney (VP of Product Development)

Yamaha Motor Corporation

Iwata, Shizuoka, Japan

**Visiting Researcher****summer 1998**

- Joint research with Yamaha Motor Corporation to present and extend the results of Georgia Tech's Personal Pet (Pepe).
- Designed an emotional long-term memory control system for a robot pet
- Implemented the system in both simulation and on a robot

Georgia Institute of Technology

Atlanta, GA

**Research Assistant****fall 1997-current**

- Worked with Ashwin Ram, Irfan Essa, and Juan Carlos Santamaria on the Personal Pet (Pepe) project
- Designed emotional model for an artificial pet based on current psychological research and robot control theory
- Advised and managed junior research assistants on the Pepe project.

SRI International

Menlo Park, CA

**Student Associate****summer 1997**

- Worked with Marie desJardins and Michael Wolverton on the Joint Maritime Crisis Action Planning (JMCAP) project
- Designed and implemented Patton, a case-based extension to the SIPE-2 planner.

Freelance

**Consultant****1995-present**

- Consulted with the Emory University ELITE Project on applications of AI technology to the Web (1995)

- Consulted with the Virtual Design Group on applications of AI technology to virtual reality (1995-1996)

Carnegie-Mellon University

Pittsburgh, PA

**Research Internship**

**summer 1996**

- Worked with Manuela Veloso to apply the experience-based agency theory to PRODIGY/ANALOGY
- Developed PRODIGY/ANALOGY load utility system.

Georgia Institute of Technology

Atlanta, GA

**Research Assistant**

**fall 1997**

- Worked with Ashwin Ram on the Introspective Multistrategy Reasoning project
- Participated in system design; participated heavily in the writing of grant proposals and project reports

Georgia Institute of Technology

Atlanta, GA

**Research Assistant/Programmer**

**1991-1992**

- Member of the Science Education Project design team
- Developed CLIM-based editor for case entry in the SciEd system.

The Sixteenth Annual Conference of the Cognitive Science Society

Atlanta, GA

**Chair, Animal Cognition Symposium**

**summer 1994**

- Organized and conducted the Animal Cognition Symposium.

**Graphic Designer**

- Designed and printed event tickets and proceedings stubs.

Georgia Institute of Technology

Atlanta, GA

**HTML Designer**

**1994-1996**

- Designed home pages for Artificial Intelligence (1994, 1996), Cognitive Science (1994, 1996), and Edutech (1994)
- Maintained AI/Cognitive Science Web Site (1994-1996)

Xitron Systems

Atlanta, GA

**Programmer**

**1989-1991**

- Developed and maintained financial aid database programs.

Digital Equipment Corporation

Greenville, SC

**Summer Intern Programmer**

**summer 1987**

- Developed and maintained process database access utilities.

## Teaching and Advising

Collegiate-Level Instruction

**CS3361: Introduction to Artificial Intelligence**

**fall 1995**

- Solo instructor for a junior-level core course with enrollment of 60.
- Redesigned Georgia Tech's AI course to fit a new textbook (Russel & Norvig 1995) which is now standard
- Prepared and graded all examinations and designed HTML course notes

**CS7342: Knowledge Structures for AI****spring 1994**

- Guest instructor on case-based reasoning and the PROTOS system.

Advising

**Advisor, Research Project****fall 1997**

- Advised a junior graduate student on implementing emotion models for the the Pepe artificial pet project

**Advisor, Research Project****winter 1997**

- Advised a graduate student on independent study to implement new planning domains in the Nicole/MPA system.

**Supervisor, Research Project****spring-fall 1996**

- Supervised an undergraduate on a multi-quarter research project to develop a customized AI simulation and integrate it with the Nicole system.

**Supervisor, Special Projects****winter 1996**

- Supervised an undergraduate on a quarter-long research project on interactive storytelling systems as part of a consulting project with the Virtual Design Group.

**Supervisor, Summer Internship****summer 1995**

- Supervised an undergraduate on a summer internship to implement a course registration advisor in the Nicole system.

**Selected Publications****Patents: 2 (Pending)****Invited Publications**

- Ram, A. & Francis, A.G. (1996) Multi-plan adaptation and retrieval in an experience-based agent. In David Leake, Ed., *Case-Based Reasoning: Experiences, Lessons, Future Directions*. MIT Press.

**Published Fiction**

- Francis, A.G. (1995) "Sibling Rivalry." (fiction) *The Leading Edge: Magazine of Science Fiction and Fantasy*, 30. Provo, Utah.

**Reviewed Publications**

- Francis, A.G., Ram, A., and Devaney, M. (2000). IRIA: The Information Research Intelligent Assistant. In Arabina, H.R., (Ed.) *Proceedings of the International Conference on Artificial Intelligence IC-AI'2000*, Las Vegas, Nevada, June 26-29, 2000. CSREA Press.
- Ram, P., Francis, A., Devaney, M., & Ram, A. (2000). Virtual Sherlock: an intelligent computer based PBL system for undergraduate chemistry. In *Proceedings of the Second International Conference on Problem-Based Learning in Higher Education*, Linköping, Sweden, September 17-20, 2000.
- desJardins, M., Francis, A.G., & Wolverton, M. (1998) Hybrid Planning: An approach to integrating generative and case-based planning. In *Case-Based Reasoning Integrations: Proceedings from the 1998 AAAI Workshop*, AAAI Technical Report WS-98-15, AAAI Press, 1998.
- Francis, A.G. & Ram, A. (1997) Can your architecture do this? A proposal for impasse-driven asynchronous memory retrieval and integration. In *Proceedings*

*of the AAAI-97 Workshop on Robots, Softbots and Immobots: Theories of Action, Planning and Control.*

- Francis, A.G. & Ram, A. (1995) Multi-plan adaptation and retrieval in an experience-based agent. In *Proceedings, AAAI 1995 Fall Symposium on Adaptation of Knowledge For Reuse.*
- Francis, A.G. & Ram, A. (1995) A Comparative Utility Analysis of Case-Based Reasoning and Control-Rule Problem Solving. In *Proceedings of ECML-95: the 8th European Conference on Machine Learning.*
- Francis, A.G., & Ram, A. (1994) A comparative utility analysis of case-based reasoning and control-rule learning systems. In *Proceedings of the AAAI-94 Workshop on Case-Based Reasoning.*
- Francis, A.G., & Ram, A. (1993) Computational models of the utility problem and their application to a utility analysis of case-based reasoning. In *Proceedings of the Workshop on Knowledge Compilation and Speedup Learning (KCSL 93)*, June 30, 1993
- Francis, A.G., & Ram, A. (1993) The Utility Problem in Case-Based Reasoning. Abstracted in *Case-Based Reasoning: Papers from the 1993 Workshop*, July 11-12, Washington, D.C., Technical Report WS-93-01, AAAI Press.

## Interests and activities

### Science Fiction Author (1987-present)

- published short story: "Sibling Rivalry", *The Leading Edge* (30) 1995.
- unpublished (but circulating!) novel *homo centauris*

### Artist / Creative Videographer (1985-present)

- Computer Skills: 2D, 3D, digital retouching, web graphics, HTML
- Primary Media: pencil/charcoal, pastel, oils
- Other media: videography, video editing

### Emory University Voice (1990-1991)

- Senior staff for Emory University's current events newspaper
- National News Editor 1990, Managing Editor 1991
- Wrote articles translated for college newspaper in Soviet Union

### Martial Arts

- 2<sup>nd</sup>-degree white belt in Taïdo under Andy Fossett / Mits Uchida
- 3<sup>rd</sup>-degree brown belt in Tae Kwon Do under Billy Hong

## References provided upon request