



Calhoun: The NPS Institutional Archive

Theses and Dissertations

Thesis Collection

2000-09-01

Modeling human and organizational behavior using a relation-centric multi-agent system design paradigm

Roddy, Kimberly A.

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/7739>



Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

**Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943**

<http://www.nps.edu/library>

NPS ARCHIVE
2000.09
RODDY, K.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE S
MONTEREY CA 93943-5050

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

**MODELING HUMAN AND ORGANIZATIONAL
BEHAVIOR USING A RELATION-CENTRIC MULTI-
AGENT SYSTEM DESIGN PARADIGM**

by

Kimberly A. Roddy
and
Michael R. Dickson

September 2000

Thesis Advisor:
Co-Advisor:

Michael Zyda
John Hiles

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

September 2000

3. REPORT TYPE AND DATES COVERED

Masters Thesis

4. TITLE AND SUBTITLE

Modeling Human And Organizational Behavior Using A Relation-Centric Multi-Agent System Design Paradigm

5. FUNDING NUMBERS

N0003900WRDR053

6. AUTHOR(S)

Roddy, Kimberly A. and Dickson, Michael R.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION
REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Navy Modeling & Simulation Office, CNO, N6M
2000 Navy Pentagon, Washington, DC 20350-2000

10. SPONSORING / MONITORING
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

Today's modeling and simulation communities are being challenged to create rich, detailed models incorporating human decision-making and organizational behavior. Recent advances in distributed artificial intelligence and complex systems theory have demonstrated that such ill-defined problems can be effectively modeled with agent-based simulation techniques using multiple, autonomous, adaptive entities. *RELATE*, a relation-centric design paradigm for multi-agent systems (MAS), is presented to assist developers incorporate MAS solutions into their simulations. *RELATE* focuses the designer on six key concepts of MAS simulations: relationships, environment, laws, agents, things, and effectors. A library of Java classes is presented which enables the user to rapidly prototype an agent-based simulation. This library utilizes the Java programming language to support cross-platform and web based designs. All Java classes and interfaces are fully documented using HTML Javadoc format. Two reference cases are provided that allow for easy code reuse and modification. Finally, an existing networked DIS-Java-VRML simulation was modified to demonstrate the ability to utilize the *RELATE* library to add agents to existing applications. LCDR Kim Roddy focused on the development and refinement of the *RELATE* design paradigm, while LT Mike Dickson focused on the actual Java implementation. Joint work was conducted on all research and reference cases.

14. SUBJECT TERMS

Multi-agent system, MAS, human and organizational behavior, agent-based simulation, adaptive agents, autonomous agents, relationship, RELATE, architecture

15. NUMBER OF PAGES

164

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT

Unclassified

20. LIMITATION OF ABSTRACT

UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**MODELING HUMAN AND ORGANIZATIONAL BEHAVIOR
USING A RELATION-CENTRIC MULTI-AGENT SYSTEM DESIGN PARADIGM**

Kimberly A. Roddy
Lieutenant Commander, United States Navy
B.S., Oregon State University, 1987

Michael R. Dickson
Lieutenant, United States Navy
B.S., Hawaii Pacific University, 1992

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN
MODELING, VIRTUAL ENVIRONMENTS AND SIMULATION**

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

PS Archive
2000.09
Roddy K.

~~Thesis
R665535
C.1~~

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Today's modeling and simulation communities are being challenged to create rich, detailed models incorporating human decision-making and organizational behavior. Recent advances in distributed artificial intelligence and complex systems theory have demonstrated that such ill-defined problems can be effectively modeled with agent-based simulation techniques using multiple, autonomous, adaptive entities. *RELATE*, a relation-centric design paradigm for multi-agent systems (MAS), is presented to assist developers incorporate MAS solutions into their simulations. *RELATE* focuses the designer on six key concepts of MAS simulations: relationships, environment, laws, agents, things, and effectors. A library of Java classes is presented which enables the user to rapidly prototype an agent-based simulation. This library utilizes the Java programming language to support cross-platform and web based designs. All Java classes and interfaces are fully documented using HTML Javadoc format. Two reference cases are provided that allow for easy code reuse and modification. Finally, an existing networked DIS-Java-VRML simulation was modified to demonstrate the ability to utilize the *RELATE* library to add agents to existing applications. LCDR Kim Roddy focused on the development and refinement of the *RELATE* design paradigm, while LT Mike Dickson focused on the actual Java implementation. Joint work was conducted on all research and reference cases.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	1
B.	GOALS.....	3
C.	ORGANIZATION	3
II.	BACKGROUND.....	5
A.	INTRODUCTION.....	5
B.	KEY CONCEPTS AND TERMS	6
1.	Agent	6
2.	Multi-Agent System (MAS).....	7
3.	MAS Simulations	8
4.	Relationship.....	10
5.	Coordination.....	11
6.	Adaptation	13
C.	ABBREVIATED HISTORY OF MULTI-AGENT SYSTEMS.....	14
1.	Holland.....	14
2.	Foundations of MAS	15
a.	Distributed Artificial Intelligence (DAI).....	16
b.	Artificial Life (A-Life)	17
3.	Significant MAS Simulations	18
a.	MACE	19
b.	Echo.....	20
c.	Swarm.....	21
d.	SimCity.....	22
e.	ISAAC	23
D.	SUITABILITY OF MULTI-AGENT SYSTEM SIMULATIONS	24
E.	SURVEY OF SIMILAR MAS SIMULATION ARCHITECTURES	25
1.	OAA	25
2.	JAFMAS	26
3.	Zeus	26
4.	JATLite.....	27
5.	DECAF Agent Framework.....	28
F.	MAS SIMULATIONS OF HUMAN AND ORGANIZATIONAL BEHAVIOR	29
1.	Sugarscape.....	29
2.	Iterated Prisoner's Dilemma.....	30
3.	Unscrupulous Diner's Dilemma.....	31
4.	TheSims™	32
G.	SUMMARY	33

III.	RELATE DESIGN PARADIGM AND ARCHITECTURE	35
A.	INTRODUCTION.....	35
B.	<i>RELATE</i> DESIGN PARADIGM.....	36
1.	Relationships	38
2.	Environment.....	40
3.	Laws	40
4.	Agents.....	41
5.	Things.....	41
6.	Effectors	42
C.	BALLOON ANALOGY	42
1.	Strings.....	42
2.	Balloons.....	43
3.	Finding a Balloon.....	44
4.	Buying a Balloon.....	44
5.	Balloons That Pull	45
6.	Life With Lots Of Balloons.....	46
D.	A RECIPE FOR MAS SIMULATIONS USING <i>RELATE</i>	47
1.	Define All Possible Relationships.....	47
2.	Identify Roles For Each Relationship	48
3.	Determine Goal/Rule/Action Types.....	49
4.	Determine Goals For Each Role.....	50
5.	Determine Rules For Each Goal.....	51
6.	Determine Feedback Mechanism For Each Goal.....	52
7.	Determine Credit Assignment For Each Rule.....	52
8.	Implement Design By Satisfying <i>RELATE</i> Interfaces	53
9.	Use Reference Cases As A Starting Point For GUI Development	53
E.	<i>RELATE</i> JAVA CLASS AND INTERFACE DEFINITIONS	54
1.	Public Class RelationshipManager.....	54
2.	Public Abstract Class Thing Extends Object	55
3.	Public Abstract Class Agent Extends Thing	56
4.	Public Interface Relationship	57
5.	Public Interface Role.....	57
6.	Public Interface Goal.....	58
7.	Public Interface Rule.....	58
8.	Public Interface Personality	58
9.	Public Interface SensedEnvironment	59
10.	Public Interface Sensor.....	59
11.	Public Interface Action.....	60
F.	SUMMARY	60

IV.	AN INTRODUCTORY MAS SIMULATION.....	61
A.	INTRODUCTION.....	61
B.	BRIAN ARTHUR'S EL FAROL BAR PROBLEM	62
C.	A <i>RELATE</i> RECIPE FOR THE EL FAROL BAR PROBLEM	63
1.	Relationships.....	63
2.	Roles.....	63
3.	Goal/Rule/Action Types.....	63
4.	Goals.....	64
5.	Rules.....	64
6.	Feedback Mechanism.....	64
7.	Credit Assignment.....	65
D.	A <i>RELATE</i> SOLUTION.....	67
1.	El Farol Rules.....	67
2.	Graphical Output.....	69
E.	SUMMARY	70
V.	ADDING AGENTS TO A NETWORKED DIS-JAVA-VRML SIMULATION.....	71
A.	INTRODUCTION.....	71
B.	CAPTURE THE FLAG	72
C.	A <i>RELATE</i> RECIPE FOR CTF AGENT	73
1.	Relationships.....	73
2.	Roles.....	73
3.	Goal/Rule/Action Types.....	73
4.	Goals.....	74
5.	Rules For Each Goal	74
6.	Goal Feedback Mechanisms.....	74
7.	Rule Credit Assignment	74
D.	A <i>RELATE</i> SOLUTION.....	74
1.	Red and Blue Start Panels	75
2.	Tank Agent.....	76
a.	Engaging Enemy Units.....	76
b.	Offensive Tank Agent	77
c.	Defensive Tank Agent.....	78
3.	Helicopter Agent	79
4.	Squad Relationship.....	80
5.	Future CTF Agent Work	82
E.	SUMMARY	83

VI.	A SITUATED LAND-COMBAT MODEL.....	85
A.	INTRODUCTION.....	85
B.	JACOB (SON OF ISAAC)	85
C.	A <i>RELATE</i> RECIPE FOR JACOB.....	86
1.	Relationships	86
2.	Roles.....	86
3.	Goal/Rule/Action Types.....	86
4.	Goals.....	87
5.	Rules For Each Goal	87
6.	Goal Feedback Mechanisms.....	87
7.	Rule Credit Assignment	89
D.	A <i>RELATE</i> SOLUTION.....	89
1.	Simulation Agent Editor.....	90
2.	Loading and Saving Environments	91
a.	Using An Open Environment.....	91
b.	Loading A Stored Environment	91
c.	Creating A New Environment.....	92
3.	Starting The Simulation	93
4.	Pausing The Simulation	94
5.	The Brain Lid	95
6.	Dynamic Goal Selection.....	96
a.	Forming A New Squad.....	97
b.	Waiting For Stragglers	99
7.	Agent Statistics.....	101
E.	SUMMARY	102
VII.	CONCLUSIONS AND RECOMMENDATIONS.....	103
A.	CONCLUSION	103
B.	RECOMMENDATIONS	103
	GLOSSARY	105
	APPENDIX A: SURVEY OF MAS SIMULATION ARCHITECTURES.....	107
	APPENDIX B: <i>RELATE</i> RELEASE NOTES	123
	APPENDIX D: <i>RELATE</i> DESIGN FOR CTF AGENT	127
	APPENDIX E: <i>RELATE</i> DESIGN FOR JACOB	129
	LIST OF REFERENCES	133
	INITIAL DISTRIBUTION LIST	139

LIST OF FIGURES

Figure 1.	Goal Satisfaction Representation.....	37
Figure 2.	Strings (Potential Relationships Names)	42
Figure 3.	Balloons (Relationships)	43
Figure 4.	Finding a Balloon (Joining an Existing Relationship)	44
Figure 5.	Buying a Balloon (Relationship Manager).....	45
Figure 6.	Balloons That Pull (Conflicting Goals).....	46
Figure 7.	Life With Lots Of Balloons (Multiple Relationships)	46
Figure 8.	Example Relationship Hierarchy and Role Assignment	48
Figure 9.	Action-Decision Loop.....	50
Figure 10.	Multiple Rules per Goal.....	53
Figure 11.	El Farol Personality	66
Figure 12.	Sample El Farol Rule Algorithms.....	68
Figure 13.	El Farol Attendance Output.....	69
Figure 14.	Red Start Panel with Agent Driven Selected	75
Figure 15.	Offensive Tank Agent	77
Figure 16.	Defensive Tank Agent	78
Figure 17.	Helicopter Agent.....	79
Figure 18.	Capture The Flag Agent Squad	81
Figure 19.	Simulation Agent Editor for JACOB	90
Figure 20.	Loading a JACOB Sample Environment.....	91
Figure 21.	Adding Floor Objects in JACOB.....	92

Figure 22.	JACOB Sample Screen Shot	93
Figure 23.	Battle Simulation in JACOB	94
Figure 24.	JACOB Brain Lid	95
Figure 25.	Blue Squad Approaching Stationary Red Agents	97
Figure 26.	Red Agents Forming Squad	98
Figure 27.	Red Squad Moves To Enemy Flag	99
Figure 28.	Waiting For Stragglers	100
Figure 29.	Agent Statistics in JACOB	101
Figure 30.	El Farol Design	126
Figure 31.	CTFAgent Design	128
Figure 32.	JACOB Design.....	131

LIST OF DEFINITIONS

Definition 1.	Agent.....	7
Definition 2.	Multi-Agent System (MAS).....	8
Definition 3.	MAS Simulation	10
Definition 4.	Relationship	11
Definition 5.	Coordination.....	12
Definition 6.	Adaptation.....	14

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF EQUATIONS

Equation 1.	Agent Goal Weight	88
Equation 2.	Leader Goal Weight	88
Equation 3.	Final Goal Weight	88

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS AND ACRONYM'S

2-D	Two Dimensional
3-D	Three Dimensional
ACL	Agent Communication Language
AI	Artificial Intelligence
A-Life	Artificial Life
BACH	Burks, Axelrod, Cohen, Holland
BT	British Telecom
CAS	Complex adaptive systems
CNA	Center for Naval Analysis
CTF	Capture The Flag
DAI	Distributed Artificial Intelligence
DECAF	Distributed, Environment-Centered Agent Framework
DIS	Distributed Interactive Simulation
DMSO	Defense Modeling and Simulation Office
DMVT	Distributed Vehicle Monitoring Test
DoD	Department of Defense
ENIAC	Electronic Numerical Integrator And Calculator
GA	Genetic Algorithm
GUI	Graphic User Interface
ICL	Interagent Communication Language
I/O	Input/Output
ISAAC	Irreducible Semi-Autonomous Adaptive Combat

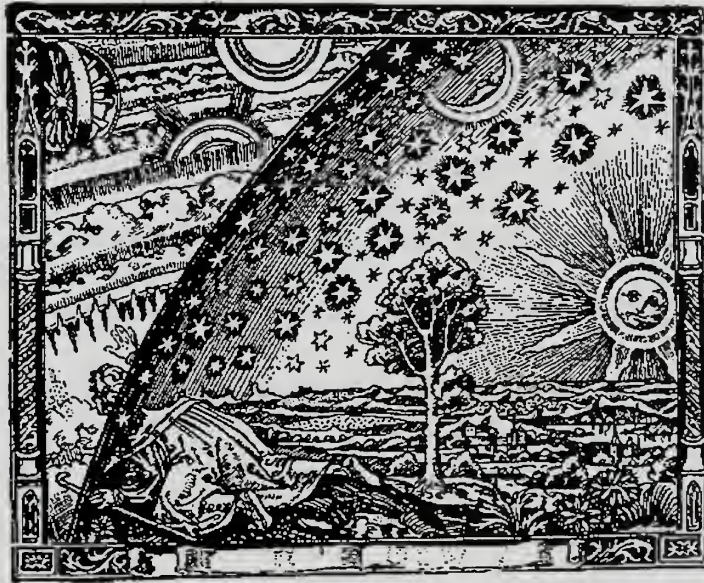
ISAACA	ISAAC Agent
JACOB	Just Another Complex Organizational Battlefield
JAFMAS	Java Agent Framework for MAS
JATLite	Java Agent Template, Lite
JWARS	Joint Warfare System
KQML	Knowledge Query and Manipulation Language
LAN	Local Area Network
LSVE	Large Scale Virtual Environment
MACE	Multi-Agent Computing Environment
M&S	Modeling and Simulation
MAS	Multi-Agent System
MOVES	Modeling, Virtual Environments and Simulation
NPS	Naval Postgraduate School
NRC	National Research Council
OAA	Open Agent Architecture
<i>RELATE</i>	Relationship, Environment, Laws, Agents, Things, Effectors
SFI	Santa Fe Institute
SRI	Previously Stanford Research Institute, now simply “SRI”
VE	Virtual Environment
VRML	Virtual Reality Modeling Language

ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support of the Navy Modeling and Simulation Office, N6M, for sponsoring our work and that of the MOVES Academic Group.

The authors would also like to thank Mr. John Hiles, who opened our eyes, directed our gaze, and joined us on our path of discovery. He provided insight and guidance during every phase of the development of this thesis and helped our agents get out of the box. We would also like to express our appreciation to Dr. Michael Zyda for his guidance and direction, not only as our thesis advisor, but as the Chair of the MOVES Academic Group. Other members of the MOVES Academic Group that provided invaluable assistance with Java code design issues include Dr. Michael Capps and Mr. Don McGregor.

Finally, we would like to thank our families for their patience, love, support, and sacrifices: Kim Roddy would like to express his appreciation to his wife, LCDR Sharon Roddy, USN, and their two daughters Elise and Nicole. Mike Dickson would like to thank his wife, Debbie for her endless support and understanding.



Famous woodcut appearing in
Camille Flammarion, *L'Atmosphère: Météorologie Populaire* (Paris, 1888), p. 163.

Although to penetrate into the intimate mysteries of nature and thence to learn the true causes of phenomena is not allowed to us, nevertheless it can happen that a certain fictive hypothesis may suffice for explaining many phenomena.

- LEONHARD EULER

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The modeling of cognition and action by individuals and groups is quite possibly the most difficult task humans have yet undertaken. Developments in this area are still in their infancy. Yet important progress has been and will continue to be made. Human behavior representation is critical for the military services as they expand their reliance on the outputs from models and simulations for their activities in management, decision making, and training.

– NATIONAL RESEARCH COUNCIL (1998)

A. MOTIVATION

Today's modeling and simulation (M&S) communities are being challenged by ever-increasing demands to create rich, detailed models of ill-defined problems. Most of these problems are complex because of the involvement of human decision-making and organizational behavior. Humans and organizations have multiple levels of internal roles, goals and responsibilities, frequently conflicting with each other. Humans are often "torn between two desires", attempting to satisfy the demanding responsibilities that come with the multiple roles often encountered in everyday life. While contemplating almost any decision, humans must evaluate a myriad of goals that they are currently attempting to achieve. These goals are sometimes supportive of each other, especially if an individual is well organized. Often these goals conflict, such as when a current goal a person is trying to achieve for his or her organization, e.g. finish a project tonight, conflicts with their current personal goals, e.g. spend the evening with my family. Developing simulations that are capable of capturing this complex, often

unpredictable, individual behavior is essential to realistically modeling large organizations accurately.

Models of unparalleled complexity are being constructed in an effort to capture key aspects of aggregate human behavior. Simulations currently in use range from decision support aides and project management trainers, to simulated cities and armies. A typical example is the theater level campaign model, such as the Joint Warfare System (JWARS), currently being developed by the Department of Defense (DoD). JWARS exhibits an unparalleled amount of combat realism and detail, but even its strongest proponents suggest the need for more work on the human and organizational behavior elements. Current JWARS decision-making mechanisms attempt to solve the problem by using expert systems, finite state machines, and finally, actually placing a human in the decision making loop (Maxwell & Raab, 1998). None of these methods can sufficiently model the cooperation, coordination and conflict which is often seen in systems with multiple, autonomous, adaptive entities. A better tool is needed to model both human decision-making and organizational behavior.

Recent research in the field of distributed artificial intelligence (DAI) and complex systems theory has demonstrated that ill-defined problems and complex systems can be effectively modeled using agent-based simulation techniques (Arthur, 1994). To satisfy many of today's M&S challenges, a number of agent-based languages, toolkits, and architectures have been developed. These tools provide the user the ability to investigate ill-defined problems with multi-agent systems (MAS), which propose a bottom-up, self-organized approach to modeling, sometimes referred to as *distillations*.

Although each agent-based tool is well suited to its particular design problem, most are either difficult to obtain or install, expensive to acquire, or often can't be applied or easily adapted to a specific problem. An easy-to-use, inexpensive (e.g. open-source), platform-independent, MAS simulation toolkit or library would be an invaluable resource. It would aid students and researchers attempting to incorporate software agents and MAS into their models and simulations. In conjunction, a straightforward design paradigm would help developers leverage the power of what is widely becoming known as *Agent-Oriented Programming*.

B. GOALS

Three main goals were taken up by this thesis, as summarized below:

- Develop a relation-centric MAS design paradigm that focuses on the relationships that exist between agent and other things in the simulation.
- Design a Java-based MAS library based on the relation-centric design paradigm that enables rapid prototyping and development of models that simulate human and organizational behaviors.
- Demonstrate the functionality of this architecture by implementing models of increasing complexity and scope, both situated and non-situated, as reference cases.

C. ORGANIZATION

Chapter II is a review of background material and similar work supporting this thesis. It develops key concepts and definitions while presenting a short history of agent-

based simulations. It also addresses the suitability of using MAS in addition to, or instead of other modeling techniques. A survey of similar MAS simulation architectures and toolkits is also presented, focusing on the original intention of the architecture being surveyed. A sample of current MAS simulations used to model human and organizational behavior is also provided for comparison to this work. Chapter III introduces the *RELATE* design paradigm for building MAS simulations. It also provides a detailed description of the development package of Java classes and interfaces presented in this thesis, including class definitions and methodology of implementation. Chapter IV describes the development of an introductory MAS simulation using the presented *RELATE* design paradigm, illuminating key aspects of the development package. Chapter V describes the development of a situated MAS simulation of a simplistic, two-dimensional battlefield consisting of two armies with hierarchical organizational structure. Chapter VI describes the incorporation of agents into an existing networked DIS-Java-VRML simulation using the *RELATE* design paradigm and library. Chapter VII provides conclusions and recommended future work. A glossary of terms is included for easy reference and look-up of commonly used terms.

A number of Appendices are provided giving additional details and references for this body of work. Appendix A is a survey of current MAS simulation architectures including brief descriptions and web pointers to each. Appendix B provides release notes for the presented *RELATE* Java classes, interfaces, and reference cases. Appendices C, D, and E give details and illustrations of the three reference cases.

II. BACKGROUND

Carl Hewitt recently remarked that the question *what is an agent?* is embarrassing for the agent-based computing community in just the same way that the question *what is intelligence?* is embarrassing for the mainstream AI (Artificial Intelligence) community. The problem is that although the term is widely used, by many people working in closely related areas, it defies attempts to produce a single universally accepted definition.

– MICHAEL WOOLDRIDGE & NICHOLAS JENNINGS

A. INTRODUCTION

Any discussion regarding MAS simulations should first be founded on a common understanding, or at least acceptance, of key terms and concepts. Unfortunately, many of the commonly used terms in the fields of DAI and MAS research do not have commonly agreed upon definitions by the research communities. Entire papers have been written addressing the concept of agency and attempting to find a common meaning of over a dozen major researchers, e.g. (Franklin & Graesser, 1996). Many experts acknowledge the difficulties of rigidly defining agency, and instead provide a list of properties (Wooldridge and Jennings, 1995), or describe agents as multi-dimensional (Nwana, 1996). Section B of this chapter attempts to clarify specific key concepts and terms that are significant for this body of research. A glossary of terms is provided at the end of the thesis for quick reference to these and other frequently referred to terms. Armed with a common vocabulary, it is also useful to have a basic knowledge of where this rapidly growing field of research has come from, where it is now, and where it is presumably going in the future. A short history of MAS research is given in Section C. For a more

detailed history, the reader is directed to (Russell and Norvig, 1995), (Weiss, 1999), and (Ferber, 1999). Section D presents a discussion of the suitability of MAS simulations and the type of applications they are best used for. Section E provides a brief summary of Java-based MAS architectures and libraries that are similar to the current work, with comments as to how they compare and contrast with the proposed package. Section F provides selected examples of MAS simulations currently being used to model human and organizational behavior. A summary of this chapter is provided in Section G.

B. KEY CONCEPTS AND TERMS

1. Agent

The remarks by Carl Hewitt at the beginning of this chapter (Wooldridge and Jennings, 1995) are substantiated in the comments of many books, papers, and articles whenever there is an attempt to define the term *agent* (Knapik & Johnson, 1998). Each definition includes many of the same basic concepts, but also often adds to, or omits from, the ‘consensus’ definition. One of the most commonly referenced definitions is in the respected text by Russell and Norvig (1995) that states, “an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.” Other leading researchers believe that satisfying objectives or goals is also a necessary attribute of an agent, e.g. (Maes, 1990 and 1995), (Ferber, 1999), (Wooldridge and Jennings, 1995). Russell & Norvig actually distinguish this as a *goal-based agent*, which is one of four types of agents that build on their basic

agent definition. The other three are: *simple reflex agents*; *agents that keep track of the world*; and *utility-based agents*.

An agent can be a software object, a robot, a living being, or anything that fulfills the basic concepts of agency. In the context of this thesis, the use of the word *agent* will always imply *software agent* as opposed to any other kind, unless specifically stated. The following definition of an agent will be used:

Agent: A software object that perceives its environment through sensors and acts upon that environment through effectors to achieve one or more goals.

Definition 1. Agent

For an in-depth comparison of leading researcher's definitions of agency, as well as descriptions of many additional ways to classify agents, such as reactive, autonomous, mobile, etc., see (Franklin and Graesser,1996). To explore the diversification in the types of agents being investigated, see (Nwana,1996).

2. Multi-Agent System (MAS)

I'll call "Society of Mind" this scheme in which each mind is made of many smaller processes. These we'll call *agents*. Each mental agent by itself can only do some simple thing that needs no mind or thought at all. Yet when we join these agents in societies--in certain very special ways-- this leads to true intelligence.

– MARVIN MINSKY

One of the most commonly recognized uses of the term *agent* can be traced to Marvin Minsky's famous book, "The Society of Mind" (1985). In the context of his book, Minsky's use of the term *agent* was effectively equated to *process*, which is slightly different than the use by computer scientists today. The short quote above indicates that Minsky's work was possibly more closely related to a collection of agents or MAS, rather than on individual agents. His work exploring how the human mind actually works did lend some credibility, however, to the hypothesis that human decision-making can be effectively modeled using MAS simulation techniques. Similar to the term *agent*, it is difficult to find a commonly accepted definition of MAS. Huhns and Stephens (1999) give the following characteristics of multi-agent environments: they have communication and interaction protocols; they are self-organizing; they contain distributed, autonomous agents that may be self-interested or cooperative. Ferber (1999) defines MAS as comprising of the following elements: environment, objects, agents, relations, operations, and laws. For the purposes of this thesis, the Weiss definition of MAS will be used (Weiss, 1999):

<p>Multi-agent system (MAS): A system in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks.</p>
--

Definition 2. Multi-Agent System (MAS)

3. MAS Simulations

The terms *modeling* and *simulation* are often used together, frequently interchangeably, and sometimes incorrectly. Merriam-Webster's "OnLine" dictionary

defines the word model in many different ways (Merriam-Webster, 2000). The word can be used as a noun, verb, or adjective, giving a slightly different meaning. The uses most suitable to this field of research include the following:

- (noun) - A model is a description or analogy used to help visualize something (as an atom) that cannot be directly observed.
- (noun) - A model is a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs.
- (verb) - To produce a representation or simulation of.
- (adjective) - Being a usually miniature representation of something.

Models used in most modern simulations are based on mathematical underpinnings, describing relationships between system variables that represent physical aspects of reality. Common modeling methods include differential equations, rule-based “if-then” systems, and other more specific methods tailored to the subject being modeled. A classical example are “Lanchester Equations,” or LEs, which are a set of coupled ordinary differential equations modeling attrition in modern warfare. Ilachinski (1997) discussed various shortcomings of LEs when attempting to capture individual behavior in land combat models. By contrast, agent-based models are “a way of doing thought experiments,” the goal of which is to “enrich our understanding of fundamental processes that may appear in a variety of applications” (Axelrod, 1997). Holland (1998) discusses *dynamic models* and states that the object of creating such models is “to find unchanging laws that generate the changing configurations.” He points out that these laws correspond roughly to the rules of a game.

Simulations attempt to model reality over a period of time. The National Research Council (1998) describes simulations as methods for implementing a model to play out the represented behavior over time. Ferber (1999) describes simulations as methods to analyze real-world properties of theoretical models. A classic AI view of computer simulations is that they can be thought of as problem-solving processes that attempt to predict the future state of a real system by studying an idealized computer model (Widman and Loparo, 1989). Simulations are used for more than just predicting future state. They are also be used to practice and rehearse problem-solving skills (Thinking Tools, 1999), to help provide insight and understanding of complex systems (Holland, 1995 & 1998), to “study life as it could be” (Langton, 1989), or to simply entertain in either a realistic or fantasy manner. The list of uses of computer simulations is almost as long as the list of developers creating them. Agent-based simulations in particular are used “to study the emergent properties of the interactions among agents” (Axelrod, 1997). The following definition of a MAS simulation will be used (Hiles, 1999):

<p>MAS Simulation: A rich, bottom-up modeling technique that uses diverse, multiple agents to imitate selected aspects of the real world system’s active components.</p>

Definition 3. MAS Simulation

4. Relationship

The word *relationship* can be used to mean slightly different things due to the multiplicity of the English language. The use of relationship in this thesis does not

directly refer to kinship nor, necessarily, the physical juxtaposition of objects. Rather, Ferber's (1999) definition is taken that a relationship is the "assembly of relations that link certain individuals to others." The next step is to define *relation*. Again, turning to Merriam-Webster (2000), one finds seven different uses of the word. Significant to this research, it is important not to confuse the uses that imply association or resemblance such as in the phrase 'the *relation* of time and space' or the uses that imply a mathematical property between an ordered pair of objects such as that expressed by *is equal to* or *is less than*. Instead, a relation in the context of this thesis, satisfies the following properties:

- The attitude or stance which two or more persons or groups assume toward one another.
- The state of being mutually or reciprocally interested.

The following definition will be used for this thesis:

Relationship: The assembly of relations, i.e. understandings and/or commitments, between mutually interested parties that link certain individuals to others.

Definition 4. Relationship

5. Coordination

DAI primarily focuses on coordination as a form of interaction that is particularly important with respect to goal attainment and task completion. The purpose of coordination is to achieve or avoid states of affairs that are considered as desirable or undesirable by one or several agents.

- GERHARD WEISS

As the next section will establish, there is a strong tie between DAI and MAS, and the quote above could just as easily be said about MAS. Coordination can include cooperation or competition (or conflict), depending if the elements of the system are working together to achieve a common goal, or more concerned with maximizing individual performance, even at the expense of others (Weiss, 1999). Malone (1988) uses the broader, common sense definition of coordination: “the act of working together harmoniously,” for the basis of his work. He goes on to list the following common problems of coordination theory:

- How can overall goals be subdivided into actions?
- How can actions be assigned to groups or to individual actors?
- How can resources be allocated among different actors?
- How can information be shared among different actors to help achieve the overall goals?

Ferber (1999) interprets Malone’s work describing the coordination of actions as “the set of supplementary activities which need to be carried out in a multi-agent environment, and which a single agent pursuing the same goals would not accomplish.” Malone’s more narrow definition of coordination is used for this thesis:

<p>Coordination: The act of managing interdependencies between activities performed to achieve a goal.</p>

Definition 5. Coordination

6. Adaptation

If an agent uses feedback to modify its decision-making process, it is no longer simply reacting to its environment, it is also adapting to form a better fit to it. Learning and adaptation are closely related, with learning actually being a means to adaptation (Scott, 1958). Merriam-Webster (2000) defines learning in this context as:

- Knowledge or skill acquired by instruction or study.
- Modification of a behavioral tendency by experience (as exposure to conditioning).

When one talks of adaptation of a species, the time scale is now much longer, spanning multiple generations, and is usually considered evolution. Merriam-Webster (2000) defines evolution as:

- A process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state.
- The historical development of a biological group (as a race or species).

Adaptation encompasses both evolution, on a population or macro scale, and learning, on an individual or micro scale. When an individual organism adapts to its environment, it is generally considered learning. When a species adapts to its environment, it is considered evolutionary change. Adaptation implies a modification according to changing circumstances (Merriam-Webster, 2000). The following definitions apply:

- Adjustment to environmental conditions.

- Modification of an organism or its parts that makes it more fit for existence under the conditions of its environment.

The following definition for adaptation will be used for this thesis:

Adaptation: The process of modifying ones behavior over time to advantageously form a better fit to the environment.

Definition 6. Adaptation

C. ABBREVIATED HISTORY OF MULTI-AGENT SYSTEMS

Agent-based computing and MAS techniques are relatively young fields that have only been in use for about a decade. Many look at John Holland's work modeling complex adaptive systems (CAS) as the defining research for the beginning of the fascinating new fields of artificial life (A-Life) and agent-based computing. Holland was one of the founding members of the "BACH" group at the University of Michigan (named after the original members---Arthur Burks, Robert Axelrod, Michael Cohen and John Holland), a small group of researchers from a variety disciplines who shared an interest in complex adaptive systems of all kinds (Festschrift, 1999). The following paragraph about Holland and his work serve as an introduction into the history of this field.

1. Holland

Receiving a Ph.D. in Communication Sciences from the University of Michigan in 1959, John Holland became interested in combining concepts from psychology, specifically cognitive science, with mathematics and computer science. While the

mainstream AI community was focused on neural nets, symbolic AI, and expert systems, Holland concentrated on methods related to machine learning. At the time, there were no mathematical theories associated with learning, but there were mathematics associated with adaptation (Fisher, 1958). Holland proposed that learning and adaptation were very much the same, distinguished mostly by the time scale, as discussed in section B above. He developed genetic algorithms (GA) in the 1960's and used them to help model CAS. Holland's subsequent work led him to the development of classifier systems: rule based models in which genetic algorithms can be applied to change rules. (Stites, 1997)

2. Foundations of MAS

The multi-agent approach lies at the crossroads of several disciplines. The two most important ones are distributed artificial intelligence (DAI), the purpose of which is to create organizations of systems capable of solving problems by means of reasoning most generally based on the manipulation of symbols; and artificial life (A-Life), which seeks to understand and model systems possessing life, that is, capable of surviving, adapting and reproducing in sometimes hostile surroundings. — **JACQUES FERBER**

In keeping with the above statement by Ferber (1999), the following sections briefly describe DAI and A-Life, two of the most influential disciplines on MAS research and development today. As a historical tool, these paragraphs focus on the key researchers in each area. A summary of key MAS simulations is then presented.

a. Distributed Artificial Intelligence (DAI)

DAI began to emerge in the late 1960's and early 1970's as a new branch of study for AI researchers. Carl Hewitt's concurrent actor model and his work on *open systems* is probably the most recognized work associated with the birth of DAI (Hewitt, 1977). A concurrent actor can be described as an object that carries out its actions in response to the communications it receives and its current behavior. The similarities to this description and the current concept of agency is so close that more than a decade after publishing his paper on concurrent actors, he described DAI as beginning with "attempts to apply and extend the study of 'Intelligent Agents' to cover activities that are distributed in space and time" (Hewitt and Inmann, 1991). Victor Lesser is also widely associated with foundational work in DAI with his work in the early 1970's on the blackboard system and the first Distributed Vehicle Monitoring Test (DVMT). Lesser concentrated on communication, cooperation, and negotiation among multiple agents (Ferber, 1999). Les Gasser's Mace system is also largely associated with pioneering work in DAI in the late 1980's. When attempting to define DAI, one runs into the same problems of a distinct lack of a consensus definition. Gasser states that "(DAI) is a sub-field of AI concerned with the problems of describing and constructing multiple 'intelligent' systems which interact" (Gasser et al., 1987). Russell and Norvig (1995) state that when using the rationalist approach, AI can be viewed as "the study and construction of rational agents." Extending this to include the distributed sense, DAI might well be defined as 'the study and construction of *distributed* rational agents.'

Weiss (1999) makes no distinction between the terms MAS and DAI system, and uses them synonymously. He defines DAI as, "...the study, construction, and application of multiagent systems...."

b. Artificial Life (A-Life)

John von Neumann's work in cellular automata, and Norbert Wiener's formulation of *cybernetics*, both in the late 1940's and 1950's have often been cited as foundational work for A-Life and MAS (Ferber, 1999). In his introduction to Von Neumann's work (Von Neumann, 1966), Arthur Burks, the inventor of the first multi-purpose electronic computer ENIAC, writes, "(Von Neumann's) 'theory of automata' formed a coherent body of concepts and principles concerning the structure and organization of both natural and artificial systems, the role of language and information in such systems, and the programming and control of such systems".

Chris Langton, a student of John Holland, invented the term *artificial life*, or A-Life in the late 1980's while conducting research at the Santa Fe Institute (SFI), and is the person most associated with this field. A-Life is concerned with "abstracting the underlying principles of the organization of living things and implementing them in a computer so as to be able to study and test them" (Langton, et al., 1990). In the early 1990's, Langton began working with a team of researchers at the Santa Fe Institute (SFI) on the development of Swarm, a multi-agent software platform designed to simulate CAS. Swarm leveraged the power of MAS by employing a collection of independent agents interacting via discrete events. The Swarm toolkit was written with no domain

specific requirements and as such, simulations have been written using Swarm for everything from ecosystems to economics, from physics and chemistry to political science (Minar et al., 1996). More information on Swarm is available in C.3 below as well as Appendix A.

Craig Reynolds is also a popularly recognized name in the field of A-Life, due in large part for his work with “boids.” Boids are artificial birds that exhibit behaviors much like that seen in a flock of birds in nature. This behavior is generated by the incorporation of three simple rules in each agent (Reynolds, 1999):

- *Separation*: steer to avoid crowding local flockmates.
- *Alignment*: steer towards the average heading of local flockmates.
- *Cohesion*: steer to move toward the average position of local flockmates.

Reynolds calls his work, “Individual-based models,” which are simulations based on the global consequences of local interactions of members of a population (Reynolds, 1999). His work involving modeling natural aggregate behavior such as schools, flocks and herds, including (Reynolds, 1982) and (Reynolds, 1987), earned him the Scientific and Engineering Award in 1997 from the Academy of Motion Picture Arts and Sciences (Academy, 1997).

3. Significant MAS Simulations

In the course of development of the MAS research field, certain key MAS simulations have been influential to students and researchers alike. The following simulation architectures are some of the most well know and referenced works by

academia, defense, and commercial research organizations. See Appendix A for a more complete list of existing MAS simulation architectures.

a. MACE

I think many people talk about agents without clearly specifying what may differentiate agents from other kinds of programmed entities. So pretty often the discussion about agents focuses on things that don't seem to be a great deal different from distributed objects.... I think a more direct way to see the prospects for a concept like 'agents' is to go back to the history of programming and locate the concept of agent in a historical progression of programming—a progression of techniques for description and action. And once we do that, maybe we can see what may be the contribution of agents, as a part of an evolution of programming languages and programming technologies.

- LES GASSER

Gassers comments above (Briot, 1998) provide a reasonable justification for exploring his work in this section. Les Gasser developed MACE (Multi-Agent Computing Environment), a language, programming environment, and test-bed for DAI systems, in the late 1980's. The goal of MACE was to "support experimentation with different styles of distributed AI systems, at different levels of complexity" (Gasser, et al., 1987). Widely cited and known worldwide as a "classic" DAI system, MACE introduced several concepts now used in virtually all experimental platforms in the field, including social-level reasoning, model composition, and role-based coordination (Gasser, 2000). MACE consisted of a collection of components including the following types: agents, system agents, facilities, a description database, and kernels. For more details on MACE, see (Gasser, et al., 1987).

b. Echo

Echo is a simulated world, somewhat like SimCity, but not nearly as concrete. It's a world in that there's a geography with different resources at different places... The resources are merely letters: resource A and B. One place has a lot of A, and another a lot of B. Now, I have these 'agents' in Echo—you could think of them as simple organisms—that move around. They have limited capabilities. In early models two agents could come together and decide to trade some resources. They carry a reservoir, a stomach, that can carry resources. One agent might have plenty of A but need B, and another might have a lot of B and need A. They'd meet and trade.

- JOHN HOLLAND

The above quote was taken from an interview with John Holland about the origins of A-Life (Stites, 1997). Echo is a simulation tool “developed to investigate mechanisms which regulate diversity and information-processing in systems comprised of many interacting adaptive agents, or CAS” (Echo, 2000). Interactions between agents in Echo include combat, trade and mating. Echo agents develop strategies to ensure survival in resource-limited environments. Rules for interactions are encoded in individual genotypes. In a typical simulation, populations of these genomes evolve interaction networks, which regulate the flow of resources. Resulting networks resemble species communities in ecological systems. Flexibly defined parameters and initial conditions enable researchers to conduct a range of "what-if" experiments. For further information on Echo, the reader is directed to (Holland, 1995) and (Echo, 2000).

c. Swarm

Many biologists have speculated wistfully about “rewinding the tape” of evolution, starting the process over again from slightly different initial conditions. What would emerge? What would be the same? What would be different? We sense that the evolutionary trajectory that did in fact occur on earth is just one out of a vast ensemble of possible trajectories—each leading to a biology that could have happened in principle, but didn’t in fact solely for reasons of accident combined with common genetic descent. We sense that the regularities we seek would be revealed to us if we could just get a glimpse of that space of possible biologies.

- CHRIS LANGTON

The quote above, taken from the editor’s introduction to (Langton, 1997), illustrates the power of MAS as applied to the study of A-Life. Chris Langton, and fellow researchers at the Santa Fe Institute, developed Swarm in the mid 1990’s, with a beta release in 1996. Swarm is a software package for multi-agent simulation of complex systems and is intended to be a useful tool for researchers in a variety of disciplines. The basic architecture of Swarm is the simulation of collections of concurrently interacting agents. Swarm supports both discrete event and time stepped models as well as a variety of generic methods for tapping data from components of the system, combining those data through statistical filters, and displaying then with generic visualization objects or saving them to files. Swarm was initially developed in the Unix operating environment and programmed in objective C. Recently, Swarm has been ported to a variety of operating systems and programming languages. For more information on Swarm, see (Swarm, 2000).

d. SimCity

With SimCity we weren't trying to make a realistic simulation of a city; we were trying to do a caricature. We exaggerated a lot of things to bring them to the forefront so you notice the relationships between different factors.

- WILL WRIGHT

This introductory quote was from an interview with Wright on Gamecenter.com (1999). Will Wright began working on an idea he had for a 'City Simulator' in 1985. His idea was to create a first-of-its-kind game that would allow the user to create and control a city as a system. He co-founded Maxis with Jeff Braun in 1987 and released SimCity™ in 1989 (IBM, 1997). This new game was unlike any other computer game on the market, using amazing new technology that let the user take control of a big area of land and attempt to build a thriving metropolis. Although the exact technology is still kept closely guarded, it is speculated that some form of cellular automata and/or agent-based simulation is at the heart of the game (Hiles, 1999). The citizens of these simulated cities are called *sims*, and are often believed to be agent-driven. The tiles that represent land and structures are also very "agent-like." This simulation is included in this section for two important reasons. It is probably the most well know simulation of this kind by researchers and non-researchers alike. SimCity and the numerous follow-on games of this type (SimEarth, SimLife, SimAnt, etc) influenced

the research direction of a number of simulation efforts. For more information on all of the Maxis “Sim”-line of games, see (Maxis, 1999) and (Maxis, 2000).

e. ISAAC

Perhaps the single most important lesson of the new sciences is the observation that the collective decentralized interaction among individual agents obeying local rules often appears locally disordered but induces – on a higher level – a globally ordered pattern of behavior. The central thesis of this report...is that the general mechanisms responsible for emerging patterns in complex adaptive systems can be used to further our insight into the patterns of behavior that arise on the real combat battlefield. That is, that *land combat can be modeled as a complex adaptive system.*

- ANDREW ILACHINSKI

As the above quote (Ilachinski, 1997) indicates, ISAAC (Irreducible Semi-Autonomous Adaptive Combat) was one of the first military research projects to openly attempt to model land combat using agent-based simulation techniques. Completed in 1997, Ilachinski’s model represented combatants as red or blue squares on a two-dimensional field of battle. Each agent, or ISAACA, moved based on its current status of alive or injured, and on it’s propensities to move towards the following objectives: alive friendly, alive enemy, injured friendly, injured enemy, own flag, or enemy flag. The goal of ISAAC was to take a bottom-up, synthesist approach to the modeling of combat, vice the more traditional top-down, or reductionist view. ISAAC represented a first step toward developing a “complex systems theoretic analyst's toolbox (or "conceptual playground") for exploring high-level emergent collective patterns of behaviors arising

from various low-level (i.e., individual combatant and squad-level) interaction rules.” For more information on ISAAC and the follow-on project, EINSTEIN, see (ISAAC, 2000).

D. SUITABILITY OF MULTI-AGENT SYSTEM SIMULATIONS

According to Ferber (1999), MAS simulations are one of the five main applications of MAS. Other applications include: problem solving; building artificial worlds; collective robotics; and program design. MAS simulations use a bottom-up approach to modeling complex, ill-defined situations. That is to say, they leverage the emergent behavior of a collection of individually acting agents to allow the discovery and exploration of the possible, underlying, base rules that exist. Axelrod describes agent-based modeling as an inductive analysis tool that aids intuition and enriches ones “understanding of fundamental processes that may appear in a variety of applications” (Axelrod, 1997). John Holland’s classifier systems are used to “get a better handle on cognition” (Stites, 1997). By treating the rules that they are based on as hypotheses, and allowing the exploration of new combinatorial rules thru GA, one can explore alternate rule-bases that generate similar (or different) emergent behavior. This is extremely useful when trying to gain an understanding of the potential reasons of observed behavior of a CAS. In terms of A-Life, MAS simulations can be used to study how computational techniques can model biological phenomena, as well as how biological techniques can help shed new methods of solving computational problems (Liekens, 2000).

E. SURVEY OF SIMILAR MAS SIMULATION ARCHITECTURES

The following MAS simulation architectures, packages and toolkits were surveyed for a comparison to the *RELATE* design goals. They are all Java-based and freely available for download. Most of the information provided here was summarized or excerpted from the various package websites as indicated. For a more complete survey of these and other existing architectures and libraries, see Appendix A.

1. OAA

Adam Cheyer, David Martin, and colleagues, developed the Open Agent ArchitectureTM (OAA[®]) at SRI International in the mid 1990's. They focused on building distributed *communities* of agents, where they defined an agent as “any software process that meets the conventions of the OAA society.” An agent satisfies this requirement by registering the services it can provide by utilizing an “Interagent Communication Language” (ICL), and by sharing functionality common to all OAA agents, such as the ability to install triggers, manage data in certain ways, etc. OAA exhibits the following characteristics, taken directly from the OAA web site, to achieve it's objective of providing a framework for integrating a community of heterogeneous software agents in a distributed environment (SRI, 2000):

- *Open*: agents can be created in multiple programming languages and interface with existing legacy systems.
- *Extensible*: agents can be added or replaced individually at runtime.
- *Distributed*: agents can be spread across any network-enabled computers.

- *Parallel*: agents can cooperate or compete on tasks in parallel.
- *Mobile*: lightweight user interfaces can run on handheld PDA's or in a web browser using Java or HTML and most applications can be run through a telephone-only interface.
- *Multimodal*: When communication with agents, handwriting, speech, pen gestures and direct manipulation (GUIs) can be combined in a natural way.

OAA 1.0 agent libraries have been ported to a number of programming languages, including Java.

2. JAFMAS

Developed by Deepika Chauhan at the University of Cincinnati in 1997, JAFMAS provides a framework to guide the coherent development of MAS along with a set of classes for agent deployment in Java. The JAFMAS methodology follows five stages: agent identification, definition of each agent's conversations, determining the rules governing each agent's conversations, analyzing the coherency between all the conversations in the system, and implementation. Only four of the provided Java classes must be extended for any application. (JAFMAS, 2000)

3. Zeus

Hyacinth Nwana and other members of British Telecom (BT) Laboratories' Intelligent Systems Research (ISR) Group, under the Agents Research Programme, developed the Zeus Agent Building Toolkit in the mid 1990's. Zeus is an integrated environment for the rapid development of collaborative agent applications. It is entirely

implemented in Java and will run on all major hardware platforms. The goal of Zeus developers was to create a toolkit that would facilitate the rapid design, development and deployment of agent systems. Zeus was designed by incorporating three main functional components, as described in (Zeus, 2000):

- The Agent Component Library - A collection of software components that implement the functionality necessary for multi-agent systems. This library provides a set of high quality, pre-written and pre-tested agent components that “liberate developers from the minutiae of agent technology, allowing them to concentrate on solving their application's problems instead.”
- The Agent Building Tools – A number of editors designed to “guide developers through the stages of the comprehensive agent development methodology.” These editors include: ontology, agent definition, task description, organization, and coordination editors.
- The Visualization Tools – These include the runtime environment that enables applications to be observed and, where necessary, debugged. The tools collect information on agent activity, interpret it and display various aspects in real-time. The “Visualiser” consists of the following tools: society viewer, reports tool, statistics tool, agent viewer, and a control tool.

4. JATLite

Created at Stanford University, JATLite (Java Agent Template, Lite), is a package of programs written in Java that is designed to allow users to quickly create new software

agents that robustly communicate over the Internet. JATLite's infrastructure registers agents with an "Agent Message Router" facilitator. Registration is done using a name and password, and allows agents to connect/disconnect from the Internet, send and receive messages, transfer files, and invoke other programs or actions on the various computers where they are running. JATLite agents send and receive messages using KQML (Knowledge Query and Manipulation Language), a language and protocol for exchanging information and knowledge that is being developed as part of the ARPA Knowledge Sharing Effort (KQML, 2000). The communications are built on open Internet standards. Developers using JATLite are able to build agent systems using other agent languages as well. (JATLite, 2000)

5. DECAF Agent Framework

Developed at the University of Delaware by Keith Decker, John Graham, and a team of graduate students, DECAF (Distributed, Environment-Centered Agent Framework) is a toolkit that provides a stable platform to "design, rapidly develop, and execute intelligent agents to achieve solutions in complex software systems." DECAF is conceptually thought of as an *agent operating system* that provides the following agent services: communication, planning, scheduling, execution monitoring, coordination, and eventually learning and self-diagnosis. The goals of the architecture are as follows (DECAF, 2000):

- Develop a modular platform suitable for research activities.
- Allow for rapid development of third-party domain agents.

- Provide a means to quickly develop complete multi-agent solutions using combinations of domain-specific agents and standard middle-agents.
- Take advantage of the object oriented-features of the JAVA programming language.

F. MAS SIMULATIONS OF HUMAN AND ORGANIZATIONAL BEHAVIOR

In addition to Echo, Swarm and SimCity, discussed above, a number of other MAS simulations have successfully captured key aspects of human and organizational behavior. A few of the most significant simulations are summarized here with most information provided from on-line resources as indicated.

1. Sugarscape

In 1996 Josh Epstein and Robert Axtell published “Growing Artificial Societies: Social Science from the Bottom Up,” as part of the 2050 Project, a joint venture of the Santa Fe Institute, the World Resources Institute, and the Brookings Institution (Axtell & Epstein, 1996). They proposed a new model, Sugarscape, which simulates the behavior of artificial people (agents) located on a landscape of a generalized resource (sugar). The agents have vision, a metabolism, a speed, and other genetic attributes. They move around the landscape, in the direction of the largest concentration of sugar visible to them, and eat the sugar to replace energy consumed by motion. Agents die if and when they burn up all their sugar. Epstein and Axtell conducted a variety of experiments on this artificial society, including adding seasons, which caused the agents to hibernate, and a

second resource (spice) which allowed the agents to trade and compete for resources, creating an emerging market. (Sugarscape, 2000)

2. Iterated Prisoner's Dilemma

The two-person iterated Prisoner's Dilemma is the *E. coli* of the social science, allowing a very large variety of studies to be undertaken in a common framework. It has even become a standard paradigm for studying issues in fields as diverse as evolutionary biology and networked computer systems. Its very simplicity has allowed political scientists, economists, sociologists, philosophers, mathematicians, computer scientists, evolutionary biologists, and many others to talk to each other. Indeed, the analytic and empirical findings about the Prisoner's Dilemma from one field have often led to insights in other fields.

- ROBERT AXELROD

In 1984, Robert Axelrod developed a multi-agent simulation of the classic social problem known widely as the Two-Person Prisoner's Dilemma (Axelrod, 1984). The Prisoner's Dilemma is a situation that represents two people arrested for a crime. They are put in separate interrogation rooms, led to believe that they will certainly be convicted of the crime, but that they might receive a reduced sentence if they testify against the other. If they do testify, the other accomplice will receive a much harsher sentence. Each individual can either cooperate, by not implicating the accomplice, or defect, by providing testimony against the partner. If these same individuals meet a number of times in the same situation, one might be able to recognize a pattern of behavior of the other and take advantage of it. In this manner, the strategic situation becomes the *Iterated Prisoner's Dilemma*. Axelrod demonstrated through a series of experiments and

tournaments that an agent-based evolutionary simulation could effectively model human decision-making. Follow-on work described in (Axelrod, 1997) continued to delve into organizational behavior and the issue of social norms.

3. Unscrupulous Diner's Dilemma

This lighthearted situation, which we call the Unscrupulous Diner's Dilemma, typifies a class of serious, difficult problems that pervade society. Sociologists, economists and political scientists find that this class of social dilemma is central to a wide range of issues, such as protecting the environment, conserving natural resources, eliciting donations to charity, slowing military arms races and containing the population explosion. All these issues involve goals that demand collective effort and cooperation. The challenge is to induce individuals to contribute to common causes when selfish actions would be more immediately and personally beneficial.

- NATALIE GLANCE & BERNARDO HUBERMAN

Glance and Huberman (1994) expanded work on the Prisoners Dilemma by looking at the complex interactions that might take place when a group of people meet regularly to dine at a fine restaurant. As often happens, there is an unspoken agreement to divide the check evenly. Some people will reasonably choose an average cost meal, while others may take advantage of the situation by picking a more expensive meal. Glance and Huberman call this the "Unscrupulous Diner's Dilemma." Where Axelrod's work focused on individual decision-making, Glance and Huberman focused on social and organizational behavior, and in particular, the impact of relationships in that organizational structure or hierarchy. They also demonstrated that even the complex

interactions and behaviors represented in this situation could be effectively modeled with MAS.

4. TheSims™

Like our everyday world, the world of the Sims requires judgment and decision-making, in affairs from the trivial to the life threatening. Just as we learn to adapt to the full scope of our world's challenges, so must you guide your Sims, from their breakfast selection to their career track. And as you'll see, they do some decision-making on their own, and sometimes you might want to pull your hair out from watching what they come up with.

THESIMS USER MANUAL

As early as 1994, Will Wright began working on a new project that he referred to as “Dollhouse” (Wired, 1994). He envisioned being able to zoom down into SimCity, all the way down to street level, to see the people interacting with each other by talking and gesturing (Hopkins, 2000). In late 1999, Maxis released TheSims™, which allows the user to “create and control people” in the simulation (Maxis, 2000). TheSims™ allows you to create characters and balance the following personality traits: neat, outgoing, active, playful, and nice. As the individual character goes on about the life you generate for it, various changing needs are displayed: hunger, comfort, hygiene, bladder, energy, fun, social, and room. Wright successfully incorporated shifting goals and changing moods, creating a very convincing, and entertaining representation of human behavior.

G. SUMMARY

This chapter highlighted several key definitions and concepts as they are used in this thesis. Many of these terms are not rigorously defined in the research community, but a common understanding, or at least temporary acceptance, is important, nonetheless. An abbreviated history of MAS was given to establish a common reference point. The work of John Holland and other significant researchers was emphasized, and the connections of MAS research to DAI and A-Life was established. A review of significant MAS simulations was also included. A short discussion was provided on the suitability of MAS simulations and solutions, concluding that, as Robert Axelrod states, they are best suited for doing “thought experiments.” A survey of MAS simulation architectures similar to *RELATE* was provided. In general, each of these architectures were shown to focus more on mobile agents and communication methods than on MAS simulations. The chapter concluded with examples of MAS simulations that effectively modeled some form of human decision-making and/or organizational behavior.

The next chapter presents *RELATE*, a relation-centric MAS design paradigm and associated Java library of classes that is tailored to assist developers create rich, hierarchical simulations that realistically represent human decision-making and organizational behavior.

THIS PAGE INTENTIONALLY LEFT BLANK

III. *RELATE* DESIGN PARADIGM AND ARCHITECTURE

The man who is striving to solve a problem defined by existing knowledge and technique is not just looking around. He knows what he wants to achieve, and he designs his instruments and directs his thoughts accordingly.... To be accepted as a paradigm, a theory must seem better than its competitors, but it need not, and in fact never does, explain all the facts with which it can be confronted.

- THOMAS KUHN (1962)

A. INTRODUCTION

This chapter presents *RELATE*, a relation-centric design paradigm for building MAS simulations. *RELATE* helps the simulation developer capture the complex interdependencies of human decision-making and organizational behavior. The name itself is actually a mnemonic device that reminds the developer to focus on relationships, as well as other key aspects of MAS. *RELATE* stands for relationships, environment, laws, agents, things, and effectors. This design is based largely on Ferber's definition of a MAS (Ferber, 1999). Modifications have been made to allow the use of the mnemonic and to shift the focus on relationships. Section B presents the details of the design paradigm and describes the importance of each of the key areas represented by the mnemonic title. Section C provides an simple analogy as a means to more easily understand the important concepts of *RELATE*, as well as how the design paradigm is implemented. Section D demonstrates a recommended technique to design MAS simulations using *RELATE*. Section E presents a brief description of each of the classes and interfaces in the Java implementation of the design paradigm.

B. *RELATE* DESIGN PARADIGM

The *RELATE* design paradigm proposes that an effective way to model the complex, human decision-making process. It focuses on how an individual *relates* to other things and individuals within its environment. By concentrating on the relationships of individuals and within organizations, the developer is encouraged to identify the various roles that are assumed by members belonging to each relationship. These roles usually have certain responsibilities and commitments, which tend to be manifested as additional goals that must be addressed by the various members of the relationship. Goals can often be categorized into different types, with each type of goal requiring action that is independent of all other types. For instance, a goal to move to a certain point (movement goal type) can be accomplished independently from a goal to shoot at all hostile forces that are detected (shooting goal type), or to keep members of a unit informed (communication goal type). These are examples of the three distinct goal types that are used in the second reference case detailed in Chapter V. Once an agent is a member of a relationship, it must base its action selection on its personality, or its particular concern for each goal, the state of achievement of each goal, and, possibly, its understanding of its superior's desire to fulfill the goal. This often leads to conflicting courses of action that the agent must resolve. If the goals are typed to match different, non-conflicting action types, such as moving and communicating, they can be selected with unique, independent mechanisms. Figure 1 shows an example of how these different goals and their associated goal types might be represented. Goals that are being

satisfied might not need much attention and are green. Goals that are being neglected or for some reason are not being satisfied may need more attention and are red.

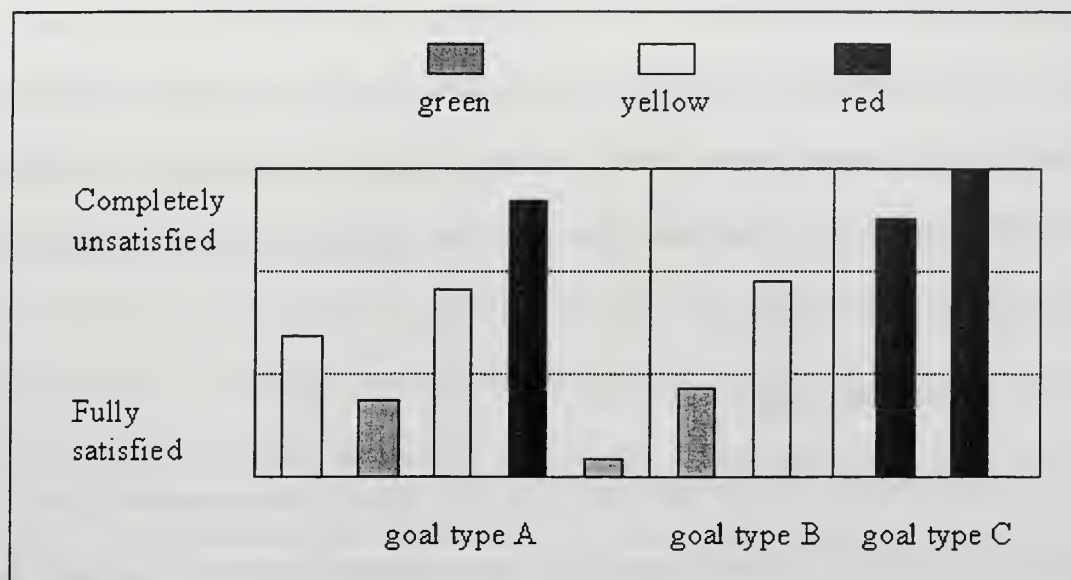


Figure 1. Goal Satisfaction Representation

Each agent must determine which goal is the active goal in each set of goal types in its goal list. Goal satisfaction is determined by evaluation of the sensed environment, taking into account the agent's personality, and possibly the direction or guidance from another agent in a common relationship, such as orders from a superior. If a particular goal an agent has is far from being satisfied, but is not very important based on the agent's personality, the goal may not be significant enough to become the active goal. A superior agent that cares about this same goal may exert influence on the subordinate agent, raising the goal's importance level. Based on the agents' personality trait to follow orders, it may shift an otherwise insignificant goal to a much higher importance level. Goal achievement is measured by some mechanism that provides feedback to the

individual through its sensed environment. As one goal is fulfilled, other goals may suffer. At some point, the attainment, or lack thereof, of certain goals may reach a threshold that requires the agent to shift its current active goal to avoid an undesirable state. This dynamic goal selection is one of the most significant features of the *RELATE* design. It allows the agent to selectively choose different goals to adapt to changing environmental factors. The following sub-sections highlight the key concepts of the *RELATE* design paradigm.

1. Relationships

To paraphrase the definition provided in Chapter II, relationships are “the assembly of relations... between mutually interested parties that link certain individuals to others.” In the *RELATE* paradigm, these are the relations connecting or binding agents to each other that result in the assignment of new roles, goals, and responsibilities.

Relationships are often formed to achieve something that is not achievable by any one individual. Shared resources and abilities often allow the individual agent to satisfy a goal it would otherwise not be able to achieve. These new abilities are usually attained at the cost of additional commitments and responsibilities incurred by the relationship.

Often a common goal is established upon formation of a relationship that was not previously held by any one individual agent. Relationships sometimes give member-agents new capabilities, such as the ability to create an offspring, or the ability to complete a coordinated task such as lifting an object too heavy for one person to lift. If

there is an overall benefit to the individual, there usually exists a desire to form the relationship.

In the *RELATE* design, an agent always attempts to fulfill any relationship that it can. To do this, it first needs to be aware that it is capable of forming a certain relationship. The agent then needs to sense the appropriate agents and/or things necessary to form the relationship. In the *RELATE* Java package, this is all handled automatically by the relationship manager, a unique, static, “singleton” agent that is responsible for verifying that prerequisites are met prior to instantiating each relationship. Once the relationship is instantiated, the relationship manager assigns the members and “releases” the relationship. The relationship is an independent agent that issues roles to each member, constantly monitors its members, and ensures minimum conditions are maintained to continue the relationship. Each role contains specific goals and rules. Since an agent can belong to more than one relationship, it can also have multiple roles. Each of these roles may have one or more goals, possibly of the same type. Goals of the same type compete for the attention of the agent. Since they require the same type of action to satisfy them, only one goal can be the *active goal* at any given time. The active goal is determined by a number of factors including the personality of the agent, the feedback from the environment on the state of achieving each goal, and any outside influences that may encourage one goal be given a higher priority than another. These goals are achieved by utilizing one of the rules, associated with the specific goal, to select an appropriate action. If more than one rule is provided to accomplish the same goal, some form of credit assignment is used to indicate which rules are more successful than

others. In this manner, genetic algorithms can be used to improve the agents rule set over time.

The relationship monitors its members and the environment to ensure conditions are maintained to continue the relationship. If there are “openings” for additional members, and a new agent seeks to join the relationship, the available role is issued and the new agent is accepted. If conditions are not maintained, the relationship withdraws all of its assigned roles, disbands the relationship and destroys itself.

2. Environment

The environment of a MAS simulation is the situated or non-situated space in which all things, including agents, exist. Rather than thinking of the environment as landscape or terrain, think of it as the collection of things and agents that interact with each other. Environments are very specific to the application and must be defined by the developer. In the three reference cases provided in Chapters IV, V, and VI, the environments include the un-situated collection of club members and a bar, a situated two-dimensional battle field containing two armies, and a three-dimensional, networked, virtual environment.

3. Laws

The limitations and restrictions in the specified environment placed on the things in the environment. Ferber refers to laws as the “reaction of the world to (attempts) at modification” (1999). Laws are not necessarily specified as a concise set of rules that must be complied with. More often, laws are intertwined into the simulation. Specific

examples might include issues related to physically based modeling such as collision detection, gravity and light propagation. Other examples might have to do with the ways relationships are formed and destroyed. Laws are probably the most intangible aspect of MAS simulations.

4. Agents

As defined in Chapter II above, an agent is “a software object that perceives its environment through sensors and acts upon that environment through effectors to achieve one or more goals.” In a *RELATE* simulation, agents are *things* that can take action, on themselves and other things in their environment, to satisfy internal goals based upon their perceived environment. Clearly, a MAS must contain more than one agent. The true power of MAS simulations is derived from the interaction between agents, often while achieving common goals.

5. Things

These are the base-level objects in the environment. All agents in the environment are also objects, or things. Things have the ability to represent themselves, either two-dimensionally (2-D) or three-dimensionally (3-D), or simply as text strings. Things also have the ability to update themselves over time. A thing can be influenced or modified by other things, including agents, in the environment. Examples of things that aren't agents include static objects such as rocks, bridges, and mountains, and dynamic objects such as rivers, machines (like a revolving door or an ATM), and even a complicated vehicle, assuming that an agent is required to operate it.

6. Effectors

These are the means by which an agent causes effects (interacts) in its environment. Effectors include sensors and operators. One can think of effectors as the Input/Output (I/O) methods for an agent. When designing agents for MAS, the developer needs to consider what sensing abilities they should be capable of, as well as what operators or actions they can employ. These effectors can be specifically type-matched to the goal/rule pairs for ease of implementation.

C. BALLOON ANALOGY

When a new methodology or tool is presented, it is of little use if few people completely understand it. As a training tool, the *RELATE* design paradigm and associated Java classes might best be understood by considering the following simplistic analogy:

1. Strings

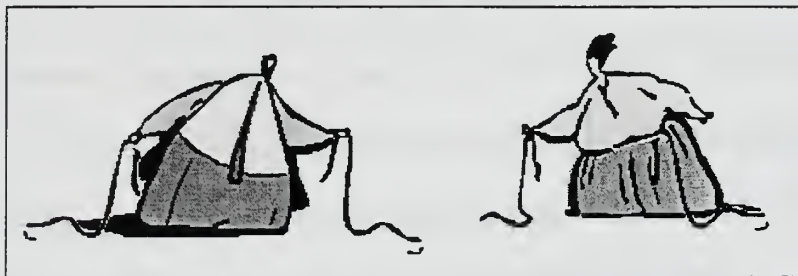


Figure 2. Strings (Potential Relationships Names)

Agents in a *RELATE* simulation can be thought of as holding a number of balloon strings, waiting for the right balloons to show up (Figure 2). The strings are the names of potential relationships that the agent is capable of joining or forming.

2. Balloons

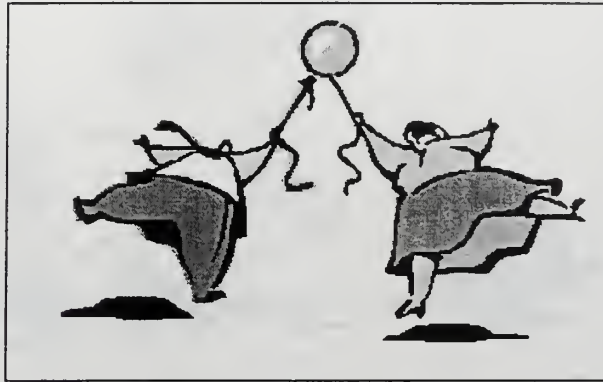


Figure 3. Balloons (Relationships)

The balloons are the actual relationships (Figure 3). An agent is constantly looking around, trying to find an existing balloon to tie its string to, or asking if a balloon can be created for it. The agent does this by passing its sensed environment to the relationship manager and asking if it can join an existing relationship or form a new one of its known potential relationships. Once a relationship is formed, the balloon passes information down the string to the agent. In particular, the relationship issues roles and provides a means for members of the relationship to communicate with each other. The balloons, or relationships, are actual agents themselves. Once they are instantiated by the relationship manager, they issue roles to each of the members of the relationship and constantly monitor the conditions to maintain the relationship. If conditions are not maintained, they remove all of the associated roles from the members and destroy themselves, leaving the agent's strings empty again.

3. Finding a Balloon

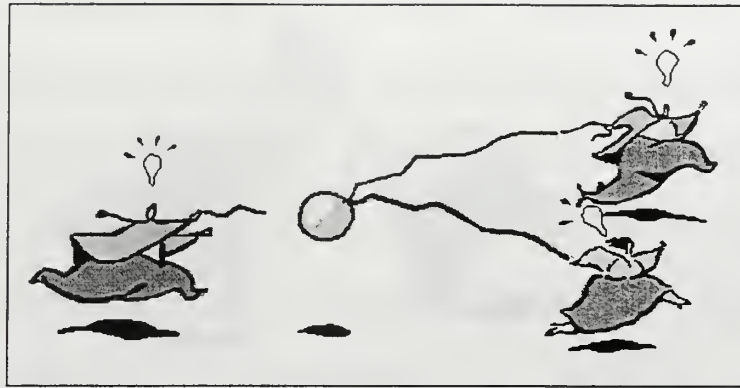


Figure 4. Finding a Balloon (Joining an Existing Relationship)

The easiest way to get a balloon is to find one that is already available, then just attach a string. If an agent is looking for a relationship for one of its strings, or known potential relationships, it will first try to join existing relationships (Figure 4). To accomplish this, before the relationship manager creates a new relationship, it looks at the agent's sensed environment to see if the agent can detect other agents that are already in the same relationship, and that have room for another agent. If the relationship has room, the searching agent is added. If not, the relationship manager continues to look for non-full relationships.

4. Buying a Balloon

The relationship manager is an agent too, but one that exists at the heart of the *RELATE* simulation. The relationship manager is responsible for keeping track of all the agents trying to form relationships. It is like the clown at a carnival, overseeing the crowd, selling and blowing up the balloons, and tying the strings to the requesting agents.

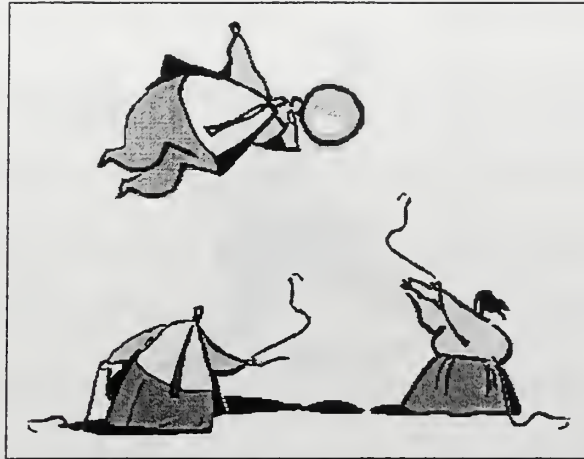


Figure 5. Buying a Balloon (Relationship Manager)

These balloons don't exist until the relationship manager verifies the prerequisites are met to form them by using the balloons, or relationships, themselves to determine the pre-requisites. Once all of the required conditions are met, the relationship manager blows up the balloons, or instantiates the relationships, and associates them with the designated agents (Figure 5).

5. Balloons That Pull

An agent may belong to a number of different relationships, each assigning various roles and their associated goals. Sometimes the different roles compliment each other and the agent finds that it can accomplish all of its goals without conflict. If the goals are slightly different, or worse, contradict each other, it could cause the agent to be "pulled" in different directions as it tries to satisfy all of its responsibilities. The agent must then prioritize its goals based on its personality, the current state of achievement

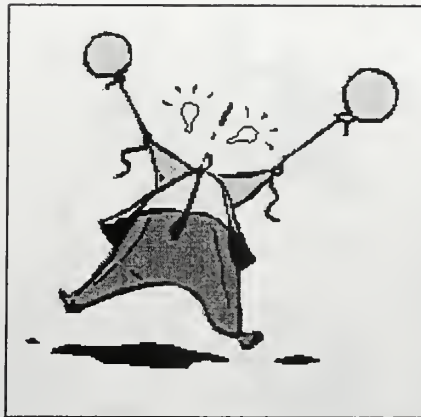


Figure 6. Balloons That Pull (Conflicting Goals)

of its goals, and often, outside influences encouraging the pursuit of one goal over another, such as orders or direction from a superior (Figure 6).

6. Life With Lots Of Balloons

A *RELATE* agent is created with a list of known potential relationships. Until all of these relationships are fulfilled, the agent will continue to attempt to join or form these relationships (Figure 7). It's like the agent is running around, carrying a bunch of strings,

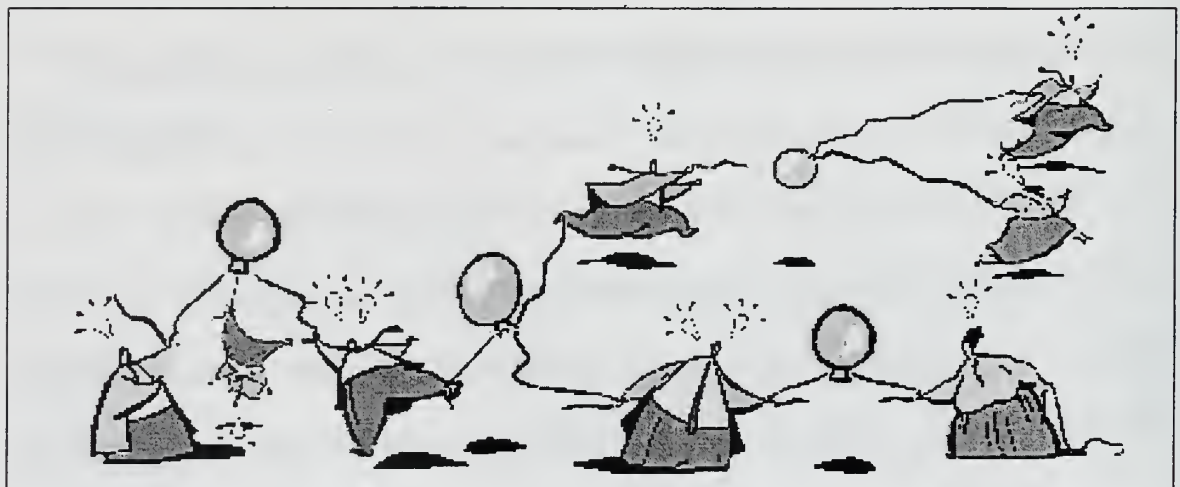


Figure 7. Life With Lots Of Balloons (Multiple Relationships)

some with balloons, and others just hanging down. The agent is constantly looking for the right kind of balloons to tie to its empty strings. The end result is a self-organizing collection of agents that automatically form organizations from the bottom up. Each relationship adds additional roles and associated goals, some cooperating, some conflicting. At the micro, or individual level, each agent has a very realistic, dynamic set of goals that it is trying to achieve. At the macro level, a fully formed organizational structure and aggregate behavior emerges, providing complex and often unpredictable results, similar to real life.

D. A RECIPE FOR MAS SIMULATIONS USING *RELATE*

As a concrete example of a MAS simulation using the *RELATE* design paradigm, consider a simulation that is attempting to model small scale, company strength battles on a simplified battlefield. Using *RELATE*, the following design method, or “recipe” could be used.

1. Define All Possible Relationships

In this simulation, two small groups of soldiers will be created. Each group of soldiers will come from a different army, so the *army* relationship will be needed, with two instances, *redArmy* and *blueArmy*. The largest organization displayed on the simulation will be a company of soldiers, so the *company* relationship is needed. There is no need to distinguish between red and blue company, because, as will be seen below, each company will only consist of soldiers from the same army. Smaller divisions might be appropriate, such as *platoon* and *squad* relationships. At the very lowest level, there may

be a need to include a *buddy* relationship. Prerequisites and maintaining conditions must be established for each of these relationships. These can include attributes such as minimum and maximum number of members, distance between members, various capabilities of members, etc.

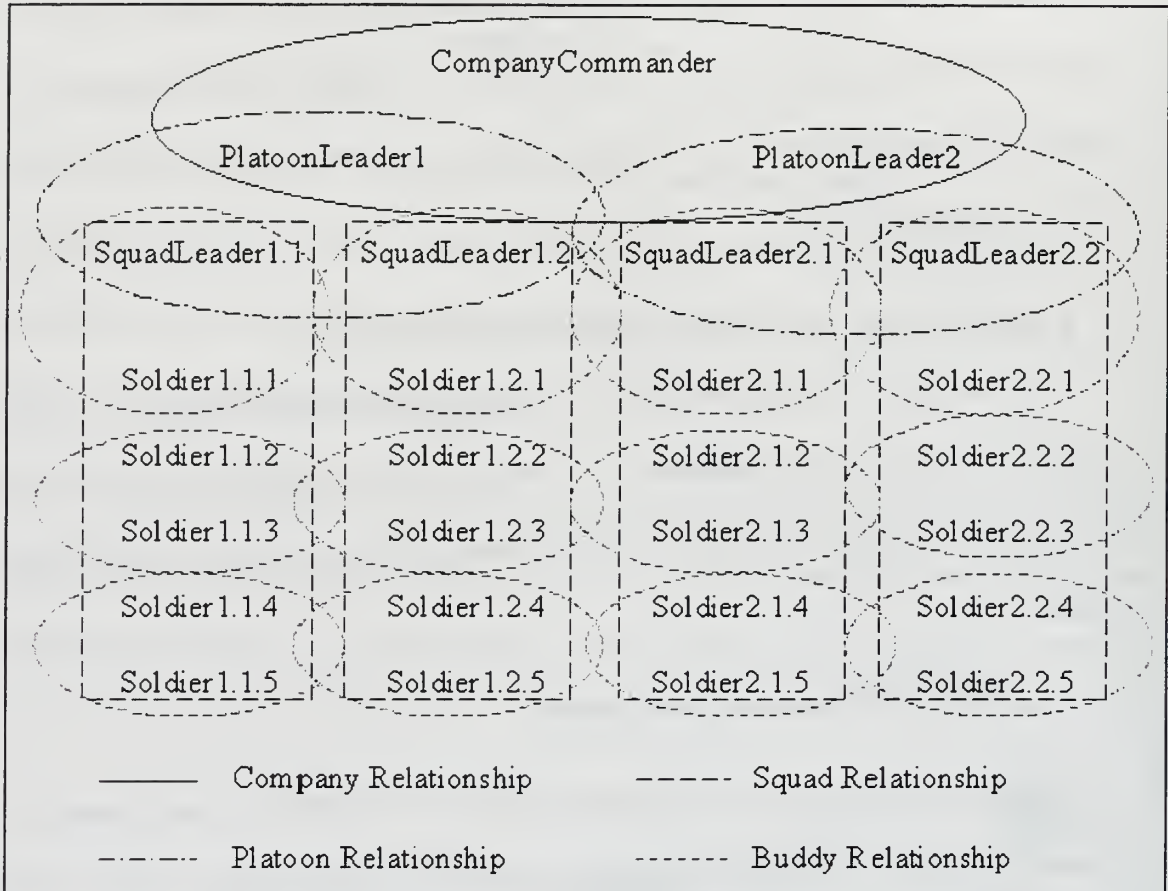


Figure 8. Example Relationship Hierarchy and Role Assignment

2. Identify Roles For Each Relationship

Each of the relationships identified above must have specific roles identified (Figure 8). Members of an army can be assigned the role of *soldier*. If desired, one could define a *general* or *global commander* of the army, but for this example, that is not

necessary. The company relationship should have a leader, or *company commander* and members. At this point, a decision regarding complexity must be made. Individual membership of the company relationship could be assigned to all soldiers under the ultimate leadership of the company commander, or company members may be restricted to leaders of subordinate units, such as platoon leaders and squad leaders. For this example, the *platoon leader* will be the company member reporting directly to the company commander, and the *squad leader* will be the platoon member reporting directly to the platoon leader. Finally, individual army *soldiers* will be the members of the squad relationship. Within the squad, two members may form the buddy relationship, with possible roles of *experienced* and *novice*.

3. Determine Goal/Rule/Action Types

Once these relationships and roles have been determined, the developer must consider the goals associated with each role. Goals are what motivates a *RELATE* agent to take action. Goals should be divided into types, corresponding with the type of action that will be required to accomplish each goal. In section D.5 below, rules will have to be identified that will allow an agent to select an action to accomplish each goal. In effect, the developer doesn't just define the goals first, and then the rules. Instead, the developer should be thinking of goal/rule pairs, the combination of which selects and action. A diagram detailing how goal/rule pairs fit into the action-decision loop is shown in Figure 9, at the top of the next page.

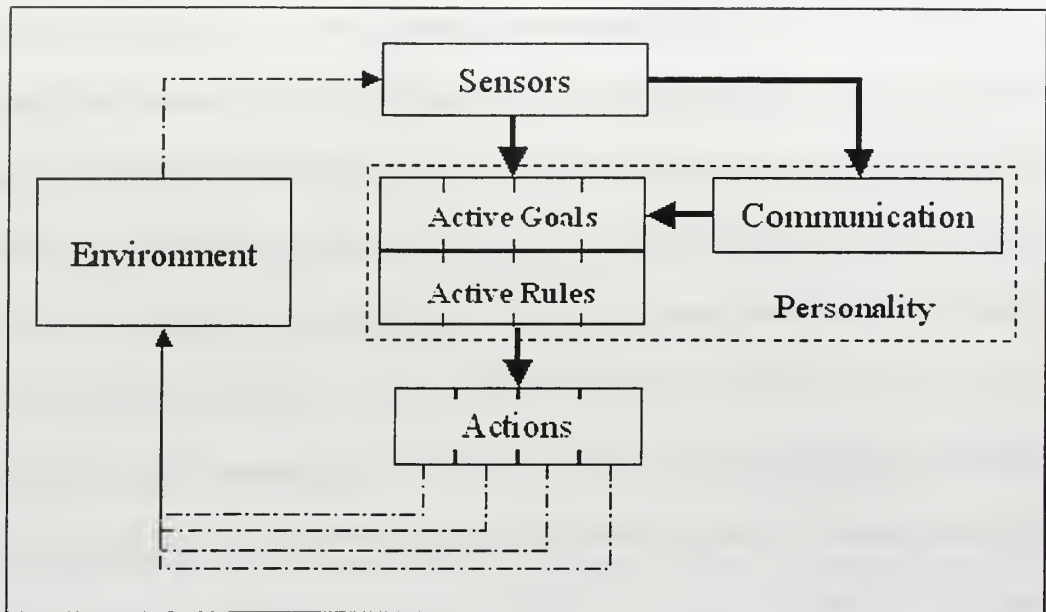


Figure 9. Action-Decision Loop

4. Determine Goals For Each Role

Keeping the concept of goal/rule pair in mind, the developer must identify all goals associated with each role in the simulation. An individual soldier's goals may include *maximizing enemy casualties* and *minimize personal injury*. If the simulation was more concerned about logistics than combat, other goals might include sustenance goals such as *remain hydrated* and *eat twice a day*. A leader's goals may differ, depending on his level in the chain of command. For Squad Leaders, they might include unit goals such as *scout* or *reposition artillery*, and *maintain unit cohesion*. A leaders goals may also include more individualize goals such as *following orders* and *keep leader informed*. Platoon Leaders may have more global objectives such as *protect right or left flank of company* and *minimize unit attrition*. The Company Commander exercises even more

global goals to achieve the overall objective. Examples may include *hold the hill*, *open the line*, or *capture the flag*.

5. Determine Rules For Each Goal

Rules are usually formulated during the goal-development process. This is because each goal must be specific enough to allow a rule, or set of rules, to achieve the goal algorithmically. A rule can be as simple as a mathematical equation, or as complex as another sub-agent operating on the perceived environment passed to it. Because of the diversity of rules, and the obvious requirement that they be tailored to the specific simulation, the Rule interface is little more than a place holder for a generic calculate() method that receives a sensedEnvironment object and returns a Java Object. The developer must cast the returned Java Object to the appropriate instance, such as Action, Integer, Float, etc.

Example rules for the members of the sample simulation could include random movement rules, gradient-type rules, or go-to-here rules. These are all examples of potential movement-type rules. Shooting-type rules could include shoot at all sensed enemy, shoot at closest enemy, shoot at enemy only when they are within a certain range, and don't shoot at enemy of a certain type. Communication rules may include send all data every turn, send partial data every turn, send data only every fifth turn, etc.

During this development phase of the structure of the simulation, it is more important to focus on the basic idea of the rule, vice defining the actual pseudo code or algorithm. A descriptive name should be used in this recipe for now.

6. Determine Feedback Mechanism For Each Goal

After an agent executes its action-decision loop, the results of its action have some effect on itself and/or the environment. Before the next action is decided, the agent must have some method to evaluate the effectiveness of its actions on achieving its goals. An action taken specifically to achieve a certain goal may improve the attainment of that goal, but worsen the attainment of one or more other goals. The agent must evaluate not just the active goal, but all goals attainment level, or health. It is therefore necessary that the developer has defined a clear, concise measure of effectiveness for each goal.

An example feedback mechanism might be the difference between current and past proximity to the enemies flag. Another example may be a comparison of past and previous internal states such as energy level, health, hunger, injury, etc.

7. Determine Credit Assignment For Each Rule

If there is only one rule defined for each goal, there is no need for this step. If the developer is interested in building intelligent agents, that adapt to improve over time, then multiple rules for each goal should be provided. The performance of each rule should be monitored by updating the weight of each rule with a credit assignment mechanism. Going one step further, if multiple rules are designed that offer different ways to accomplish the same goal, they can be used in different combinations. Genetic Algorithms (GA) can then be used to explore various combinations of rule sets. See Figure 10, on the next page, for a graphical representation of multiple rule sets, credit assignment and active rule selection.

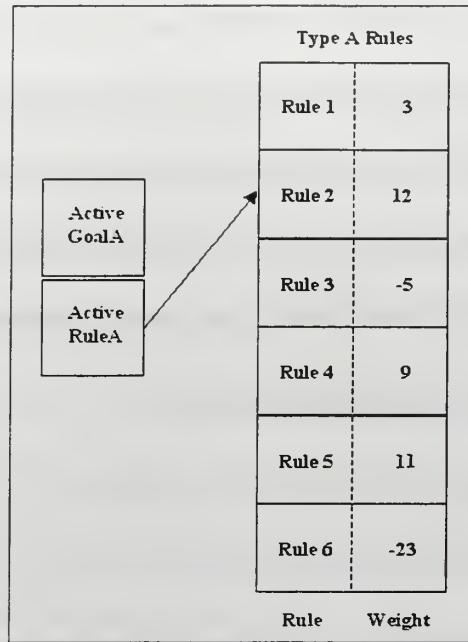


Figure 10. Multiple Rules per Goal

8. Implement Design By Satisfying *RELATE* Interfaces

At this point, the developer should have a concise, well-defined structure that can be used as a guideline to create the simulation. Starting from the Rules and Goals, and working up to the Roles and Relationships, the developer should construct specific Java classes for the simulation by implementing the appropriate interface.

9. Use Reference Cases As A Starting Point For GUI Development

The Graphical User Interface (GUI) for each simulation will need to be tailored to by the developer. Situated and non-situated environments, statistical output, and other factors greatly influence these decisions. The three reference cases described in the following chapters can be used as a starting point, or simply as a source for ideas.

E. RELATE JAVA CLASS AND INTERFACE DEFINITIONS

The following sub-sections provide a brief description of the classes and interfaces of the *RELATE* Java package. A more complete description is available in JavaDoc format provided in Appendix C. The actual source code itself is provided in Appendix E. Both JavaDocs and source code are also included in the attached CD-ROM.

1. Public Class RelationshipManager

RelationshipManager is the heart of the *RELATE* simulation package.

RelationshipManager is an example of the *singleton* programming pattern. “Singleton” means that there is one, and only one, instance of this class per application. This is accomplished by the static, synchronized *getRelationshipManager()* method. This method creates a new *RelationshipManager* upon the first request, or returns the existing, unique *RelationshipManager* if it has already been created. The *RelationshipManager* is the only complete Java class in *RELATE*. All other classes are either abstract or interfaces. An abstract class contains one or more abstract methods that are required to be defined by the developer. An interface has no data members and the developer must define all methods.

The *RelationshipManager* handles the formation and administration of all relationships by requiring agents to form relationships with the *checkForRelationships()* method. This method is the most significant method in this class. It checks for every possible relationship that can be formed between the requesting, passed in agent and other agents in its *sensedEnvironment*. Since the requirements for formation of new relationships are defined within the individual relationships, the *RelationshipManager*

instantiates the requested relationship using the *createRelationship()* method. This is possible due to the no-argument constructor used in the Relationship interface. If conditions are met, it adds the agent to this new Relationship. Otherwise, the relationship is never utilized and it is cleaned up with the automatic garbage collection feature of Java. Relationship administration is accomplished by maintaining a current list, or vector, of active relationships, available through a getter method. The *RelationshipManager* adds relationships to this vector, but they are removed by the individual relationships. See the Relationship class below for more details on how relationships interact with the *RelationshipManager*.

2. Public Abstract Class Thing Extends Object

This abstract class implements the *RELATE* Thing. This is the minimal entity that can exist in a *RELATE* simulation. This class defines the minimum requirements for a Thing in the *RELATE* architecture. A Thing has a unique entity identification number and name with associated getter and setter methods. If the entity identification number and name are not provided in the constructor, a no-argument constructor will assign the number “0” and name “unnamed” to the Thing. The only other methods that a Thing has are the *step()* and *drawSelf()* abstract methods. The *step()* method is used to update the object and the *drawSelf()* method is used to update the appearance during the simulation run. Both methods are unique to each simulation and therefore must be defined by the developer.

3. Public Abstract Class Agent Extends Thing

This abstract class implements the *RELATE* Agent. Since all Agents are objects, or things, Agent extends Thing, giving it the ability to update and draw itself. By the definition provided in Chapter II, an agent “...perceives its environment through sensors and acts upon that environment through effectors to achieve one or more goals.” To satisfy these requirements, the developer must first establish a method for the agent to gather information about the environment it exists in. This information is stored in the *sensedEnvironment* data member. This class allows the developer to do this by defining the abstract *getSensedEnvironment()* and *setSensedEnvironment()* methods to interact with the simulation environment directly. These methods should utilize the *sensorList* hashtable to define what each agent is capable of sensing, and only allow the agent to sense the specific attributes of the environment that its current sensor list can detect or discern.

Next, the agent must select an appropriate action. Two things are needed for this: actions that can be selected, and a method or methods to select these actions. The actions each agent can take are simulation-dependent and must be defined by the developer. For simpler simulations, actions can be built into the Agent class itself. An Action interface is provided as a tool for the developer designing more complex simulations. The mechanism for selecting actions is based upon the relationships each agent forms, and the associated roles and goals that the agent attempts to fulfill.

Agents have a relationship hashtable that the developer must fill with the class names of all the potential relationships that the agent can form. This can be accomplished by using the *addRelationshipName()* method.

4. Public Interface Relationship

Relationships are the life-blood of the *RELATE* architecture. One of the most important aspects of a Java class that implements the Relationship interface is that it must have a no-argument constructor. As described above, this allows the *RelationshipManager* to create it dynamically to verify prerequisites and assign members. Relationships have a *conditionsMaintained()* method that is used by the *RelationshipManager* to verify prerequisites are met prior to creating the relationship. Once created by the *RelationshipManager*, the Relationship objects are independent agents that issue roles to each member agent using the *issueRoles()* method. They also monitor conditions of, their members. They also destroy themselves if minimum requirements for existence are not maintained using the method *destroyRelationship()*. When this happens, the relationship withdraws all of its associated Role objects from each member, then removes itself from the *RelationshipManager*'s active relationship vector.

5. Public Interface Role

Defines the minimum requirements for a *RELATE* Role. A Role object brings additional capabilities and responsibilities to a host Agent. This can include, but is not

limited to, sensors, goals (with their associated rules), and actions that enable it to act upon itself and its environment.

6. Public Interface Goal

Defines the minimum requirements for a *RELATE* Goal. Provides a mechanism to select from a collection of methods (rules) that affect an Agent's internal state or external environment. These methods are selected based on the *activeRule* of a collection of Rules. The method, or action, taken is intended to satisfy the Goal. A feedback mechanism must be provided in the *assignCredit()* method to determine the health of the Goal as well as the success or failure of the current active Rule.

7. Public Interface Rule

Rule interface for use with the *RELATE* architecture. Requires the user to define the method "calculate" which receives a *sensedEnvironment* object and returns an object representing the result of the rules calculation. A *toString()* method is required to assist in the display and evaluation of the Rule.

8. Public Interface Personality

Defines the minimum requirements for a Personality in the *RELATE* architecture. This should be a simple data structure can be modified or added to for expandability. Personality is used to influence goal selection and measurement as well as credit assignment to rules. Typically they are mathematical factors that capture key aspects of the individual in the particular situation being modeled.

Example personality traits could include the following:

- Loyalty - An agent's propensity to reward a Rule when it was successful.
- Persistence - An agent's propensity to penalize a Rule when it fails.
- Tolerance - An agent's ability to reward varying degrees of accuracy.
- Obedience – An agent's propensity to take direction from another source.
- Independence – An agent's propensity to operate alone.

A simulation may be designed to use the same personality for all agents, or randomly issue personalities to stress variations. Personalities may remain fixed or be allowed to change based on experience or external pressures.

9. Public Interface SensedEnvironment

A *SensedEnvironment* is a complex data structure unique to the simulation and defined by the developer. It must contain appropriate data members to store all aspects of the perceived environment. This interface requires the developer to implement methods to get the *SensedEnvironment* object, as well as a vector of the sensed agents that is used by the *RelationshipManager*.

10. Public Interface Sensor

Interface for sensor objects to be used in *RELATE*. These objects are defined by the developer as a way of describing specific sensor capabilities for agents. Sensor objects are not currently implemented in any reference cases but are included in the *RELATE* package for future work and conceptualization.

11. Public Interface Action

Interface for action objects to be used in *RELATE*. The developer defines these objects as a way of describing specific actions agents may be capable of. Action objects are not currently implemented in any reference cases but are included in the *RELATE* package for future work and conceptualization.

F. SUMMARY

This chapter has described the *RELATE* design paradigm and associated Java development package. A simple analogy was provided as a training aide that described agents holding balloon strings trying to find balloons (relationships) to tie them on to. A recipe for using *RELATE* to develop MAS simulations that model human and organizational behavior was provided, as well as descriptions of each of the Java package classes and interfaces. The next three chapters detail reference cases that were developed and implemented using the *RELATE* MAS design paradigm and Java development package.

IV. AN INTRODUCTORY MAS SIMULATION

There are two reasons for perfect or deductive rationality to break down under complication. The obvious one is that beyond a certain complicatedness, our logical apparatus ceases to cope—our rationality is bounded. The other is that in interactive situations of complication, agents cannot rely upon the other agents they are dealing with to behave under perfect rationality, and so they are forced to guess their behavior. This lands them in a world of subjective beliefs, and subjective beliefs about subjective beliefs. Objective, well-defined, shared assumptions then cease to apply. In turn, rational, deductive reasoning...itself cannot apply. The problem becomes ill-defined.

– BRIAN ARTHUR

A. INTRODUCTION

In 1994, Brian Arthur, a leading economist and researcher at SFI, wrote a groundbreaking paper on inductive reasoning and bounded rationality (Arthur, 1994). In this paper he presented a seemingly simple problem, based upon the weekly patron attendance at a local nightclub in Santa Fe, New Mexico. The problem posed could not be solved using traditional deductive reasoning because of its complex nature. Deductive reasoning was defined as, “deriving a conclusion by perfect logical processes from well-defined premises.” The introductory quote above clearly states the reasons why deductive reasoning fails in this situation.

Arthur outlined a solution using multiple agents and argued convincingly that this was one of the only methods available that could solve such a problem. In turn, this

problem has become widely known as “The El Farol Problem”, and is solved here as an introductory MAS reference case using the *RELATE* package.

B. BRIAN ARTHUR’S EL FAROL BAR PROBLEM

The problem is based on the bar “El Farol”, in Santa Fe, New Mexico, which offers Irish music on Thursday nights. A number of people go to this bar to enjoy both the music as well as the company of other guests. If few people show up on a particular night, there is not enough social interaction to make the evening enjoyable. If, on the other hand, too many people show up, then the bar is over-crowded and patrons are unhappy. Each week, regular customers must decide whether they will go to the bar, believing that a good crowd will be present, or stay home, to avoid the unpleasant experience of an overly crowded bar. A major assumption in the problem is that they must make this decision independently, without communicating or collaborating with other patrons. Past performance and current state is also not considered during the decision-making process. The only information available to patrons is the attendance from previous weeks. To formalize the problem, the population of patrons will be fixed at N people total for the duration of the sampling period. If more than 60% of these people show up on a given night, the bar will be too crowded. Each patron is forced to make decisions based on what he or she *believes* other patrons will do. As Arthur states, this “lands them in a world of subjective beliefs, and subjective beliefs about subjective beliefs.” Because of the complex interactions and subjective beliefs, the actual week-to-week attendance is quite unpredictable. The average attendance, however, can be predicted using a MAS solution similar to that described below.

C. A *RELATE* RECIPE FOR THE EL FAROL BAR PROBLEM

1. Relationships

There is only one relationship necessary to solve the El Farol Bar Problem. The relationship consists of all the patrons that frequent the bar. One can think of this relationship as a social club, or drinking club. This relationship will be simply named the *ElFarol* relationship. If the problem didn't specifically prohibit individual members from communicating with each other, the simulation might allow patrons to form relationships with smaller groups. That form of cooperation and competition would certainly be an interesting extension of the original problem, but it is prohibited here.

2. Roles

The single role to be assigned from the *ElFarol* relationship is that of *barMember*. The only obligation associated with this role is that each week the *barMember* must decide if it will go to the bar or stay home. In this simple, introductory problem multiple roles are not used.

3. Goal/Rule/Action Types

Again, since there is only one goal in this simulation, there is only one goal type. As long as the rules and actions support this goal, this step in the *RELATE* recipe is completed. For clarification, however, one could classify the goal/rule/action type as that of a making a decision whether to attend or stay home. Therefore, the goal/rule/action type needed is an *attendance decision* type.

4. Goals

The basic goal of every agent in this simulation is to be *happy*. To accomplish this goal, an agent must do one of two things:

- Go to the bar and experience a good crowd (attendance ≤ 60).
- Stay home, and later find out the bar was too full and no one had a good time.

If the agent attended the bar and discovered it was over-crowded, or stayed home only to learn later that there was a good crowd that night, the agent would not be happy. The problem statement stipulates that choices are unaffected by previous visits, so it doesn't matter to the agent whether it was happy or sad the week before.

5. Rules

The rules used to achieve this goal must take as input a sensed environment consisting of the historical attendance of the bar, and return as output a predicted attendance value for the current week. The rules can simply be any kind of mathematical manipulation of last weeks attendance, or some combination of the past weeks attendance, that make a prediction about the current weeks attendance.

6. Feedback Mechanism

Since only one goal is defined in El Farol, there was no need for a feedback mechanism for goal selection. For a slightly more complicated problem, one could also add a goal of social happiness, which might take into account past performance, or attendance, to ensure that the individual agent actually goes to the bar every now and then, just to keep from being lonely.

7. Credit Assignment

If a person tries a certain product, and is satisfied with the results, they may be more inclined to purchase that same product the next time they have to decide among competing brands. The building strength of this conviction is often determined by the number of times the product satisfies the customer. At some point, there is little doubt that the same product will be chosen again. This is known as developing brand *loyalty*. If, on the other hand, a product fails to satisfy, the number of times that a customer will continue to purchase the same product before abandoning it is called *persistence*. This is the same as holding onto a losing stock until finally, it reaches a low threshold and is sold. Often it is not sufficient for a product or solution to be good, it must also fully satisfy the needs of the user. The willingness for an individual to put up with an inaccurate solution or prediction is called *tolerance*. Thus, the personality traits used for the agents in El Farol are:

- Loyalty (**L**) – credit assigned for being correct.
- Persistence (**P**) – (negative) credit assigned for being wrong.
- Tolerance (**T**) – fixed number for prediction accuracy calculation.

These personality traits were selected to assign credit to rules for being correct, penalize them for being incorrect, and reward them for being accurate. A rule is successful if it predicts an attendance on the same side of the “good crowd” number that the actual attendance is on. For example, if the total number of *barMembers* was 100, then 60 would be the maximum number of clients that the bar could hold before it became too crowded. If a certain rule predicted there would be 78 agents attending, the

action selected to satisfy the goal would be to stay home. If the actual attendance were 62, then the rule would have been successful. Staying home was the right decision. On the other hand, if the rule had predicted 59 agents would attend and the actual attendance was 62, then the rule would have failed. One can see, however, that although in the first case the rule correctly predicted that the bar would be overcrowded, it was not very accurate in its prediction. The rule in the second case, although incorrect in its prediction of overcrowding, more accurately predicted actual attendance. As this method of credit assignment is repeated week after week, individual rules begin to earn a reputation of being reliable predictors (Figure 11).

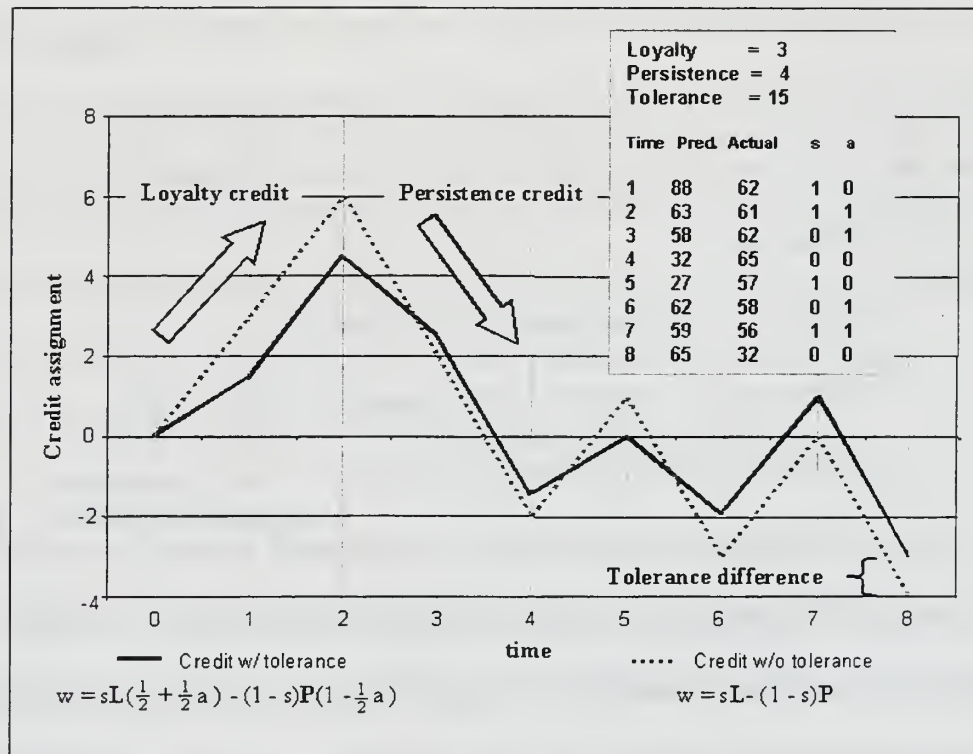


Figure 11. El Farol Personality

Credit (w) is assigned to each rule based on the individual agent's personality traits (L , P , and T), the success ($s = 1$) or failure ($s = 0$) of the rule at predicting bar attendance, as well as the accuracy of its prediction ($a = 1$, if prediction is within tolerance; $a = 0$, if prediction is outside of tolerance). As a rule gains more credit for being correct, its reputation rises. Each agent is issued a small collection of rules, but only uses the rule with the highest credit assignment, or weight. If a rule continues to perform poorly, it will receive more and more negative credit, and eventually reach a threshold that the agent chooses to turn it in for a new rule. This method of trading in rules allows the agent to explore a large number of rules in the total rule set and find the ones that are best suited for its personality.

D. A *RELATE* SOLUTION

The design structure developed using the *RELATE* recipe above is outlined and illustrated in Appendix C. This reference case is very simple in that there is only one relationship, role, and goal. It is very complex, however, in the rule base and credit assignment affecting rule selection.

1. El Farol Rules

32 different rules were created and distributed randomly in groups of seven to each agent. These rules are all mathematical or logical rules. They are written as methods that receive an attendance history vector and return a predicted attendance integer value. The mathematical representation of the rule is actual found in the *calculate()* method in the Rule object. The authors acknowledge that implementing

several Rule objects, with one attendance prediction algorithm method each, is actually more work than is required to solve the problem. Because this problem is so basic and does not utilize the main feature of the *RELATE* architecture, namely multiple relationships, it would be easier to simply write a big switch statement inside the Goal itself, that includes all of the rules. Figure 12 provides an example rule definition. See the *RELATE* Release Notes provided in Appendix B for access to all rules and El Farol source code.

```
public class Rule3 implements Rule
{
    Vector calcVec;
    int total;
    int prediction;
    ElFarolSensedEnvironment sensedEnv;

    /*****
    * no argument constructor
    *****/
    public Rule3()
    {
        total = 0;
    }

    /*****
    * Calculates the average of last four weeks.
    * @return an Object (Integer) depicting predicted attendance.
    *****/
    public Object calculate( SensedEnvironment pSE )
    {
        sensedEnv = (ElFarolSensedEnvironment)pSE;
        calcVec = sensedEnv.getSensedAttendance();

        for( int count = 0; count < 4; count++)
        {
            total += ((Integer) (calcVec.elementAt(count))).intValue();
        } // end for

        prediction = total / 4;

        //clean up
        total = 0;

        return new Integer(prediction);
    }
} // end Rule3 class
```

Figure 12. Sample El Farol Rule Algorithms

2. Graphical Output

When implementing a *RELATE* MAS solution such as this, the developer must design an appropriate user interface. Since there is little to be gained by watching *barMembers* actually moving to and fro, and more to be gained by evaluating the statistical data generated from the model, a graph output was created. This output displays the actual weekly attendance and the running average. The screen shot shows a run based on 100 agents using 7 rules per agent. The run covers a period of 100 weeks (Figure 13).

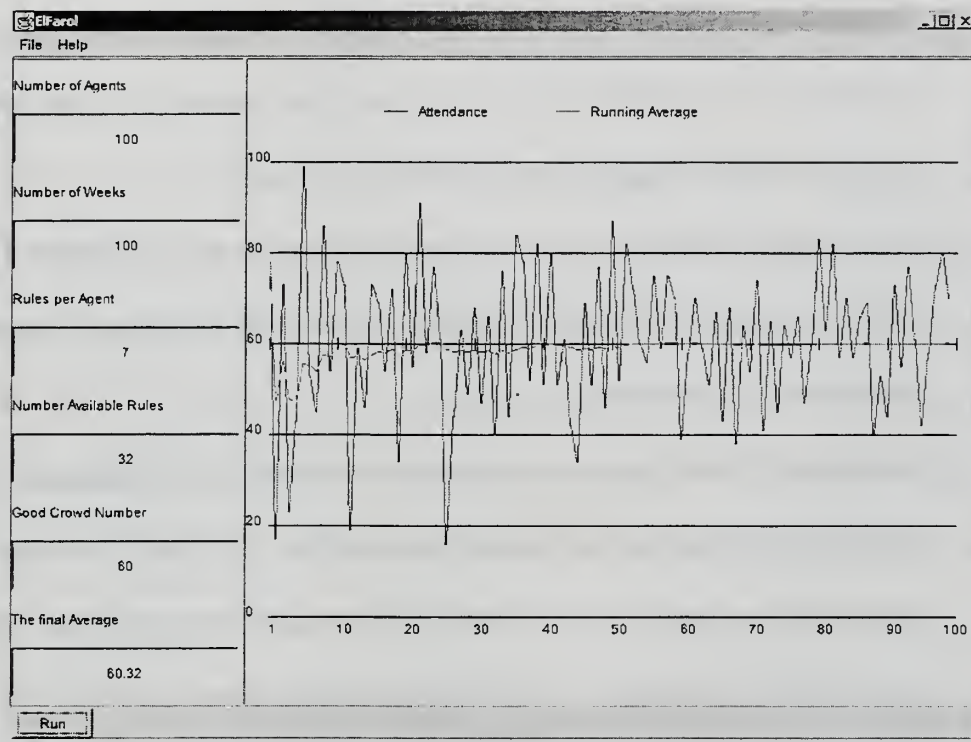


Figure 13. El Farol Attendance Output

Although the weekly attendance (jagged line in blue) varies chaotically, this was expected due to the dynamic nature of the problem. Over time, however, the average

attendance (smoother line in magenta) quickly stabilized to the “good crowd” number of 60. Note that the final average attendance number is 60.32 and stabilized near 60 around week 10.

E. SUMMARY

The *RELATE* architecture supports the mechanisms needed to solve the El Farol Bar Problem. This problem demonstrated dynamic active rule selection from a large pool of potential rules.

Future work on *RELATE* El Farol could include:

- Improving the GUI interface to allow the user to adjust various parameters prior to the run including number of agents, total number of weeks, number of rules per agent, and good crowd number.
- With slight modifications, the successful rule sets could be combined over time using Genetic Algorithms (GA) to cause the fluctuations of actual attendance to be damped.
- Add an additional goal involving social satisfaction. As it stands, an individual *barMember* may seldom go to the bar, but overall, because of the bar attendance, may be happy. With the additional social goal, the member may become unsatisfied with continued poor experiences.
- Converting the program to run as an Applet so that it would be available for web-based viewing.

V. ADDING AGENTS TO A NETWORKED DIS-JAVA-VRML SIMULATION

The IEEE Distributed Interactive Simulation (DIS) Protocol is used to communicate state information (such as position, orientation, velocities and accelerations) among multiple entities participating in a shared network environment. Java is a portable networked programming language that can interoperate on any computer that includes a Web browser. The Virtual Reality Modeling Language (VRML) enables platform-independent interactive three-dimensional (3-D) graphics across the Internet, and can be used to compose sophisticated 3-D virtual environments.

– DON BRUTZMAN

A. INTRODUCTION

The Naval Postgraduate School's (NPS) Modeling, Virtual Environments and Simulation (MOVES) Academic Group is an "interdisciplinary department dedicated to education and research in all areas of modeling, virtual environments and simulation" (MOVES, 2000). A particular focus of study involves research and development of enabling methods for Large Scale Virtual Environments (LSVE). Capture the Flag (CTF), a distributed DIS-Java-VRML simulation, has been developed and improved since 1996 by students and faculty associated with the MOVES Academic Group. In particular, Professor Don Brutzman has used CTF as a test-bed for LSVE's associated with the DIS-Java-VRML Working Group and the Physically Based Modeling course he teaches at NPS (DIS-Java-VRML, 2000). An excellent test of the *RELATE* architecture was to add relation-centric agents into this existing distributed simulation.

B. CAPTURE THE FLAG

CTF is a 3-D virtual battle space modeled after a 2500-square kilometer portion of the Ft. Irwin terrain, located in the southern California desert. Users of this distributed simulation control tanks or helicopters to attempt to capture their opponent's flag and return it to the users home base. Both tanks and helicopters have the ability to maneuver and shoot weapons. Vehicle manipulation is accomplished via separate control panels for each entity. Typically, a group of players will start up CTF applications on a number of computers connected on the same Local Area Network (LAN), and each will control a single entity. If they conduct team play, they must agree on the teams before hand and coordinate actions verbally, assuming they are in the same room or space. When demonstrating this software, a number of computers must be reserved and volunteers must be found to operate each entity. *RELATE* agents offer a solution to this problem. By incorporating self-organizing agents into CTF, players are able to start up a squad of agents and play against them without any other human players. Alternatively, a single player can play a team of agents against another player's team, simply by choosing the composition of the team. The addition of agent-driven capability also makes it much easier for a single person to demonstrate CTF to visitors. One person can start up one or two computers and set two squads of agents against each other without the additional assistance manning the controls for the entities.

C. A *RELATE* RECIPE FOR CTF AGENT

1. Relationships

The only relationships desired for CTF Agent is a squad relationship. The intent of these relationships is to balance out offense and defense for each collection of agent-driven entities. Since the simulation is limited to blue and red forces, and only three tanks and three helicopters on each side, the two instances of the squad relationship are *BlueSquad* and *RedSquad*. See “Future Work” in section 5, below, for suggestions of additional relationships.

2. Roles

The roles assigned in CTFAgent are *squadLeader* and *squadMember*. Although the *RELATE* relationship automatically reassigns the role of leader if the leader is destroyed, this does not actually happen in CTF Agent. The reason is that, although the leader may appear to be destroyed, the original CTF never actually truly destroys an entity. Instead, a destroyed entity is simply repositioned back to its starting point and allowed to continue.

3. Goal/Rule/Action Types

Action types are limited to vehicle movement. Additional types were not implemented due to the simplicity of the simulation and time requirements for delivery. Tank agents shoot at enemy units that enter their sensor range, but this is not handled through goals and rules. All tanks automatically engage any detected enemy, closest one first. Shooting capability was not implemented for helicopter agents.

4. Goals

Three primary goals are used in the simulation: *offense*, *defense*, and *coordinate*. The offense goal is to drive to the enemy flag, obtain it, and return to base. The defense goal is to stay close to your own flag and protect it from all enemy vehicles. The coordinate goal is used by the *squadLeader* to balance its squad forces in an attempt to win the game.

5. Rules For Each Goal

Only one rule per goal was used in the simulation to facilitate rapid development and to assist in working within the simulations current network design. The rules correspond directly with the goals and are named *offense*, *defense*, and *coordinate*.

6. Goal Feedback Mechanisms

Goal feedback is accomplished through the leaders direction. There is no direct link to the agents' goal via a perceived environment.

7. Rule Credit Assignment

No credit assignment is required since only one rule per goal is used.

D. A *RELATE* SOLUTION

The design structure developed using the *RELATE* recipe above is outlined and illustrated in Appendix D. Three existing CTF classes were modified and four new java classes were added, in addition to the various required *RELATE* classes, to provide autonomous, agent-driven entities. These classes provide the user the ability to select

“Agent Driven” for any unit available on the red or blue start panel. The modified classes are the *Referee*, the *BlueStartPanel* and the *RedStartPanel*. New classes added include the *AgentTankActionInterpreter*, *AgentHeloActionInterpreter*, *AgentTankControlPanel*, and *AgentHeloControlPanel*.

The following paragraphs describe the new features available to CTF, as well as the tactics employed by the *RELATE* agents if selected.

1. Red and Blue Start Panels

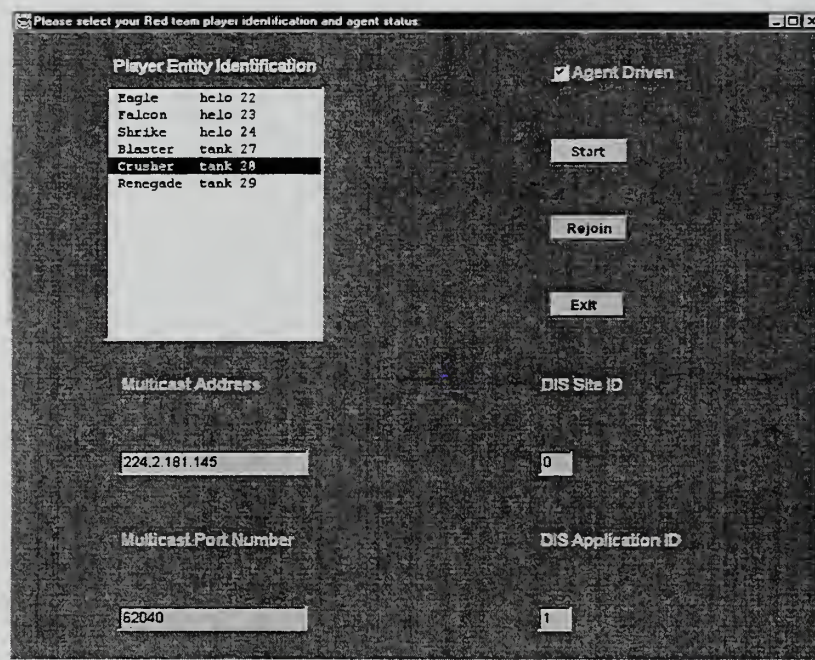


Figure 14. Red Start Panel with Agent Driven Selected

The only changes to the start panel classes was to modify the panel to include an “Agent Driven” selection block in the upper right-hand corner, and to launch the appropriate agent control panel if it is selected (Figure 14). Any tank or helicopter entity

normally available to a user can be agent driven simply by selecting this feature, then starting as usual.

2. Tank Agent

The tank agent uses the same basic control panel as a manually controlled tank, except that all of the buttons are disabled. The only button that is not disabled is the “new vehicle” button. The agent updates the heading, speed, and turn rate indicating blocks, as well as the sliders, as it maneuvers through the battlefield. Main and Aux Gun Ammo values do not change and never run out. Although the user can physically manipulate the maneuvering controls (slider bars), the agent will correct these user-entered speed changes, causing a slight cycling effect. As soon as the user releases the speed control, the agent’s commands will take full effect.

a. Engaging Enemy Units

All tank agents develop a target list based on a fixed sensor range of 1750m (approximately the same range that a user would be able to visually detect another entity). Once an entity is detected, a determination is made as to friend or foe. If the newly detected entity is an enemy unit, it is added to the target list. The tank will always attempt to engage the closest enemy unit in its target list. The engagement behavior only controls the aiming and shooting of the main gun. Direction of the vehicle motion is controlled by the role of offense or defense, as directed by the squad leader.

b. Offensive Tank Agent



Figure 15. Offensive Tank Agent

Figure 15 shows an offensive tank approaching the enemy base and flag. It is shooting at another tank that is engaging a different, off-screen opponent. An offensive tank will always attempt to capture the enemy flag, wherever it is located, engaging the closest enemy unit within sensor range. It does this by driving maximum speed (60 kts) to close the distance to the flag until it has captured it. Its turning radius is set based on 40% of the desired heading change. Once it captures the enemy flag, it returns at maximum speed, in a straight line to it's own base.

c. Defensive Tank Agent



Figure 16. Defensive Tank Agent

Figure 16 shows a defensive tank as it drives by it's own flag, engaging an unseen helicopter (evident by the super-elevated turret). A defensive tank will always drive at maximum speed around its own flag, engaging the closest enemy unit within sensor range. If an enemy unit successfully captures the flag, the defensive tank will follow the flag, and the enemy unit carrying it, all while continuing to engage the closest enemy unit with gunfire. If the defensive tank chases the successful enemy unit all the way back to the enemy base, then destroys the opponent just prior to the flag being won and point given, the defensive tank will not shift to capture the enemy flag. Instead, it

will continue to drive in circles around its own flag, attempting to protect it from any approaching opponents. The original Capture the Flag code does not allow a unit to carry its flag back to its own base, so the agent cannot do this either.

3. Helicopter Agent

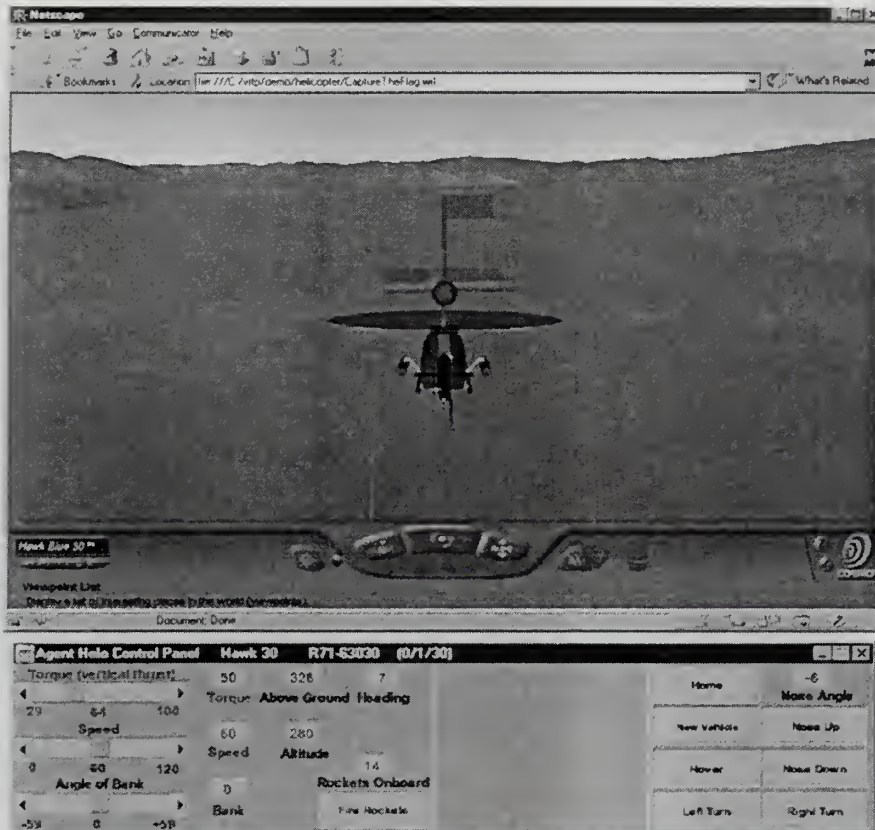


Figure 17. Helicopter Agent

Figure 17 shows a helicopter carrying the enemy flag back to its own base. The helicopter agent is, by default, an offensive agent, but it is not capable of shooting. Aiming the helicopter gun is accomplished by controlling the attitude and direction of the helicopter. This was not implemented in the helicopter agent due to time constraints and the complicated nature of CTF helicopter weapon aiming technique. The helicopter

agent's only goal, therefore, is to capture the enemy flag and return it to home base. When the helicopter gets within 1000m of the enemy flag, it begins a decent algorithm and flies to within 100m of the flag to capture it. It executes the same decent profile upon return to home base with the captured enemy flag.

4. Squad Relationship

Squad relationships are formed with the *RELATE* agents in a central server type class called "Referee," which is part of the original CTF. The Referee class was modified to support the CTF Agent implementation in place of the *RelationshipManager*. Squad relationships are formed when two or more, same color, agent-driven vehicles are started. The implementations of the squads are as described above, but the ramifications are the automatic assignment of offense or defense to each agent-driven entity created. If only one agent is created on a particular side (blue or red), no relationship can exist and the agent is automatically offensive. The rational being that if there is just one agent, the game can only be won if that agent pursues the enemy's flag. No points are earned for defending your own flag, no matter how successful the defender. Once another same-color, agent-driven entity is started up, a squad relationship is formed and a leader is assigned. The leader determines the make-up of the squad and ensures a defender always protects the team's flag. If the squad is made up of two tanks, one will be assigned offense, the other defense. If, on the other hand, the squad is made up of a tank and a helicopter, the tank will always be assigned defense. This is because the helicopter is unable to defend the flag, since it has no weapons capabilities, but it can capture the

enemy flag. If more than two agent-driven entities are started on the same team, one tank will be assigned defense and all others will be assigned offense. Figure 18 shows a self-organized squad of three red tanks fighting against a self-organized squad of two blue tanks and a helicopter. Note the tactic identifiers circled on the three red agent control panels that show two offensive and one defensive tank control panels.

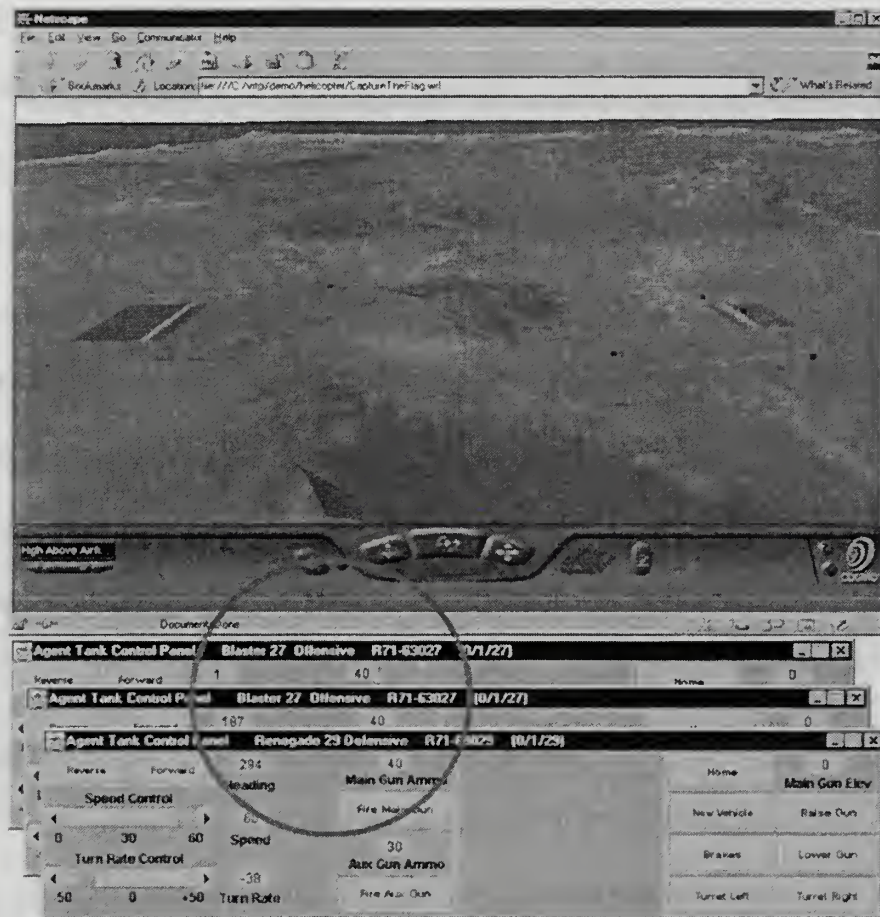


Figure 18. Capture The Flag Agent Squad

5. Future CTF Agent Work

The following additional improvements are suggested for future work that would improve performance of CTF Agents:

- Currently, an agent's targeting and shooting is not restricted by line-of-sight. A target is acquired once it is within sensing range, regardless of whether it can actually be "seen" from the point of view of the agent. An *isVisible()* method is needed that checks to make sure entities are visible prior to adding them to the *targetList*. This will probably be accomplished by simply utilizing the existing line-of-sight algorithm. Terrain collision detection needs to be integrated so that rounds cannot be fired through mountains.
- The agent helicopter needs the ability to shoot. This would be a good class project for a student studying DIS-Java-VRML and agent based simulations. Once the agent helicopter can shoot, then it should be modified similar to the tanks so that it can be assigned roles of offensive and defensive and be a full member of the team.
- Agents should have the ability to alter their motion to avoid or close enemy units, vice always just driving to the enemy or friendly flag.
- Agents should have the ability to dynamically shift their goal priorities to take advantage of the changing environment, such as when a defensive agent finds itself near the enemy flag after its team loses a point (which returns both flags to their respective home bases).

- Allow the formation of helicopter/tank team relationships, where the helicopter defends the tank as they both go to capture the enemy flag.
- Improve the goal feedback mechanism to include a measurement of the state of each flag.

E. SUMMARY

By adding relation-centric agents to an existing distributed simulation, CTF Agent provided an invaluable reference case. There was no attempt to drastically alter the networking or run time execution of the code. This led to interesting problems that would have not had to be dealt with if a similar simulation were to be designed from scratch instead. Even with these limitations, however, the *RELATE* architecture greatly simplified assembling the required components. The result of adding these agent has led to new and fascinating areas of study and design that were previously unavailable.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. A SITUATED LAND-COMBAT MODEL

Most traditional models focus on looking for equilibrium ‘solutions’ among some set of (pre-defined) aggregate variables...ISAAC focuses on understanding the kinds of emergent patterns that might arise while the overall system is out of, or far from, equilibrium. - ANDY ILACHINSKI

A. INTRODUCTION

The study of land combat has been the primary focus of military simulations for decades. Recently, there has been a concentrated effort at augmenting these models with agent models that simulate human decision-making and organization. Andy Ilachinski’s ISAAC is a first step in this arena (ISAAC, 2000). To demonstrate the capability of the *RELATE* design paradigm and architecture, a simulation similar to ISAAC was developed in about three weeks. Since it is only an example of the potential capabilities of *RELATE*, it does not have all the functionality of ISAAC in many respects, yet has more accurate modeling in respect to organizational hierarchy and dynamic goal selection. Because of this, JACOB only resembles the original ISAAC, and thus the modifier, “Son of ISAAC.” Appendix A provides additional information on ISAAC and the follow-on simulation environment, EINSTEIN.

B. JACOB (SON OF ISAAC)

JACOB was designed as a test-bed to demonstrate how the *RELATE* architecture can be used to create a simulation similar to Ilachinski’s ISAAC. Like ISAAC, JACOB takes place in a 2-D battle-space with agents graphically represented by blue and red

dots. This is where the similarity ends. The *RELATE* implementation focuses on a dynamic relationship structure that allows qualified agents to join or form relationships as their sensed environment changes. These agents then base their movement, communication and shooting actions on their assigned roles and goals.

C. A *RELATE* RECIPE FOR JACOB

1. Relationships

The relationships found in JACOB are *BlueArmy*, *RedArmy*, *Squad*, and *Company*. Although not completely representing an accurate company formation, since it is missing the platoon level, JACOB provides a close approximation to the organization often seen with unit strengths of around 100 soldiers. The army relationships are needed to be able to distinguish opposing sides, similar to uniforms or badges.

2. Roles

The roles in JACOB correlate directly to realistic roles one would find in these typical organizational relationships. For simplification purposes, only a few of the actually occurring roles are included. The roles are *Soldier*, *SquadMember*, *SquadLeader*, *CompanyMember*, and *CompanyCommander*.

3. Goal/Rule/Action Types

The action types developed for the JACOB simulation were limited to: *movement*, *communication*, and *shooting*. This allows competing and non-conflicting goals to exist within the agents. It also provides a straightforward example of dynamic goal selection.

4. Goals

JACOB provides numerous goals, of all allowed types, within each assigned role. These goals range from “maintaining unit cohesion,” in the Squad Member role, to “keep Company Commander informed,” in the Company Member role. A complete list of goals is given in Appendix E.

5. Rules For Each Goal

Only one rule is provided for each goal. This was deemed sufficient to demonstrate the hierarchical nature of the *RELATE* package. Additional rules can be included similar to the El Farol simulation discussed in Chapter IV. Movement rules return a new location for the agents, communication rules return a “JACOB Agent Report” object filled with specific information about the agent or squad, and shooting rules actually decrement the health of enemy agents.

6. Goal Feedback Mechanisms

Keeping with the *RELATE* design, all goals have a feedback mechanism device that allows them to develop a current weight based on the agents personality, the changing environment, and the direction of it’s superiors. All goals use a standard formula for determining this weight that incorporate the following terms:

- Goal attainment (\hat{g}_i) - The measurement of how close an agent’s current situation is to fulfilling the i^{th} goal (g_i).
- Goal personality factor (p_i) – The personality trait that influences g_i .

- Agent goal weight (Aw_{gi}) – The weight that g_i has that affects the agent's next decision. Aw_{gi} is determined from the agent's current perceived goal attainment value and the agent's goal personality factor (i.e. how much the *agent* cares about that particular goal being satisfied):

$$Aw_{gi} = A\hat{g}_i * Ap_i$$

Equation 1. Agent Goal Weight

- Leader's goal weight (Lw_{gi}) – The weight that the g_i has that affects the leader's next decision. It is determined from the leader's current perceived goal attainment value and the leader's goal personality factor for that particular goal (i.e. how much the *leader* cares about the goal being satisfied):

$$Lw_{gi} = L\hat{g}_i * Lp_i$$

Equation 2. Leader Goal Weight

- Agent's obedience factor (θ) – The extent to which an agent obeys its leader.
- Final Goal Weight (W_{gi}) – The final weight for the i^{th} goal based upon the agent's goal weight and a portion of the leader's goal weight as modified by the agents obedience factor:

$$W_{gi} = Aw_{gi} + (\theta * (Lw_{gi} - Aw_{gi}))$$

Equation 3. Final Goal Weight

In basic terms, the agent has a will of it's own for achieving various goals. The leader attempts to exert influence on the subordinate, but this influence is tempered by

how well the subordinate follows orders. This includes positive as well as negative influence. It is the difference in goal weights, or concern for the goal, that is added or subtracted from the agent's own goal weight.

7. Rule Credit Assignment

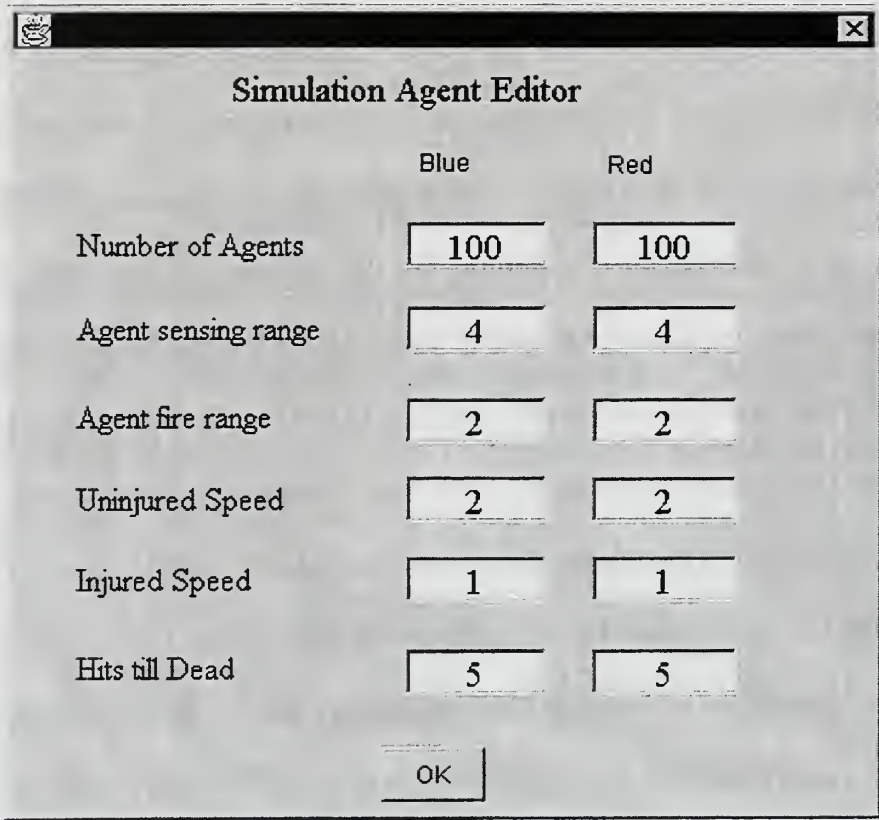
Since there is only one rule per goal, rule credit assignment is not currently utilized in this simulation.

D. A *RELATE* SOLUTION

The design structure developed using the *RELATE* recipe above is outlined and illustrated in Appendix E. It is clear after looking at the schematic representation of JACOB, especially compared to the previous two reference cases, that JACOB is much more complex. Not only does it have agents that are members of more than one relationship, but also there is a highly structured chain of command. JACOB consists of over 50 classes and hundreds of methods. The task of putting this simulation together and managing the interactions was greatly reduced by using the *RELATE* design recipe. The reader is reminded that the overall goal of JACOB was to exercise the dynamic goal selection and self-organizing behavior often seen in company strength groups. Much more work is needed to incorporate functionality similar to ISAAC, making this a useful model of land combat. The following sections describe various features of JACOB, as well as their limitations at this stage of development. The working code for this simulation, as well as all the *RELATE* source code, is available on-line and in the attached CD-ROM, as described in Appendix B.

1. Simulation Agent Editor

JACOB was built with a Simulation Agent Editor that was designed to allow the user access to selected red or blue variables (Figure 19). The simulation is preset with specific values that are tuned for the current goals and rules. Until further work is accomplished tuning the simulation, changes to these values will produce predictable, but undesirable results. In particular, changing the agent's sensing range causes the agents to veer off to the left or right, instead of engaging the enemy or moving in the direction of the enemy's flag. Changes in agent speed also have no affect on the actual speed of the agent across the virtual battlefield.



	Blue	Red
Number of Agents	100	100
Agent sensing range	4	4
Agent fire range	2	2
Uninjured Speed	2	2
Injured Speed	1	1
Hits till Dead	5	5

OK

Figure 19. Simulation Agent Editor for JACOB

2. Loading and Saving Environments

After accepting the agent variable as listed in the Simulation Agent Editor, the JACOB window opens up, as well as a blank Agent's Statistics window. The user continues by selecting the start button and then has an option of using an open environment, loading a stored environment, or creating a new environment.

a. Using An Open Environment

If the user wants to use an open environment with no obstacles, mountains or tree cover, then all that is needed to continue is to press the “cow” button (see Section 3 below).

b. Loading A Stored Environment

To load a previously saved stored environment, or the sample environment included with the simulation, the user should use the File pull-down menu at the upper left of the simulation window, and select *Load Environment* (Figure 20).

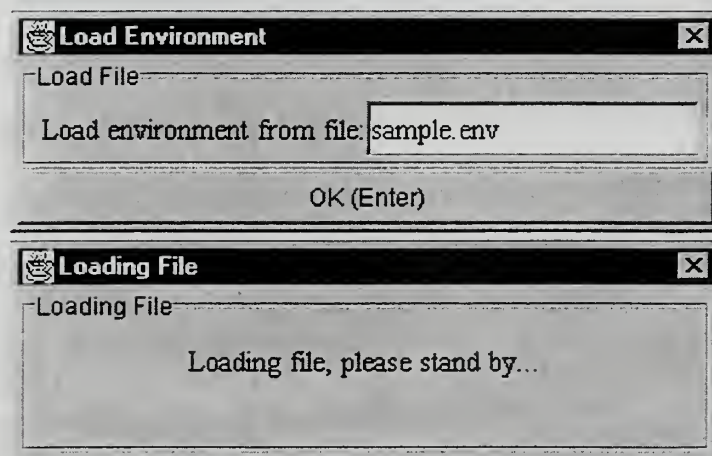
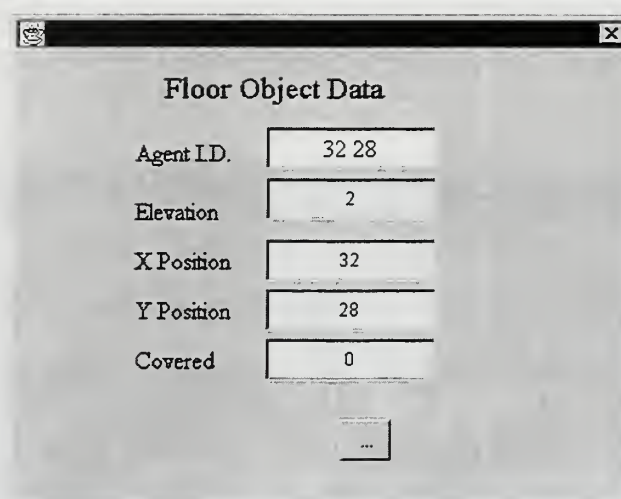


Figure 20. Loading a JACOB Sample Environment

c. *Creating A New Environment*

To create a new environment, JACOB allows the user to select any position on the simulated battlefield, called a floor object, and designate specific attributes (Figure 21). A Floor Object Data editor will appear and allow the user to specify the elevation (0, 1, or 4), and whether or not it is covered (1) or not (0). If a floor object's elevation is set to 4, it is an obstacle and no agent can cross over it or reach the top of it.



Floor Object Data	
Agent I.D.	32 28
Elevation	2
X Position	32
Y Position	28
Covered	0

Figure 21. Adding Floor Objects in JACOB

Once an environment is created, it can be saved for reuse, or the simulation can be started without saving it, by pressing the “cow” button. After the simulation run, this environment will not be available again without recreating it from scratch. To save an environment, *Save Environment* is selected from the *File* pull-down menu. After the new environment is saved, the simulation can be started by pressing the “cow” button.

3. Starting The Simulation

To start the simulation, the user presses the “cow” button (Why a cow? -- Why not?). *As of this writing, there are no sound effects associated with this button.* Agents will appear near their flags in opposite corners and the simulation will commence. Figure 22 shows the beginning of a simulation using the sample environment. Red agents are at the upper right, and blue agents at the lower left. The black squares seen in the center are obstacles. Other elements in the center are hills and foliage.



Figure 22. JACOB Sample Screen Shot

4. Pausing The Simulation

The simulation can be paused momentarily by pressing the stop button. Figure 23 shows a paused battle. Notice the blue squad breaking out towards the upper right. The light colored agent at the front is the squad leader of this group of agents. Notice also the scattered cluster of blue agents at the lower left, near the blue flag. They were unable to form squad relationships due to conditions not being met. There is not a sufficient number in either the group to the left and above the flag, or the group to the right, beside

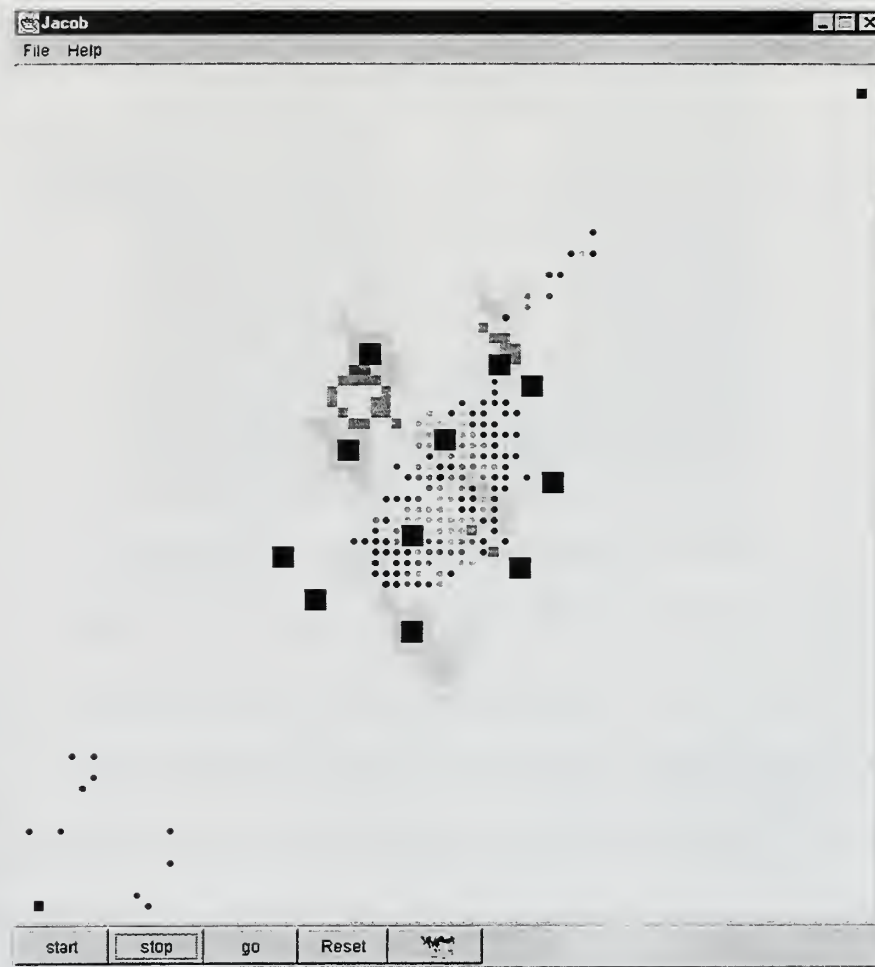


Figure 23. Battle Simulation in JACOB

the flag. Since they cannot sense each other, or the enemy, they remain stationary.

Future work is needed in this area to include motivation to move about so that they find sufficient numbers to form a squad.

5. The Brain Lid

Agent Brain Lid

RedJacobAgent23

Personality & Relationships

Independence	0.25
Aggressiveness	0.8
Self Preservation	0.2
Loyalty	0.2
Obedience	0.9
Army	Member
Squad	Member
Company	

Movement Goals

Active Move Goal	MaintainUnitCohesionGoal
Capture Enemy Flag	0.238625856124222
Defend Own Flag	0.009875791107551837
Minimize Injury	0.08800000000000001
Engage Enemy	0.22799999999999998
Maintain Unit Cohesion	0.8933181546628475
Seek Friendlies	0.21375
Agent Health	3 of 5

Sensed Environment

A grid of 10x10 dots representing the sensed environment. A solid black square is located at the bottom left of the grid, approximately at row 8, column 3.

OK

Figure 24. JACOB Brain Lid

While the simulation is paused, the user can select any agent displayed and get information on why the agent is doing what it is doing. This new window is called a “Brain Lid”, because it allows the user to ‘open up an agents lid and see what makes it tick’ (Figure 24). The Brain Lid displays the selected agent’s personality and any relationships it belongs to in the upper portion, the agent’s movement goals and their associated goal weights in the middle portion, as well as it’s health, and the agent’s sensed environment in the bottom portion. The sensed environment section displays the selected agent in the center, and all things that the agent can see in its sensor range. Line-of-sight has not been implemented in JACOB, as evident by the visible blue agents on the opposite side of the obstacle. A shadow-mask algorithm is needed to cast shadows on all areas not visible to the center, selected agent.

Another feature built into JACOB is that when an agent is selected as discussed above, not only does the Brain Lid appear, but the agent’s color also changes to green. If it is a squad leader, all of it’s squad members change color to light green. If it is a squad member, its leader is highlighted in light green.

6. Dynamic Goal Selection

There are a number of examples that clear illustrate the dynamic goal selection caused by changing environmental parameters as well as newly formed relationships. Two such examples will be discussed below. Unfortunately due to black and white printing, graphics associated with these examples do not show up well. Verbal

descriptions will be provided, and one color figure will be included in the first example for viewing by Internet or CD-ROM of this thesis.

a. Forming A New Squad

Besides the initial squad formation at the beginning of a simulation run, dynamic squad formation often occurs with the agents that were unable to form squad relationships. This usually happens when the opposing force breaks through the main battle and approaches the enemy flag. Figure 25 shows a group of blue agents coming up from the lower left. The groups of red agents, enclosed in circles, are stationary.

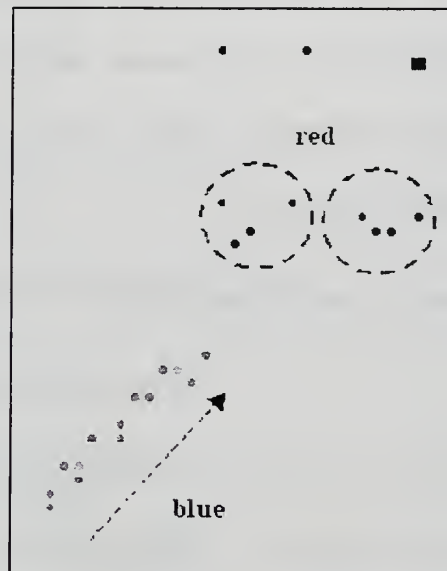


Figure 25. Blue Squad Approaching Stationary Red Agents

The stationary agents are not members of squads, and thus receive no external guidance on which direction to move. At this point, the red agent's only goals are those associated with being a soldier in the army relationship: *MaximizeEnemyCasualties*, *SeekFriendly*, *MinimizeInjury*, and *EngageEnemy*. The reasons why none of these four goals has any

affect on the agent's motion are highlighted below.

- MaximizeEnemyCasualties – This is a shooting-type goal and has no effect on an agent's motion.
- SeekFriendly – Each of these agents can detect friendly agents. This rule does not require them to get closer, or seek a minimum amount of friendlies.
- MinimizeInjury – Since there are no enemies shooting at them, and none of these agents are injured, this rule has no effect.
- EngageEnemy – There are no enemies within sensor range.

When the blue squad comes within sensing range of the red agents, they move to fulfill the *EngageEnemy* goal. This causes each red group to finally sense the other and they form a squad relationship. Figure 26 shows this formation, and the newly assigned leader in the middle of the circle.

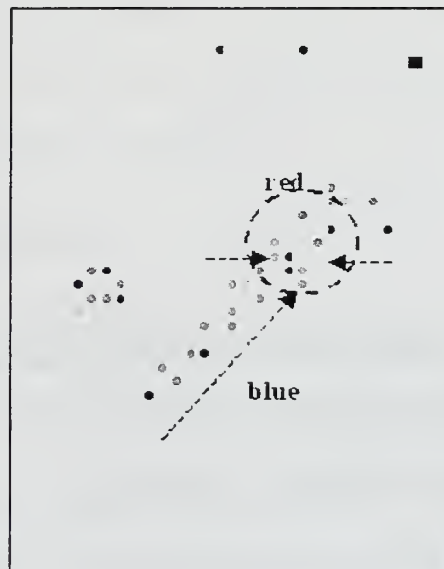


Figure 26. Red Agents Forming Squad

Now that the red agents have a squad leader, they receive new direction from the Company Commander, via the Squad Leader. Figure 27 shows the newly formed red squad breaking off engagement and moving down and to the left towards the blue flag.

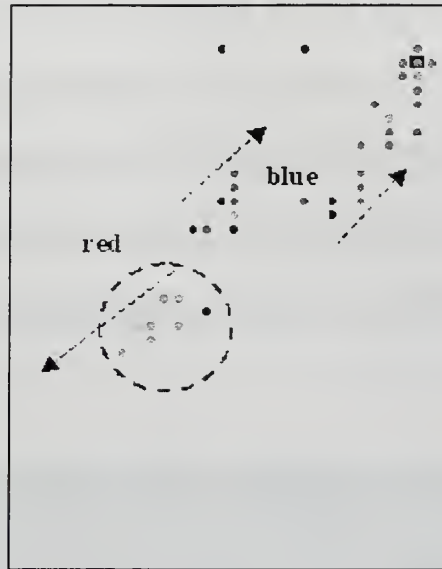


Figure 27. Red Squad Moves To Enemy Flag

b. Waiting For Stragglers

A similar example can be shown when a squad is disrupted during a battle. Once a squad is formed, the Squad Leader is in communication with each of his members. Because of this, if a squad member is held back from the squad, caught up in a quagmire of agents, the Squad Leader will hold the rest of the squad back, preventing them from going after the enemy flag while it waits for the stragglers to free themselves. If the stuck squad members die and there are still enough squad members left in the relationship, the Squad Leader will recognize this and press on to the objective. If

enough squad members die that the relationship can no longer exist, the squad will be disestablished and individual members will attempt to join other squads or reform new squads. Figure 28 shows a series of screen captures with a Squad Leader (in the diamond) and his partially stuck squad (all identified in the mass by circles). The sequence starts with the leader waiting for the stuck squad members, holding the other two back (Figure 28-1). Next, some of the members have freed themselves, but the leader still waits for stragglers (Figure 28-2). The last stuck member is killed, allowing the leader to begin moving (Figure 28-3). Finally, all of the members are free and following the leader towards the enemy flag (Figure 28-4).

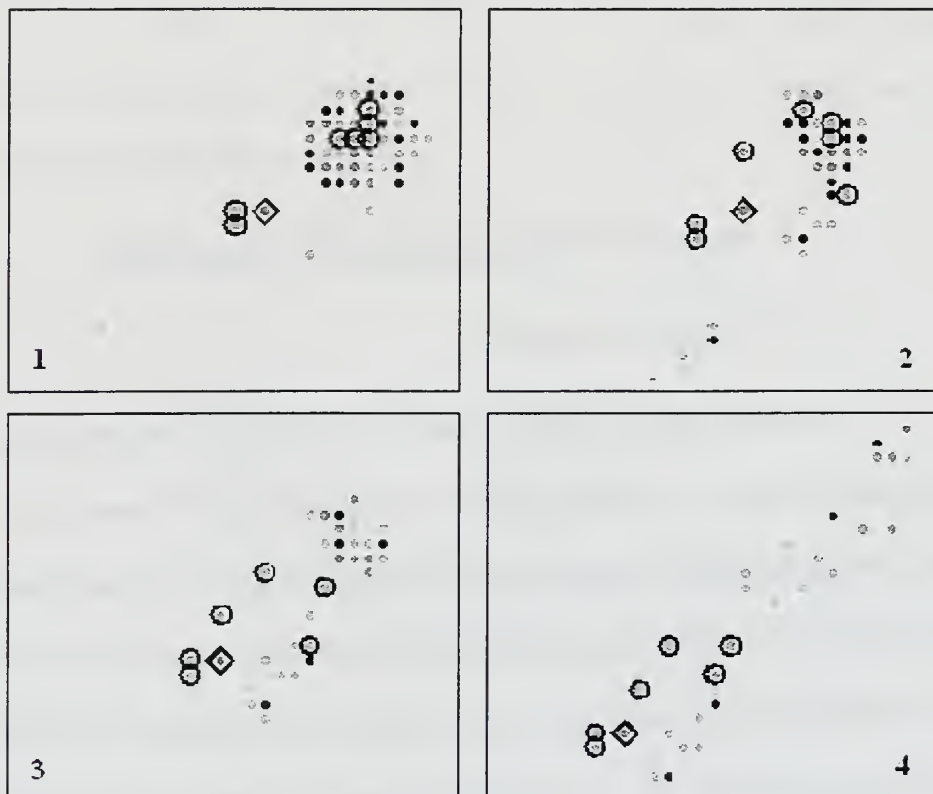
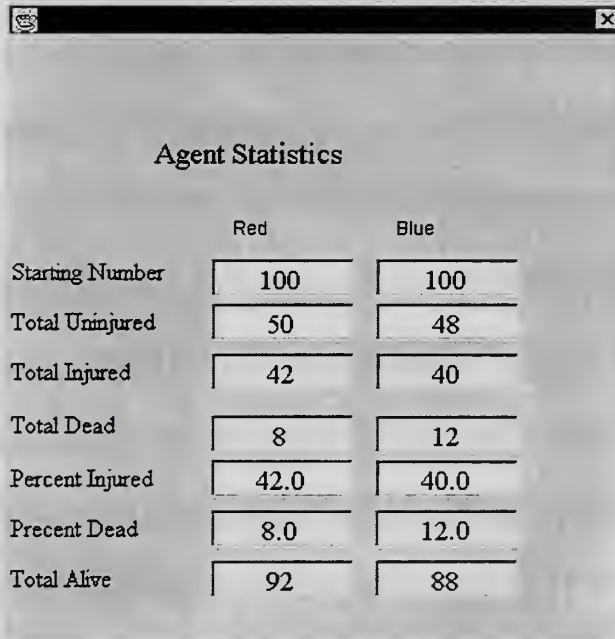


Figure 28. Waiting For Stragglers

7. Agent Statistics

It is recognized that models such as JACOB produce emergent behavior that is sometimes predictable, but often surprising. To help understand a particular run, or to collect data over a number of runs for analysis, certain statistical data must be recorded and displayed. Although insufficient for multiple runs, the Agent Statistics window that can be seen updating during the simulation run provides an example of methods that can be used to display key information. Figure 29 shows a sample of statistics gathered after a particular simulation run. This window is included for future developers to use as an example and starting point for gathering and displaying statistics, as well as any other data in the simulation.



	Red	Blue
Starting Number	100	100
Total Uninjured	50	48
Total Injured	42	40
Total Dead	8	12
Percent Injured	42.0	40.0
Precent Dead	8.0	12.0
Total Alive	92	88

Figure 29. Agent Statistics in JACOB

E. SUMMARY

This reference case provided a dramatic example of the strength of the *RELATE* design paradigm as well as the associated Java package. By working through the recipe, a clear path was established to build the underlying structure required to simulate two company-sized forces doing battle in a simplified environment. When agents are created on the battlefield in their opposing corners, the user can watch as squads are formed and Squad Leaders are assigned. The squads then move off with purpose as they go to engage the enemy and capture the opponent's flag. During battle, when attrition occurs, squads are dissolved and reformed *from the bottom up*, allowing continuing, dynamic role assignment and goal selection.

Future work on the JACOB simulation is needed to fully implement design features only suggested by this first attempt. Additional improvements could include:

- Further tuning of agents personality and goal measurements to make a more realistic simulation.
- Addition of multiple rules for each of the goals to allow dynamic rule selection such as that found in the El Farol reference case.
- Improvement of the GUI's to allow run time manipulation of static variables.
- Tuning and debugging the Simulation Agent Editor to allow the user to define various sensor ranges, firing ranges and speeds.
- Addition of a sound effect for the "cow" button.

VII. CONCLUSIONS AND RECOMMENDATIONS

Everything should be made as simple as possible, but not simpler.
- ALBERT EINSTEIN

A. CONCLUSION

RELATE provides a realistic, believable bottom-up simulation approach for modeling human decision-making and organizational behavior. The El Farol reference case demonstrated the ability to model individual decision-making in an unsure environment. It used multiple rules and could be easily modified to allow GA selection and discovery of the “best” rule sets, given a specific personality and environment. The CTF Agent simulation demonstrated the flexibility of the *RELATE* design paradigm and development package, allowing agents to be added to an existing, distributed simulation. The final reference case, JACOB, demonstrated the ability of *RELATE* agents to self organize into pre-determined relationships and select from competing goals.

B. RECOMMENDATIONS

The following paragraphs focus on improvements and recommendations for the *RELATE* design paradigm, as well as the associated Java classes and interfaces. Recommendations for improvements to the reference cases were included in their respective chapter summaries.

- More work is needed refining the Action and Sensor interfaces and their use.

This addresses gaining increased capabilities, either as in individual or as a

group when forming relationships. Also, some relationships may take away from, or diminish, existing capabilities.

- More work is needed in identifying and dealing with various obligations associated with forming relationships, as well as commitments to fulfilling these obligations. Agents may begin to earn reputations as they have multiple relationships.
- More work is needed smoothly integrating GA into *RELATE* agents to allow them to selectively improve their rule sets and explore leverage points in the environment.
- Additional relationships could be added to the potential relationship list once an agent is in another relationship.
- Dynamic relationship discovery mechanisms should be explored to allow agents to form relationships based on needs, capabilities, etc., vice a pre-determined list of potential relationships.
- Finally, explorations into the different meaning of relationship, that of association of ideas and concepts, using *RELATE*, or some derivative of *RELATE*, could yield interesting and powerful breakthroughs in the modeling of human cognition.

GLOSSARY

- Agent - A software object that perceives its environment through sensors and acts upon that environment through effectors to achieve one or more goals.
- Model – A description or analogy used to help visualize something that cannot be directly observed.
- Coordination – The act of managing interdependencies between activities performed to achieve a goal.
- Simulation – A method for implementing a model to play out the represented behavior over time.
- Adaptation – The process of modifying ones behavior over time to advantageously form a better fit to the environment.
- Complex adaptive system (CAS) – A self-organizing system that maintains coherence in a changing environment through interactions and adaptation.
- Learning - The acquisition of knowledge, formation of associations, and modification of behavior to improve performance based on exposure to and exploration of the environment.
- Evolution – A process of continuous change from a lower, simpler, or worse state to a higher, more complex, or better state.
- Multi-agent system (MAS) – A system in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks.
- MAS simulation – A rich, bottom-up modeling technique that uses diverse, multiple agents to imitate selected aspects of the real world system's active components.
- Supervised learning (or observational learning) – Learning from examples provided by a knowledgeable external supervisor or by observing others performing the same task or ability.
- Unsupervised learning (or autonomous learning) – Any type of learning that does not involve examples, expert advice, or direction.
- Relationship – The assembly of relations, i.e. understandings and/or commitments, between mutually interested parties that link certain individuals to others.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: SURVEY OF MAS SIMULATION ARCHITECTURES

Almost every major researcher has a methodology that they think is best for their particular area of study. Each of these models are well suited for the study of CAS in the context that they were designed. Almost none of them are readily available or easily understandable to the novice researcher. Some are available from the internet, but at the time of this writing, only a few can be successfully downloaded and run on a standard desktop computer or workstation without loading other additional software or having a specific operating system. Many of these researchers are currently working on Java based implementations to allow portability of their application or toolkit. Others are working on the extensibility of their code to allow web-based simulations. Most of the information presented in this Appendix has been obtained directly from applicable web sites or papers, modified only for clarity and ease of understanding. It is not the authors' intention to claim the following summaries as their own work, but merely to gather and present this information in an easily understandable format. The following summaries are provided as an aide for fellow researchers and interested parties in MAS simulations.

1. Echo (<http://www.santafe.edu/projects/echo/>)

Presented by John Holland and refined by researchers at the Santa Fe Institute and Central Michigan University, this simulation architecture was used to investigate the mechanisms that regulate diversity and information-processing in systems comprised of

many interacting adaptive agents, or CAS. Echo abstracts away virtually all of the physical details of real systems and concentrates on a small set of primitive agent-to-agent and agent-to-environment interactions. The extent to which Echo captures the essence of real systems is still largely undetermined. The goal of Echo is to study how simple interactions among simple agents lead to emergent high--level phenomena such as the flow of resources in a system or cooperation and competition in networks of agents (e.g., communities, trading networks, or arms races). Echo agents interact via combat, trade and mating and develop strategies to ensure survival in resource-limited environments. Individual genotypes encode rules for interactions. In a typical simulation, populations of these genomes evolve interaction networks that regulate the flow of resources. Resulting networks resemble species communities in ecological systems. Flexibly defined parameters and initial conditions enable researchers to conduct a range of "what-if" experiments (Holland, 1995).

An Echo world consists of a lattice of sites. Each is populated by some number of agents, and there is a measure of locality within each site. Sites produce different types of renewable resources; each type of resource is encoded by a letter (e.g., "a," "b," "c," "d"). Different types of agents use different types of resources and can store these resources internally. Sites charge agents a maintenance fee or tax. This tax can also be thought of as metabolic cost.

Agents fight, trade and reproduce. Fighting and trading result in the exchange of resources between agents. There is sexual and non--sexual reproduction, sexual

reproduction results in offspring whose genomes are a combination of those of the parents. Each agent's genome encodes various genes which determine how it will interact with other agents (e.g., which resource it is willing to trade, what sort of other agents it will fight or trade with, etc.). Some of these genes determine phenotypic traits, or "tags" that are visible to other agents. This allows the possibility of the evolution of social rules and potentially of mimicry, a phenomenon frequently observed in natural ecosystems. The interaction rules rely only on string matching.

Echo has no explicit fitness function guiding selection and reproduction. An agent self-reproduces when it accumulates a sufficient quantity of each resource to make an exact copy of its genome. This cloning is subject to a low rate of mutation. (Echo, 2000)

2. Swarm (<http://www.swarm.org/>)

A multi-agent software platform developed by Chris Langton, for the simulation of CAS. In the Swarm system the basic unit of simulation is the swarm, a collection of agents executing a schedule of actions. Swarm supports hierarchical modeling approaches whereby agents can be composed of swarms of other agents in nested structures. Swarm provides object-oriented libraries of reusable components for building models and analyzing, displaying, and controlling experiments on those models. Swarm is currently available as a beta version in full, free source code form. It requires the GNU C Compiler, Unix, and X Windows. (Minar, et al. 1996)

3. SugarScape (<http://www.brook.edu/SUGARSCAPE/>)

Developed by Robert Axtell and Joshua Epstein at the Center on Social and Economic Dynamics (CSED), Brookings Institute, Washington, D.C., Sugarscape can be used to model a variety of complex situations -- including an entire proto-history, complete with cultural evolution, population pressures, and warfare. Rather than design models from stem to stern, the authors grow them by imposing a few simple rules on Sugarscape's agents, then studying the aggregate effects of the resulting interactions. Sugarscape is presented in (Axtell and Epstein, 1996), a groundbreaking book that posits a new mechanism for studying populations and their evolution. By combining the disciplines of cellular automata and "artificial life", Epstein and Axtell have developed a mechanism for simulating all sorts of emergent behavior within a grid of cells managed by a computer. In their simulations, simple rules governing individuals' "genetics" and their competition for foodstuffs result in highly complex societal behaviors. The authors explore the role of seasonal migrations, pollution, sexual reproduction, combat, and transmission of disease or even "culture" within their artificial world, using these results to draw fascinating parallels with real- world societies. In their simulation, for instance, allowing the members to "trade" increases overall well-being but also increases economic inequality. In *Growing Artificial Societies*, the authors provide a workable framework for studying social processes in microcosm, a thoroughly fascinating accomplishment. (Sugarscape, 2000)

4. StarLogo (<http://el.www.media.mit.edu/groups/el/Projects/starlogo/>)

StarLogo was developed by Mitchel Resnick at the MIT Media Lab , Epistemology and Learning Group, Massachusetts Institute of Technology (MIT), Boston, MA (Resnick, 1998). It is a programmable modeling environment for exploring the workings of decentralized systems -- systems that are organized without an organizer, coordinated without a coordinator. With StarLogo, the user can model (and gain insights into) many real-life phenomena, such as bird flocks, traffic jams, ant colonies, and market economies. In decentralized systems, orderly patterns can arise without centralized control. Increasingly, researchers are choosing decentralized models for the organizations and technologies that they construct in the world, and for the theories that they construct about the world. But many people continue to resist these ideas, assuming centralized control where none exists -- for example, assuming (incorrectly) that bird flocks have leaders. StarLogo is designed to help students (as well as researchers) develop new ways of thinking about and understanding decentralized systems. StarLogo is a specialized version of the Logo programming language. With traditional versions of Logo, you can create drawings and animations by giving commands to graphic "turtles" on the computer screen. StarLogo extends this idea by allowing you to control thousands of graphic turtles in parallel. In addition, StarLogo makes the turtles' world computationally active: you can write programs for thousands of "patches" that make up the turtles' environment. Turtles and patches can interact with one another -- for example, you can program the turtles to

"sniff" around the world, and change their behaviors based on what they sense in the patches below. StarLogo is particularly well-suited for Artificial Life projects.

5. ISAAC-EINSTien (<http://www.cna.org/isaac/default.htm>)

ISAAC is an acronym for Irreducible Semi-Autonomous Adaptive Combat. It was designed by Andrew Ilachinski at the Center for Naval Analysis (CNA) in the early 1990's. ISAAC is a combat model designed to allow the user to explore the evolving patterns of macroscopic behavior that result from the collective interactions of individual agents, as well as the feedback that these patterns might have on rules governing the individual agents' behavior (Ilachinski, 1995). EINSTEIN is a follow-on project that takes lessons learned from ISAAC and incorporates much more functionality, as well as a windows-based development environment.

ISAAC/EINSTEIN are simple multiagent-based "toy models" of land combat that are being developed to illustrate how certain aspects of land combat can be viewed as emergent phenomena resulting from the collective, nonlinear, decentralized interactions among notional combatants. These models take a bottom-up, synthesist approach to the modeling of combat, vice the more traditional top-down, or reductionist view, and represent a first step toward developing a complex systems theoretic analyst's toolbox (or "conceptual playground") for exploring high-level emergent collective patterns of behaviors arising from various low-level (i.e., individual combatant and squad-level) interaction rules. The idea is not to model in detail a specific piece of hardware (M16 rifle, M101 105mm howitzer, etc.), but to provide an understanding of the fundamental

behavioral tradeoffs involved among a large number of notional variables. In ISAAC and EINSTEIN, the final outcome of a battle -- as defined, say, by measuring the surviving force strengths -- takes second stage to exploring how two forces might co-evolve during combat. (ISAAC, 2000)

6. AgentBuilder® (<http://www.agentbuilder.com>)

AgentBuilder is a commercially available integrated software development tool, available from Reticular Systems, Inc. It allows software developers with “no background in intelligent systems or intelligent agent technologies” to quickly and easily build intelligent agent-based applications. AgentBuilder reduces development time and development cost and simplifies the development of high-performance, robust agent-based systems. (AgentBuilder, 2000)

Software developers need a set of tools that will aid them in developing agent-based applications. Tools are needed that can help the software developer analyze the application domain; formally recognize and describe the concepts, relationships and objects relevant to that domain, and specify the behavior of the agent(s) operating in that domain. The software developer also needs tools that can specify a collection of agents; analyze and specify the messages and message protocols between agents; and execute and evaluate the actions of the agents. Reticular Systems, Inc. has developed the AgentBuilder toolkit which provides these capabilities. The AgentBuilder product consists of two major components: the development tools and the run-time execution environment. The development tools are used for analyzing an agent’s problem domain

and for creating an agent program that specifies agent behavior. The run-time system provides a high-performance agent engine that executes these agent programs. Agents constructed using AgentBuilder communicate using the Knowledge Query and Manipulation Language (KQML) and support the performatives defined for KQML. In addition, Agent-Builder™ allows the developer to define new interagent communications commands that suit his particular needs. The AgentBuilder toolkit and the run-time engine are implemented using the Java programming language. Thus, AgentBuilder will run on any platform that supports Java development. Agents created with AgentBuilder are themselves Java pro-grams and will execute on any platform with a Java virtual machine.

AgentBuilder allows software developers with no background in intelligent systems or intelligent agent technologies to quickly and easily build intelligent agent-based applications. AgentBuilder reduces development time and development cost and simplifies the development of high-performance, robust agent-based systems.

7. Sim_agent Toolkit (http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html)

Developed at the University of Birmingham's School of Computer Science, Sim_agent Toolkit uses the Pop-11 language in the Poplog software development environment. Pop-11, like Common Lisp, is a powerful extendable multi-purpose programming language supporting multiple paradigms. Within the Poplog environment it also supports programs written in Prolog, Common Lisp or Standard ML. The current version of the toolkit is very general and flexible, though perhaps not as easy to use as a

toolkit dedicated to a particular type of architecture. Designed to explore architectural design requirements for intelligent human-like agents, as well as others kinds of agents. It provides a facility for rapidly implementing and testing out different agent architectures, including scenarios where each agent is composed of several different sorts of concurrent interacting sub-systems, in an environment where there are other agents and objects. Agents can have sensors and effectors, and can communicate with other agents. Agents also have hybrid architectures including, for example, symbolic mechanisms communicating with neural nets.

8. ACE (<http://www.cecer.army.mil/pl/ace/homepage.html>)

“ACE is a software environment which facilitates collaboration between design activities through the use of agents. ACE is applicable to a wide variety of domains which can benefit from task automation, information processing and group activities. Agents are small expert systems that are tightly integrated with traditional CAD tools and other engineering applications. ACE agents communicate with each other using libraries of design objects such as beams, columns, or footings. Although most agents act under the user's direction, they can run in the background and act in an advisory capacity. Such an agent might use rule-based techniques to check for code violations or act as a source of expertise by making suggestions on improving design quality.”

“The primary role of agents in ACE is as design assistants that use heuristic rules and a powerful checklist facility to automate routine design tasks, thus enhancing productivity and ensuring repeatable design quality. Experienced designers can store their

knowledge in agents for use by others. The true strength of ACE, however, is as an integration platform. Each user has a workspace which allows the possibility of blurring the distinction between data in CAD drawings, analysis programs, and bid specifications. ACE improves document consistency by providing the user with a central object repository which reduces redundant data input and the associated risk of human error.”

“Beyond its ability to integrate design tools, ACE also has the ability to integrate architects and engineers through a virtual workspace which is comprised of multiple individual workspaces. Each user, and the agents they employ, contribute to an interest set for their workspace. Once a project leader has added them to a design project, users determine when to send and receive design information to other members of the design team. Notification of object and relationship instantiation or modification is broadcast over the virtual workspace based on each workspace's interest set, thus providing a mechanism for intelligent data replication. The virtual workspace also supports conflict detection, negotiation and resolution strategies to assist collaboration activities.”

9. Open Agent Architecture (OAA) (<http://www.ai.sri.com/~oaa/>)

Developed by SRI International, OAA is an open source, downloadable agent development architecture with libraries available in as many as 8 programming languages.

“When designing the Open Agent Architecture, we realized that it is imperative that the human user must be able to interact with the collection of distributed agents as an equal member of the community, not just as an outsider to whom is presented a result

once real agents have done all the work. Multiple agents can provide services for retrieval, combination, and management of the growing amount of online information, but this is only useful if controlling and interacting with the network of agents remains less complicated than interacting with the online services themselves!”

“With this in mind, we designed the InterAgent Communication Language (ICL) to be a logic-based declarative language capable of representing natural language expressions. In addition, we incorporated techniques into the architecture for communicating with agents using simultaneous multiple (natural) input modalities; humans can point, speak, draw, handwrite, or use standard graphical user interface when trying to get a point across to a collection of agents. The agents themselves will compete and cooperate in parallel to translate the user's request into an ICL expression to be handled. These techniques, in combination with the use of special class of agents called Facilitator agents (Facilitator agents reason about the agent interactions necessary for handling a given complex ICL expression), allow human users to closely interact with the ever-changing community of distributed agents.”

Characteristics:

- *Open*: agents can be created in multiple programming languages and interface with existing legacy systems.
- *Extensible*: agents can be added or replaced individually at runtime.
- *Distributed*: agents can be spread across any network-enabled computers.
- *Parallel*: agents can cooperate or compete on tasks in parallel.

- *Mobile*: lightweight user interfaces can run on handheld PDA's or in a web browser using Java or HTML and most applications can be run through a telephone-only interface.
- *Multimodal*: When communication with agents, handwriting, speech, pen gestures and direct manipulation (GUIs) can be combined in a natural way.

10. STEAM (<http://www.isi.edu/teamcore/tambe/steam/steam.html>)

“STEAM was developed at the Information Science Institute, University of Southern California, in Los Angeles, CA. STEAM is a general model of teamwork, intended to enable agents to participate in coherent teamwork. It has so far been applied in three different complex multi-agent domains. Two of the domains involve building teams of pilot agents for distributed interactive simulation environments: (i) synthetic helicopter attack, (ii) synthetic helicopter transports. A third domain is building a team of virtual players for RoboCup simulated soccer. While synthetic pilot-teams based on STEAM are currently participating in the STOW-97 exercises (virtual battlefield exercises involving thousands of virtual and real entities), our player-team based on STEAM won the third place in the RoboCup-97 synthetic soccer tournament held at IJCAI-97 in Nagoya, Japan.”

“STEAM is currently based in Soar. A key component of STEAM is the notion of a "team operator", which generalizes the use of operators in Soar. In particular, while

operators are used by individuals to engage in individual activities, team operators are those performed by multiple team members together, as a team --- of course, without a physically shared memory. Team operators are based on the *joint intentions framework*; please read the references above for pointers and other details. Team operators are distinguished from normal individual operators, because they are tagged as modeling a team activity, and because they activate the rules in the package listed below. However, individual operators are also within STEAM scope, at least to the extent that they bear upon team activities. Thus, STEAM's rules can apply to team operators, as well as individual operators. (Simultaneously, it is possible to circumvent STEAM if you must, since there are areas where STEAM falls short in modeling teamwork)."

11. Xraptor (<http://www.informatik.uni-mainz.de/~polani/XRaptor/XRaptor.html>)

"XRaptor is an environment for simulation of scenarios in continuous virtual multi-agent worlds. It is written in C++ for UNIX platforms with the X Window System and Motif 1.2. XRaptor allows studying the behavior of agents in different 2-D or 3-D continuous worlds. In the current version an agent is either a point in its world, or occupies a circular area or a spherical volume. It contains a sensor unit through which it obtains information about the outside world as well as an actor unit through which it performs actions. The agents are controlled by a user-defined control kernel. The control kernel determines an agent's action based on the sensory input. A typical control kernel is designed to maximize the agent's survival time. The user can construct an agent by implementing concrete control kernels derived from abstract classes (i.e. classes with

pure virtual methods) provided by the XRaptor environment, whose pure virtual methods have to be overridden by the user.”

12. WebSim (<http://www.isima.fr/andre/websim/>)

“Web-based Simulation of Agent Behaviors. In this work we present a web-based simulation of autonomous software agents enabling the user to change interactively their behavior. The simulation is implemented in the Java language in order to be incorporated into Web pages and run remotely. The main goal of this work is to provide an example of distributed exploration environment, allowing online changes on the behavioral model of a multiagent system, particularly when domain expert end users are not locally present.”

13. SeSAm (<http://ki-server.informatik.uni-wuerzburg.de/~vki/sesam/SeSAm2/SeSAm-short.html>)

“The SeSAm (Shell for Simulated Agent Systems) provides a generic environment for modeling and experimenting with agent-based systems. We specially focused on providing a framework for the easy construction of complex models. Although the idea of a domain-independent multi-agent simulation environment is not new, none of the existing environments fulfills the claim of usage without direct programming. Despite of providing a powerful general architecture or a rather focused one in many simulation environments the user has to program using a language that provides some additional specialized concepts or classes, but still is based on the syntax

of for example C++. SeSAm on the other side keeps the balance between generality and easy usage.”

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: *RELATE* RELEASE NOTES

The sample simulations provided in this thesis are an excellent source for simulation development using the *RELATE* architecture. It is intended that future development start with the installation and running of these simulations. The sample simulations cover a variety of situations and the future simulation developer should be able to use “code reuse” of a variety of algorithms found in the code provided. The following instructions are intended to give the user specific instructions to install and run these Java-based applications.

1. Check to see if you have the latest Java build on your machine.

- a. At the “C prompt” type: `java -version`. You should see something like:

```
java version "1.3.0"  
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.0-C)  
Java HotSpot(TM) Client VM (build 1.3.0-C, mixed mode)
```

The version should be “1.2.0” or higher.

2. If the latest Java JDK is not installed on the computer being used:

- a. Copy the “j2sdk1_3_0-win.exe” file to the computers desktop, or other temporary directory.

- b. Double click the icon to start installation, or run the “.exe” file. Java version 1.3.0 will be installed and set up on your machine.

- c. Java Docs 1.3 are also included on the CD if you're a developer and want the latest from documentation from Sun.
3. Copy the folders: "ElFarol" and "Jacob" into a new folder of your own choice (we recommend a new folder called RELATE).
4. To run the ElFarol and Jacob simulations simply double click, or run, the included ".bat" files included in these directories. If you're a little computer daring you can create shortcuts on your desktop and direct them to run these ".bat" files.
5. Running Capture the Flag is a little more complicated than simply installing Java and copying a few directories. We recommend going to <http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/download.html> and following the installation instructions. Running this simulation requires your computer to be on a LAN or at least have a network card (multicast capable) installed. Running Capture the Flag requires that you use Netscape Navigator with a Cosmo player plug-in. These programs are also available on the web site. The latest Capture the Flag build on the web site includes all the *RELATE* code found in this thesis.

For the simulation developer: The complete code listing for all reference simulations are included in the attached CD or is available at <http://www.npsnet.org/~moves/RELATE> or <http://www.rodgy.net/Kim/Relate.zip> . We encourage any interested parties to look through the code and if there are any questions or comments, please contact use through <http://www.Roddy.net/Kim>.

APPENDIX C: *RELATE* DESIGN FOR EL FAROL

Relationship – RelationshipElFarol

Role – RoleBarMember

Goal – GoalHappy

Rule – Rule1
Rule – Rule2
Rule – Rule3
Rule – Rule4
Rule – Rule5
Rule – Rule6
Rule – Rule7
Rule – Rule8
Rule – Rule9
Rule – Rule10
Rule – Rule11
Rule – Rule12
Rule – Rule13
Rule – Rule14
Rule – Rule15
Rule – Rule16
Rule – Rule17
Rule – Rule18
Rule – Rule19
Rule – Rule20
Rule – Rule21
Rule – Rule22
Rule – Rule23
Rule – Rule24
Rule – Rule25
Rule – Rule26
Rule – Rule27
Rule – Rule28
Rule – Rule29
Rule – Rule30
Rule – Rule31
Rule – Rule32

El Farol Design

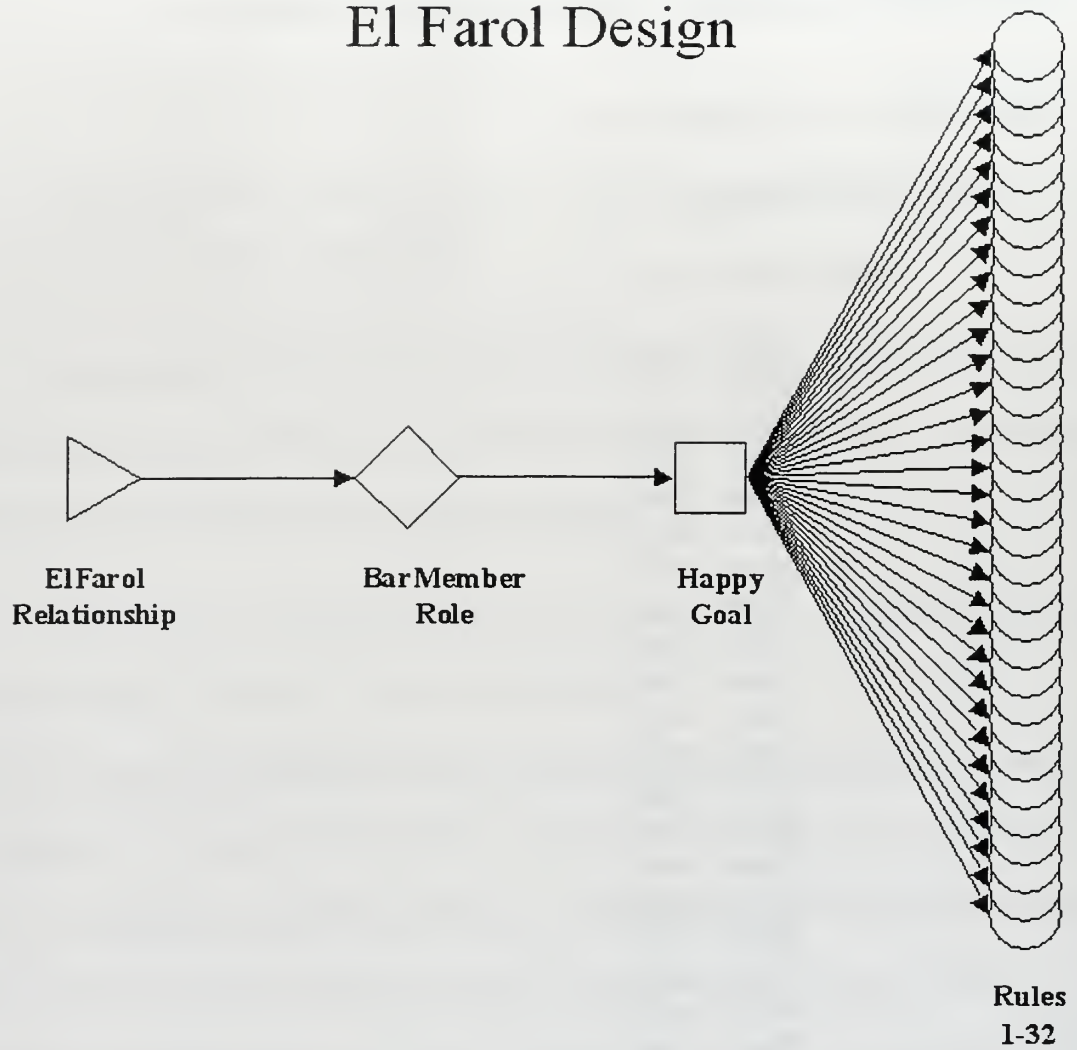


Figure 30. El Farol Design

APPENDIX D: *RELATE* DESIGN FOR CTF AGENT

Relationship – RelationshipBlueSquad

Role – RoleSquadLeader

Goal – GoalCoordinate

Rule – RuleCoordinate

Role – RoleSquadMember

Goal – GoalDefense

Rule – RuleDefense1

Goal – GoalOffense

Rule – RuleOffense1

Relationship – RelationshipRedSquad

Role – RoleSquadLeader

Goal – GoalCoordinate

Rule – RuleCoordinate

Role – RoleSquadMember

Goal – GoalDefense

Rule – RuleDefense1

Goal – GoalOffense

Rule – RuleOffense1

CTF Agent Design

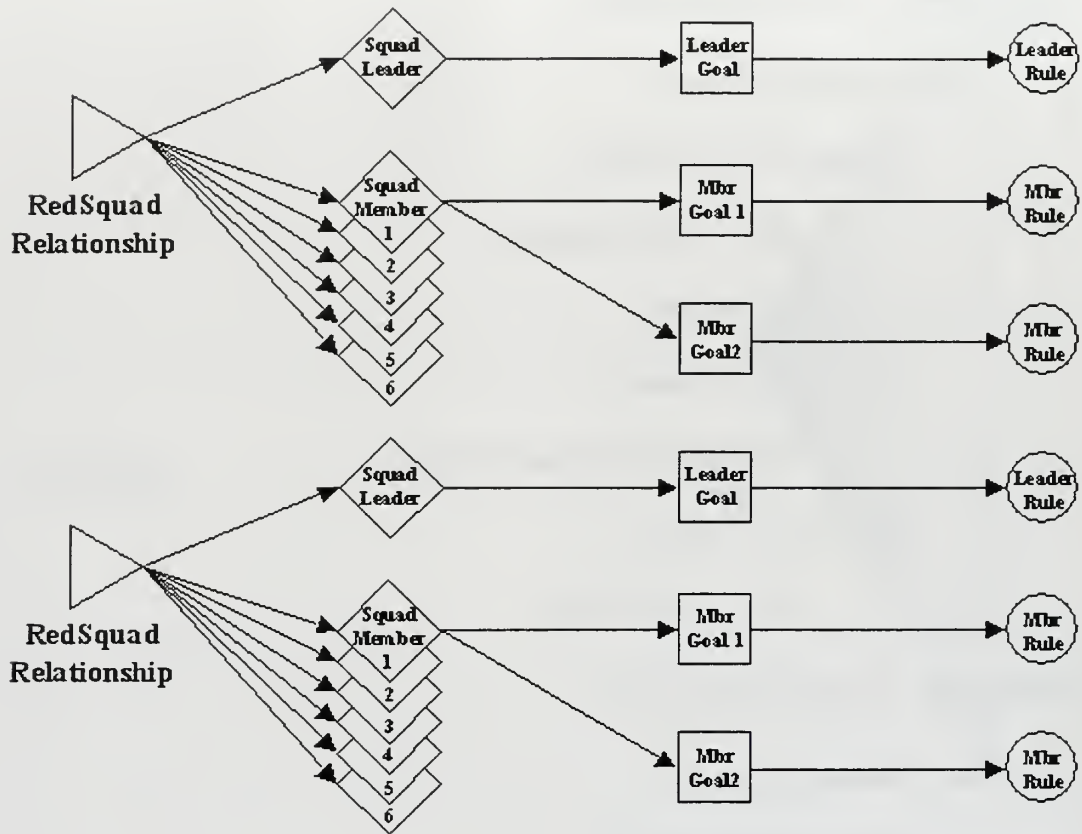


Figure 31. CTF Agent Design

APPENDIX E: *RELATE* DESIGN FOR JACOB

Relationship – ArmyRelationship (BlueArmy & RedArmy)

Role – SoldierRole (RedSoldier & BlueSoldier)

Goal – MaximizeEnemyCasualties

Rule – ShootAllPerceivedEnemyRule

Goal – SeekFriendly

Rule – MoveToClosestFriendlyRule

Goal – MinimizeInjury

Rule – DisengageEnemyRule

Goal – EngageEnemy

Rule – MoveToClosestEnemyRule

Relationship – SquadRelationship (BlueSquad & RedSquad)

Role – SquadLeaderRole

Goal – CaptureEnemyFlagGoal

Rule – MoveToEnemyFlagRule

Goal – DefendOwnFlagGoal

Rule – MoveToOwnFlagRule

Goal – KeepSquadLeaderInformedGoal

Rule – FullReportRule

Goal – MaintainUnitCohesionGoal

Rule – MoveToLeaderRule

Role – SquadMemberRole

Goal – CaptureEnemyFlagGoal

Rule – MoveToEnemyFlagRule

Goal – DefendOwnFlagGoal

Rule – MoveToOwnFlagRule

Goal – KeepSquadLeaderInformedGoal

Rule – FullReportRule

Goal – MaintainUnitCohesionGoal

Rule – MoveToLeaderRule

Relationship - CompanyRelationship

Role – CompanyCdrRole

Goal – CaptureEnemyFlagGoal

Rule – MoveToEnemyFlagRule

Goal – DefendOwnFlagGoal

Rule – MoveToOwnFlagRule

Goal – KeepCompanyCommanderInformedGoal

Rule – FullSquadReportRule

Role – CompanyMbrRole

Goal – KeepCompanyCommanderInformedGoal

Rule – FullSquadReportRule

JACOB Design

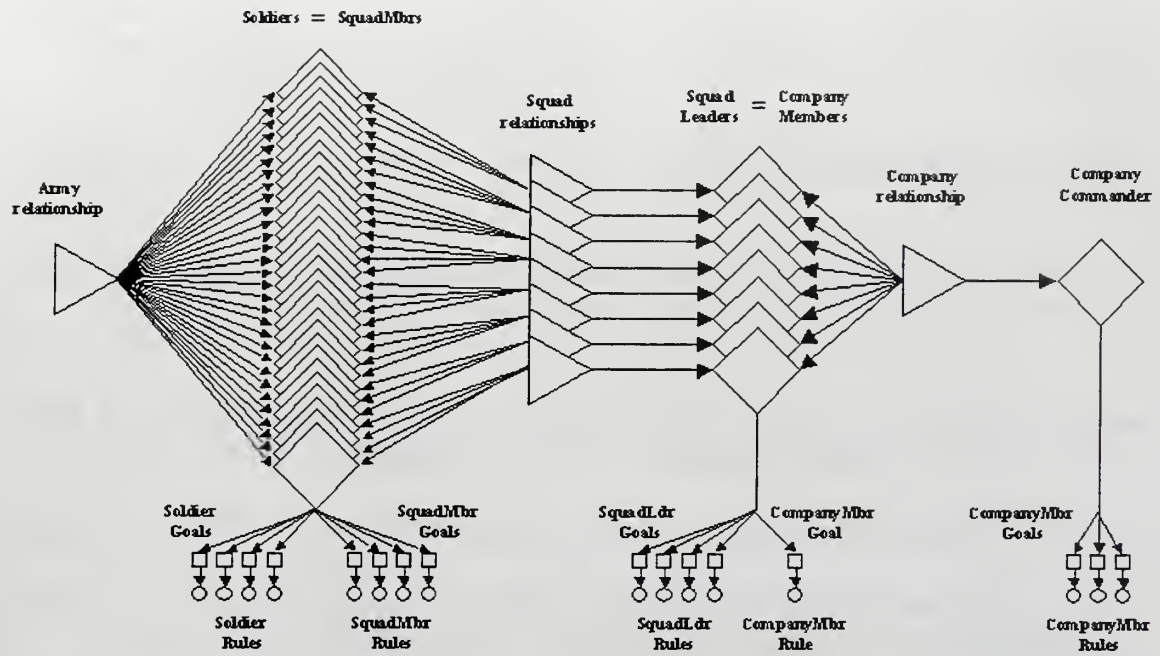


Figure 32. JACOB Design

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- Academy of Motion Picture Arts and Sciences (1997). *Scientific And Engineering Award*. http://www.oscars.org/ampas/plsql/ampas_award.detail?primekey_in=1999070716:28:14974243842 (30 Jul 00).
- AgentBuilder (2000). *AgentBuilder Home Page*. <http://www.agentbuilder.com/> (30 Jul 00).
- Arthur, W. B. (1994). "Inductive Reasoning and Bounded Rationality (The El Farol Problem)." *American Economic Review* (Papers and Proceedings), 84, 406-411, 1994.
- Axtell, R., and Epstein, J. M. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. Washington, D.C.: The Brookings Institute.
- Axelrod, R. (1984). *The Evolution of Cooperation*. Perseus Books Group.
- Axelrod, R. (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton, NJ: Princeton University Press.
- Briot, J. (1998). "Agents and Concurrent Objects" (An Interview with Les Gasser). *IEEE Concurrency*. 6 (4); 74 -77, 81; 1998.
- DECAF (2000). *DECAF Agent Framework*. <http://www.eecis.udel.edu/~decaf/> (30 Jul 00).
- DIS-Java-VRML (2000). *Distributed Interactive Simulation DiS-Java-VRML Working Group*. <http://web.nps.navy.mil/~brutzman/vrtp/dis-java-vrml/> (30 Jul 00).
- Echo (2000). *John Holland's Echo*. <http://www.santafe.edu/projects/echo/> (30 Jul 00).
- Ferber, J. (1999). *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*, (English edition) Harlow, England: Addison-Wesley.
- Festschrift (1999). *Festschrift in honor of John H. Holland, 15-18 May, 1999*. <http://www.pscs.umich.edu/jhhfest/proceedings.html> (30 Jul 00).
- Fisher, R. (1958). *The Genetical Theory of Natural Selection*. New York, NY: Dover Publications.

- Franklin, S. and Graesser, A. (1996). "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents." *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Berlin: Springer-Verlag.
- Gamecenter.com (1999). *Will Wright on The Sims*.
<http://www.gamecenter.com/News/Item/0,3,0-2739,00.html> (30 Jul 00).
- Gasser, L., Braganza, C. and Herman, N. (1987). "MACE: A Flexible Testbed for Distributed AI Research." In Michael Huhns, ed., *Distributed Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann Publishers, Inc.
- Gasser, L. (2000). *Les Gasser's C.V.* <http://www.topintegration.com/les-cv.html> (30 Jul 00).
- Glance, N. and Huberman, B. (1994). "The Dynamics of Social Dilemmas." *Scientific American*, March 1994.
- Hewitt, C. (1977). *Viewing Control Structures as Patterns of Passing Messages*. Artificial Intelligence 8 (3); 323-364; 1977.
- Hewitt, C. and Inman, J. (1991). "DAI Betwixt and Between: From 'Intelligent Agents' to Open Systems Science". *IEEE Transactions on Systems, Man, and Cybernetics*, 21 (6); 1409-1419; 1991.
- Hiles, J. (1999). *Course Notes for MV-4015 Agent-Based Autonomous Behavior for Simulations*. Winter, 2000, Naval Postgraduate School.
- Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity*. Reading, MA: Perseus Books.
- Holland, J. H. (1998). *Emergence*. Reading, MA: Helix Books.
- Hopkins, D. (2000). *Designing User Interfaces to Simulation Games: A summary of Will Wright's talk*. <http://catalog.com/hopkins/simcity/WillWright.html> (30 Jul 00).
- Huhns, M. and Stephens, L. (1999). "Multiagent Systems and Societies of Agents." In Gerhard Weiss, ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: The MIT Press.
- IBM (1997). *Mental Models and Game Play*.
<http://www.almaden.ibm.com/almaden/npuc97/1997/wright.htm> (30 Jul 00).

- Ilachinski, A. (1997). *Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare*. Center for Naval Analyses Research Memorandum CRM 97-61.10, August 1997. Alexandria, VA: Center for Naval Analyses.
- ISAAC (2000). *Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare*. <http://www.cna.org/isaac/extabs.htm> (30 Jul 00).
- JAFMAS (2000). *JAFMAS: A Java-based Agent Framework for Multi-Agent Systems*. <http://www.ececs.uc.edu/~abaker/JAFMAS/> (30 Jul 00).
- JATLite (2000). *JATLite*. <http://java.stanford.edu/> (30 Jul 00).
- Knapik, M. and Johnson, J. (1998). *Developing Intelligent Agents for Distributed Systems*. New York, NY: McGraw-Hill.
- KQML (2000). *What is KQML?* <http://www.cs.umbc.edu/kqml/whats-kqml.html> (30 Jul 00).
- Kuhn, T. (1962). *The Structure of Scientific Revolutions*, First edition, revised edition. Chicago, IL: The University of Chicago Press.
- Langton, C. (1989). *Artificial Life: Sante Fe Institute studies in the sciences of complexity*. (Proc. Vol. VI). Reading, MA: Addison-Wesley.
- Langton, C., Taylor, C., Farmer, J., and Rasmussen, S. (Ed.)(1990). *Artificial Life II*. Addison-Wesley.
- Langton, C. (Ed.) (1997). *Artificial Life: An Overview*. Cambridge, MA: The MIT Press.
- Lieken, A. (2000). *Alife Online 2.0*. <http://alife.org> (30 Jul 00).
- Maes, Pattie (1990). "Situated Agents Can Have Goals." In Pattie Maes, ed., *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. Special Issues of Robotics and Autonomous Systems, Cambridge, MA: MIT Press.
- Maes, Pattie (1995). "Modeling Adaptive Autonomous Agents." In Chris Langton, ed., *Artificial Life: An Overview, Complex Adaptive Systems series*. Cambridge, MA: The MIT Press.

- Malone, T. (1988). "What is coordination theory and How Can It Help Design Cooperative Work Systems?" *Proceedings of the Conference on Computer-Supported Cooperative Work*, Oct 1990.
- Maxis (1999). *History of a Classic*.
<http://www.simcity.com/us/buildframes.phtml?guide/classic/history> (30 Jul 00).
- Maxis (2000). *About The Sims*. <http://www.thesims.com/us/about/index.html> (30 Jul 00).
- Maxwell and Raab (1998). *Representing C4ISR in JWARS: Theater Level Abstractions and Modeling Principles (U)*.
- Merriam-Webster (2000). *Merriam-Webster OnLine: The Language Center*.
<http://www.m-w.com> (30 Jul 00).
- Minar, N., Burkhart, R., Langton, C., and Askenazi, M. (1996). *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations*.
<http://www.swarm.org/intro.html> (30 Jul 99).
- Minsky, M. (1985). *The Society of Mind*. New York: Touchstone, Simon & Schuster.
- MOVES (2000). *Naval Postgraduate School Modeling, Virtual Environments & Simulation (MOVES) Academic Group*. <http://www.npsnet.org/~moves/> (30 Jul 00).
- National Research Council (NRC) (1998). *Modeling Human and Organizational Behavior: Applications to Military Simulations*. Washington, D.C.: National Academy Press.
- Nwana, H. (1996). "Software Agents: An Overview." In *Knowledge Engineering Review*, Sept 1996. Cambridge University Press.
- Resnick, M. (1998). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. Cambridge, MA: The MIT Press.
- Reynolds, C. W. (1982) "Computer Animation with Scripts and Actors", in *Computer Graphics*, 16(3) (SIGGRAPH 82 Conference Proceedings) pages 289-296.
<http://www.red3d.com/cwr/papers/1982/ASAS82.html> (30 Jul 00).
- Reynolds, C. W. (1987) "Flocks, Herds, and Schools: A Distributed Behavioral Model", in *Computer Graphics*, 21(4) (SIGGRAPH 87 Conference Proceedings) pages 25-34. <http://www.red3d.com/cwr/papers/1987/boids.html> (30 Jul 00).

- Reynolds, C. (1999). *Individual-Based Models*. <http://www.red3d.com/cwr/ibm.html> (30 Jul 00).
- Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice-Hall, Inc.
- Scott, John P. (1958). *Animal Behavior*. The College Library of Biological Sciences. Chicago, IL: The University of Chicago Press.
- SRI (2000). *The Open Agent Architecture*. <http://www.ai.sri.com/~oaa/> (30 Jul 00).
- Stites, J. (1997). *And Then There Was A-Life: The Man Who Gave Birth to Artificial Life, John Holland and the Thinking Machine*. <http://www.omnimag.com/archives/features/alife> (30 Jul 00).
- Sugarscape (2000). *Welcome to Sugarscape*. <http://www.brook.edu/SUGARSCAPE/> (30 Jul 00).
- Swarm (2000). *Swarm Development Group: Swarm*. <http://www.swarm.org/intro.html> (30 Jul 00).
- Thinking Tools (1999). "Agent Based Adaptive Simulation Technology." <http://www.thinkingtools.com/html/technology.html> (2 Feb. 98) no longer available.
- Von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. Edited and completed by Arthur Burks. Urbana, IL: University of Illinois Press.
- Weiss, G. (Ed.) (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press.
- Widman, L. and Loparo, K. (1989). "Artificial Intelligence, Simulation, and Modeling: A Critical Survey." In L. Widman, K. Loparo, and N Nielsen, eds., *Artificial Intelligence, Simulation & Modeling*. New York, NY: John Wiley & Sons.
- Wired (1994). "Will Wright: The Mayor of SimCity." *Wired Archive 2.01*. Jan 1994. <http://www.wired.com/wired/archive/2.01/wright.html> (30 Jul 00).
- Wooldridge, M. and Jennings, N. R. (1995). "Intelligent Agents: Theory and Practice." *Knowledge Engineering Review*, October 1994. Cambridge University Press.
- Zeus (2000). *The Zeus Agent Building Toolkit*. <http://193.113.209.147/projects/agents/zeus/index.htm> (30 Jul 00).

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
8725 John J. Kingman Road, Suite 0944
Fort Belvoir, Virginia 22060-6218

2. Dudley Knox Library2
Naval Postgraduate School
411 Dyer Road
Monterey, California 93943-5101

3. RADM Richard W. Mayo.....1
CNO, N6
2000 Navy Pentagon
Washington, DC 20350-2000

4. CDR George Phillips, USN (Ret).....1
CNO, N6M1
2000 Navy Pentagon
Washington, DC 20350-2000

5. Dr. Allen Zeman.....1
CNO, N7B
2000 Navy Pentagon
Washington, DC 20350-2000

6. AFIT Academic Library.....1
ATTN: Barry Boettcher
2950 P Street
Area B, Building 642
AFIT/LDR
Wright Patterson AFB, Ohio 45433-7765

7. W. Lewis Johnson.....1
Director, Center for Advanced Research in Technology for Education (CARTE)
University of Southern California / Information Sciences Institute,
4676 Admiralty Way
Marina del Rey, California 90292

8. Mr. John Hiles..... 1
Research Professor
MOVES Academic Group
Naval Postgraduate School
Monterey, California 93943-5118
9. MOVES Academic Group Reference Library..... 1
Attn: Michael Zyda
Chair, MOVES Academic Group
Naval Postgraduate School
Monterey, California 93943-5118
10. LCDR Kimberly A. Roddy..... 1
7879 Amethyst Loop Road, NW
Bremerton, Washington 98312-1077
11. Tamara Velikonia..... 1
7045 Sonoma Highway
Santa Rosa, California 95409
12. LT Michael Dickson..... 1
S. 3019 Wilbur RD
Spokane, Washington 99206
13. Joan K. Grommet..... 1
S. 3019 Wilbur RD
Spokane, Washington 99206

63 290NP6 2485
TH
6/02 22527-200 NLE



DUDLEY KNOX LIBRARY



3 2768 00402782 1