INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

AgenTINA: An Agent-Based Architecture for TINA-Compliant Systems

By

Amir Zeid, B.Sc.

A Thesis submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute of Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

June 2000

© Copyright 2000, Amir Zeid

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-57636-1

Canada

The undersigned hereby recommend
to the faculty of Graduate Studies and Research
acceptance of the thesis

AgentTINA: An Agent-based Architecture for TINA-compliant Systems

Submitted by Amir Zeid, B.Sc.
In partial fulfillment of the requirements for
The degree of Doctor of Philosophy

Professor E. Kranakis
Director, School of Computer Science

Dean S. Mahmoud
Thesis Supervisor

Professor J. Pugh
Thesis Supervisor

Professor A. Mili
External Examiner

Carleton University
June 2000

ABSTRACT

In recent years, agent-based systems have received considerable attention in both academia and industry. The agent-oriented paradigm can be considered a natural extension to the object-oriented (OO) paradigm. In spite of the fact that there are many OO analysis and design methods, there is very little work reported on design and analysis of agent-based systems.

Today's telecommunication service and software management architectures are insufficient for the highly dynamic electronic service market of the future. One of the most promising candidates for the future telecommunication system architecture is the "Telecommunication Information Networking Architecture" (TINA). TINA was developed by the international TINA consortium (TINA-C). TINA is based on distributed object-oriented concepts and standards. Although there are possibilities for dynamic configuration and object migration, TINA is at its core still based on the traditional client-server concepts.

In this dissertation, we propose a software engineering process for agent-based systems based on existing object-oriented software engineering concepts. We give emphasis to the development of mobile agents systems. Then, we investigate the application of agents to enhance future telecommunications architectures. More specifically, we introduce AgentTINA which is an agent-based architecture for TINA-compliant architectures. AgentTINA is designed using the software engineering process we propose. We verify the reliability of AgentTINA design using Petri-Nets. We also present an application developed using AgentTINA components.

ACKNOWLEDGMENTS

Patience is a virtue but persistence to the point of success is a blessing. I would like to thank God for granting me patience and blessing me with persistence to achieve one of my dreams. I would like to thank my family, the greatest family in the world, for their continuous moral support. Special thanks are due to my dad who is always there for me as a father and as an understanding friend. I would like to thank my mom for helping me throughout my early school years and for her dedication to our family. I do not know how to express my sincere thanks to my sister who taught me that faith always leads to the right path. She is probably the main reason of my success.

I would like to thank my thesis supervisor John Pugh. Professor Pugh was the one who inspired me to start my research in object-oriented systems. I would like also to thank professor Oommen, his care about graduate students is really appreciated. Finally, I want to thank professor Mili for his helpful comments and cooperation.

The financial support of the Natural Sciences and Engineering Research Council of Canada, Ontario Ministry of Education, and CITO helped to keep me in school till I finished my degree. Also, special thanks are due to the School of Computer Science at Carleton University for granting me teaching assistantship throughout my studies.

I would like to thank my best friend, Ahmed Kamel for standing by my side when I really needed a friend. Ahmed shared with me some of the most difficult days of my life and he was always a good listener and a sincere friend. I would like also to acknowledge my late grand mother who taught me to be patient and pure. Finally, I want to express my appreciation for my future wife, for being caring, supportive and extremely patient. Her moral support was one of the main factors that kept me motivated.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PROBLEM OVERVIEW	1
1.2 MOTIVATIONS	3
1.2.1 AGENT-BASED SOFTWARE PARADIGM	3
1.2.2 FUTURE TELECOMMUNICATIONS ARCHITECTURES.....	4
1.3 RESEARCH APPROACH	7
1.3.1 DEVELOPING AN AGENT-BASED ANALYSIS/DESIGN PROCESS	7
1.3.2 APPLYING THE METHOD TO DEVELOP AGENTINA	8
1.4 OBJECTIVES AND SUMMARY OF FEATURES AND CONTRIBUTIONS	9
1.4.1 MAIN FEATURES OF AGENT-BASED SOFTWARE ENGINEERING PROCESS	9
1.4.2 MAIN FEATURES OF AGENTINA	9
1.5 THESIS ORGANIZATION.....	10

CHAPTER 2 AGENTS LITERATURE SURVEY AND RELATED RESEARCH

2.1 BACKGROUND.....	13
2.1.1 DEFINITIONS	14
2.1.1.1 <i>Fundamental Attributes of Agents</i>	15
2.1.1.2 <i>Other Attributes of Agents</i>	16
2.1.2 TYPES OF AGENTS	17
2.1.3 ARCHITECTURE OF MULTI-AGENT SYSTEMS AND AGENTS COMMUNICATIONS.....	20
2.1.4 MOBILE AGENTS.....	24
2.1.4.1 <i>Approach</i>	24
2.1.4.2 <i>What is a mobile agent?</i>	25
2.1.4.3 <i>Conceptual architecture for mobile agents</i>	26
2.1.4.4 <i>Problems introduced due to mobility</i>	31

2.1.4.5 Alternatives to mobile agents.....	31
2.1.5 EXAMPLES OF AGENTS.....	32
2.1.6 AGENT DEVELOPMENT ENVIRONMENTS.....	33
2.1.6.1 Java.....	33
2.1.6.2 Aglets from IBM.....	33
2.1.6.3 Concordia from Mitsubishi.....	35
2.1.6.4 Voyager from ObjectSpace.....	36
2.1.6.5 Microsoft Agents.....	36
2.1.7 AGENTS STANDARDS BODIES.....	37
2.1.8 FUTURE OF MOBILE AGENTS.....	37
2.2 RELATED RESEARCH.....	39
2.3 SUMMARY.....	41

CHAPTER 3 INTELLIGENT NETWORKS AND TINA LITERATURE SURVEY

3.1 BACKGROUND.....	42
3.1.1 INTELLIGENT NETWORKS (IN).....	42
3.1.1.1 What is IN?.....	43
3.1.1.2 IN Elements.....	46
3.1.1.3 The attractions of IN.....	47
3.1.2 TINA.....	47
3.1.2.1 What is TINA?.....	47
3.1.2.2 Basic Definitions.....	48
3.1.2.3 Organization of concepts and principles.....	50
3.1.2.4 Decomposition of the architecture.....	51
3.1.2.5 Relationships between the architectures.....	52
3.1.2.6 Conformance.....	53
3.1.2.7 Overall framework of TINA systems.....	54
3.1.2.8 Main separations.....	55
3.1.2.9 Service architecture.....	57
3.1.2.10 Advantages of TINA.....	59
3.1.2.11 Open issues for research.....	59
3.2 RELATED RESEARCH.....	59
3.2.1 ACTIVE NETWORKS.....	59
3.2.1.1 Massachusetts Institute of Technology.....	60
3.2.1.2 University of Pennsylvania.....	60
3.2.1.3 Columbia University.....	60
3.2.1.4 University of Arizona.....	60
3.2.2 NETWORK MANAGEMENT USING AGENTS.....	61
3.2.2.1 University of Berlin.....	61
3.2.2.2 Carleton University.....	61
3.2.2.3 EURESCOM.....	61

3.2.2.4 Network agent vendors.....	62
3.3 SUMMARY.....	63

CHAPTER 4 AGENT-BASED SOFTWARE ENGINEERING

4.1 AGENT-BASED DEVELOPMENT VERSUS OTHER DEVELOPMENT PARADIGMS	64
4.1.1 AOP VERSUS OOP	67
4.1.2 AOP VERSUS CLIENT/SERVER.....	68
4.1.3 OO ANALYSIS AND DESIGN METHODS	69
4.1.3.1 Responsibility-Driven Design.....	69
4.1.3.2 Unified Modeling Language (UML)	70
4.1.3.3 Conclusion.....	74
4.2 SYSTEMATIC PROCESS FOR AGENT-BASED DEVELOPMENT	76
4.2.1 DEVELOPMENT LIFE-CYCLES.....	76
4.2.2 ANALYSIS	79
4.2.2.1 Define tasks	79
4.2.2.2 Define Places.....	80
4.2.2.3 Identify Agents.....	82
4.2.2.4 Identify classes and objects.....	85
4.2.2.5 Model relationships between agents, classes and places	86
4.2.2.6 Create data dictionary.....	86
4.2.3 DESIGN	87
4.2.3.1 Define authorization rights	88
4.2.3.2 Define State charts for agents, classes and places.....	89
4.2.3.3 Life-cycle model	89
4.2.3.4 Define subsystems and neighborhoods	89
4.2.3.5 Describe the dynamic behavior of the system.....	91
4.2.3.6 Detailed design of methods, attributes of each place, class and agent.....	91
4.2.4 DEPLOYMENT.....	93
4.2.5 VERIFICATION USING PETRI-NETS	93
4.2.5.1 Marking.....	95
4.2.5.2 Transition firing and reachability	95
4.2.5.3 Incidence matrix.....	97
4.2.5.4 Safe Petri-Net	98
4.2.5.5 Liveness and Deadlock	98
4.3 STEREOTYPES (EXTENSIONS TO UML LANGUAGE).....	99
4.4 SUMMARY.....	100

CHAPTER 5 AGENTINA REQUIREMENTS SPECIFICATIONS

5.1 OVERVIEW	102
5.1.1 USER REQUIREMENTS	104
5.1.1.1 Connection to be set at any place any time	104
5.1.1.2 Dynamic reconfiguration	105
5.1.1.3 Reliability	105
5.1.1.4 Fault Tolerance	105
5.1.1.5 On line billing	105
5.1.1.6 Security Levels	106
5.1.1.7 Intelligent/flexible customer care	106
5.1.2 SERVICE PROVIDER REQUIREMENTS	106
5.1.2.1 Rapid service development and provisioning	106
5.1.2.2 Tailored services	107
5.1.2.3 Support for a multi-player environment	107
5.1.2.4 Profit Maximization	107
5.1.2.5 Negotiation position with customer	108
5.1.2.6 Negotiation position with network provider	108
5.1.2.7 User satisfaction	108
5.1.3 THIRD PARTY (NETWORK OPERATOR) REQUIREMENTS	108
5.1.3.1 Profit Maximization	108
5.1.3.2 Negotiation Position	109
5.2 SATISFYING THE REQUIREMENTS	109
5.2.1 USER REQUIREMENTS	111
5.2.2 SUPPORT FOR A MULTI-VENDOR MARKET	112
5.2.3 SERVICE MANAGEABILITY	112
5.2.4 PROVIDER REQUIREMENTS	113
5.3 AGENTS REQUIREMENTS	113
5.3.1 RELIABILITY	113
5.3.2 FAULT TOLERANCE	114
5.3.3 TRACEABILITY	114
5.4 SUMMARY	114

CHAPTER 6 AGENTINA ARCHITECTURE

6.1 MAIN SUBSYSTEMS AND RATIONALE	115
6.1.1 PCA MANAGEMENT	116
6.1.2 SHOPPING ENVIRONMENT	116
6.1.3 CONNECTION & NOTIFICATION MANAGEMENT	117

6.1.4 SERVICE MANAGEMENT ENVIRONMENT	117
6.1.5 CONVERSION UTILITIES MANAGEMENT	117
6.2 GENERAL COMPONENTS.....	117
6.2.1 FACILITATORS.....	117
6.2.2 TRIGGERS	118
6.2.3 NOTIFICATION AGENTS	119
6.2.3.1 <i>Typical Events</i>	119
6.2.3.2 <i>Reasons for notification</i>	121
6.2.3.3 <i>Methods of notification</i>	121
6.2.3.4 <i>Types of notification</i>	122
6.2.4 THE MICROSOFT AGENT.....	122
6.3 PCA MANAGEMENT	123
6.3.1 INTERFACE AND OVERVIEW	123
6.3.2 MAIN AGENTS, PLACES AND CLASSES	125
6.3.2.1 <i>PCA</i>	125
6.3.2.2 <i>NamedPCA</i>	126
6.3.2.3 <i>AnonPCA</i>	130
6.3.2.4 <i>PCAF</i>	131
6.3.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES	131
6.3.4 DATA DICTIONARY	131
6.4 SHOPPING ENVIRONMENT.....	132
6.4.1 INTERFACE AND OVERVIEW	133
6.4.2 MAIN AGENTS, PLACES AND CLASSES	134
6.4.2.1 <i>Shopper</i>	134
6.4.2.2 <i>Store</i>	135
6.4.2.3 <i>Mall</i>	135
6.4.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES	136
6.4.4 DATA DICTIONARY	137
6.5 NOTIFICATIONS & CONNECTION MANAGEMENT	137
6.5.1 INTERFACE AND OVERVIEW	137
6.5.1.1 <i>Dialer</i>	138
6.5.1.2 <i>Browser</i>	139
6.5.1.3 <i>Email</i>	140
6.5.1.4 <i>WWW pages monitor</i>	142
6.5.2 MAIN AGENTS PLACES AND CLASSES	142
6.5.2.1 <i>Email monitor</i>	142
6.5.2.2 <i>www monitor</i>	143
6.5.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES	144
6.5.4 DATA DICTIONARY	145
6.6 SERVICE MANAGEMENT	145
6.6.1 INTERFACE AND OVERVIEW	146

6.6.1.1 Monitoring a network addition.....	146
6.6.1.2 Negotiation.....	148
6.6.2 MAIN AGENTS, PLACES AND CLASSES	149
6.6.2.1 Facilitator	149
6.6.2.2 Service Provider Agent (SPA)	151
6.6.2.3 Network Provider Agent (NPA)	152
6.6.2.4 Service Upgrading Agent (SUA).....	154
6.6.2.6 Service Detection Agent (SDA)	154
6.6.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES	155
6.6.4 DATA DICTIONARY	156
6.7 UTILITY MANAGEMENT	156
6.7.1 INTERFACE AND OVERVIEW	156
6.7.2 MAIN AGENTS, PLACES AND CLASSES	158
6.7.2.1 Microsoft agent.....	158
6.7.2.2 Code generator.....	159
6.7.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES	160
6.7.4 DATA DICTIONARY	161
6.8 SUMMARY	161

CHAPTER 7 VERIFICATION OF AGENTINA DESIGN USING PETRI-NETS

7.1 PROCESS OF VERIFICATION.....	162
7.2 SHOPPING USING MOBILE AGENTS.....	164
7.2.1 SCENARIO DESCRIPTION	164
7.2.2 LIFE-CYCLE MODEL.....	164
7.2.3 INTERACTION DIAGRAM.....	165
7.2.4 PETRI-NET MODEL.....	166
7.2.4.1 The net (places and transitions)	166
7.2.4.2 Reachability Graph.....	167
7.3 NEGOTIATION BETWEEN PCA AND SPA.....	168
7.3.1 SCENARIO DESCRIPTION	168
7.3.2 LIFE-CYCLE MODEL	168
7.3.3 INTERACTION DIAGRAM.....	169
7.3.4 PETRI-NET MODEL.....	170
7.3.4.1 The net (places and transitions)	170
7.3.4.2 Reachability Graph.....	171
7.4 ESTABLISHING A CONNECTION FOR A MOBILE AGENT	173
7.4.1 SCENARIO DESCRIPTION	173
7.4.2 LIFE-CYCLE MODEL	173
7.4.3 PETRI-NET MODEL.....	174

7.4.3.1 The net (places and transitions)	174
7.4.3.2 Reachability Graph.....	175
7.5 EMAIL MONITORING.....	176
7.5.1 SCENARIO DESCRIPTION	176
7.5.2 LIFE-CYCLE MODEL.....	177
7.5.3 INTERACTION DIAGRAM.....	177
7.5.4 PETRI-NET MODEL.....	178
7.5.4.1 The net (places and transitions)	178
7.5.4.2 Reachability Graph.....	179
7.6 UPGRADING A SERVICE.....	181
7.6.1 SCENARIO DESCRIPTION	181
7.6.2 LIFE-CYCLE MODEL.....	181
7.6.3 INTERACTION DIAGRAM.....	181
7.6.4 PETRI-NET MODEL.....	182
7.6.4.1 The net (places and transitions)	183
7.7 SUMMARY.....	184
 <u>CHAPTER 8 APPLICATIONS: UNIFIED MESSAGING</u>	
8.1 WHAT IS UNIFIED MESSAGING?!	185
8.2 WHY AGENTS IN UNIFIED MESSAGING?	186
8.3 MAIN AGENTS, PLACES AND CLASSES.....	187
8.3.1 PCA	187
8.3.2 UTILITIES (CONVERSION AGENTS)	187
8.3.2.1 Text-Speech Agent	188
8.3.2.2 Fax-Text Agent	188
8.3.2.3 Video-Audio Agent.....	188
8.3.3 CONNECTION AGENTS	188
8.3.3.1 WWW Connector.....	189
8.3.3.2 Pager Connector.....	189
8.3.3.3 Phone Connector	189
8.3.4 NETWORK ELEMENTS AGENTS	189
8.3.5 FACILITATOR	191
8.3.6 DATABASE AND CALENDAR AGENTS	191
8.3.7 NOTIFY	192
8.3.8 TRIGGERS	192
8.4 RELATIONSHIPS BETWEEN AGENTS AND CLASSES.....	192
8.5 SCENARIO.....	193

8.6 RELATED WORK.....	196
------------------------------	------------

8.7 SUMMARY.....	197
-------------------------	------------

CHAPTER 9 CONCLUSION AND SUGGESTIONS FOR FUTURE RESEARCH

9.1 CONCLUSION	198
-----------------------------	------------

9.1.1 MAIN FEATURES AND CONTRIBUTIONS OF AGENT-BASED SOFTWARE ENGINEERING PROCESS	200
9.1.2 MAIN FEATURES AND CONTRIBUTIONS OF AGENTINA.....	201
9.1.3 NEW (UNIQUE) CONCEPTS.....	202
9.1.4 PROBLEMS AND OBSERVATIONS.....	203

9.2 DIRECTIONS FOR FUTURE RESEARCH	205
---	------------

9.2.1 WEARABLE COMPUTERS.....	205
9.2.2 VIRTUAL CORPORATIONS	205
9.2.3 AGENT STANDARDIZATION.....	206
9.2.4 AGENT-BASED SOFTWARE ENGINEERING.....	206
9.2.5 AGENT-BASED TECHNOLOGY IN TELECOMMUNICATIONS MARKETPLACE.....	207
9.2.5.1 Service Providers.....	207
9.2.5.2 Product Development.....	208
9.2.5.3 Agent Technology Developers.....	208
9.2.6 AGENT ORIENTED SERVICE DEVELOPMENT	209
9.2.7 AGENTINA AND AGENT-BASED TELECOMMUNICATIONS ARCHITECTURES	209

REFERENCES

AGENTS.....	211
MOBILE AGENTS	214
INTELLIGENT NETWORKS AND TINA	216
OBJECT-ORIENTED ANALYSIS/DESIGN	217

APPENDIX A JAVASCRIPT CODE GENERATED BY AGENTINA

CODE SAMPLE	218
-------------------	-----

APPENDIX B PETRI-NET ANALYSIS

1-DISPATCHING A MOBILE AGENT.....	223
2-NEGOTIATION BETWEEN PCA AND SPA	224
3- ESTABLISHING A CONNECTION FOR A MOBILE AGENT	226
4- MONITORING EMAIL.....	227

ABBREVIATIONS

ABBREVIATIONS LIST	229
--------------------------	-----

LIST OF FIGURES

Figure 1-1 Evolution waves.	3
Figure 1-2 The evolution of telecommunications standards.	6
Figure 1-3 Research approach.	8
Figure 1-4 Relationships between thesis chapters.	11
Figure 2-1 Federated systems	21
Figure 2-2 Mobile Agents Approach.	25
Figure 2-3 Conceptual model for mobile agent computing	30
Figure 2-4 Mobile code, the Java style	34
Figure 3-1 The intelligent network conceptual model.	44
Figure 3-2 The global service logic.	45
Figure 3-3 IN elements.	46
Figure 3-4 The TINA architecture.	52
Figure 3-5 Relationships between Tina's sub-architectures.	53
Figure 3-6 Basic structure of telecommunication software in a TINA environment.	54
Figure 3-7 Main separations in TINA.	56
Figure 3-8 The session concept in TINA.	58
Figure 4-1 Evolution of software development paradigms [Pont-96].	66
Figure 4-2 Evolution of autonomous-agent systems [Bak-96].	66
Figure 4-3 Classes notations in UML[Rat-98].	71
Figure 4-4 Sequence charts in UML.	72
Figure 4-5 State charts in UML.	72
Figure 4-6 Deployment models in UML.	73
Figure 4-7 The fountain life-cycle.	77
Figure 4-8 SOMA's life-cycle.	78
Figure 4-9 Life cycle model for agent-based development process.	78
Figure 4-10 Analysis process.	79

Figure 4-11 Task template.....	80
Figure 4-12 Place template.....	82
Figure 4-13 Agent template.	84
Figure 4-14 Notations for Agents, Mobile Agents and Places.....	85
Figure 4-15 Notations for relationships between component.....	86
Figure 4-16 Data dictionary.	87
Figure 4-17 Design process.....	88
Figure 4-18 Neighborhood diagram.....	90
Figure 4-19 Life-cycle notations.....	90
Figure 4-20 Itinerary graph.	92
Figure 4-21 Interaction diagram.....	92
Figure 4-22 An example of a Petri-Net.	94
Figure 4-23 Reachability graph.....	96
Figure 4-24 Extensions to UML.....	99
 Figure 5-1 AgenTINA involved parties.....	 103
 Figure 6-1 Main subsystems of AgenTINA.....	 116
Figure 6-2 Microsoft agent (Genie).....	123
Figure 6-3 PCA Management Interface.....	123
Figure 6-4 Current PCAs.	124
Figure 6-5 Relationships between PCA agent and classes.	132
Figure 6-6 Shopping Demo Interface.....	133
Figure 6-7 Mobile agent shoppers.	134
Figure 6-8 Relationships between shopping components.....	136
Figure 6-9 Communication management interface.....	138
Figure 6-10 Dialer interface.	139
Figure 6-11 Establishing a connection.....	139
Figure 6-12 Browser interface.	140
Figure 6-13 Emailer interface.	141
Figure 6-14 Email monitor.	141
Figure 6-15 Webpages monitor.....	142
Figure 6-16 Relationships between components for email monitor.....	144
Figure 6-17 Relationships between components for webpages monitor.....	145
Figure 6-18 Service management interface.....	146
Figure 6-19 Adding a network printer.	147
Figure 6-20 Adding a network printer.	147
Figure 6-21 The agent notifies users that a network printer was added.....	148
Figure 6-22 Negotiation between SPA and PCA.....	149
Figure 6-23 Relationships between service agents.	155
Figure 6-24 Testing the agents actions.....	157

Figure 6-25 Text-to-speech interface.....	157
Figure 6-26 Relationships between utilities agents.....	160
Figure 7-1 Verification process.....	163
Figure 7-2 Itinerary graph for a shopper.....	165
Figure 7-3 Interaction diagram for a shopping scenario.....	165
Figure 7-4 Petri-net model for a shopping scenario.....	166
Figure 7-5 Reachability graph for dispatching mobile agents.....	167
Figure 7-6 Interaction diagram for negotiation between PCA and SPA.....	169
Figure 7-7 Petri-net model for negotiation between PCA and SPA.....	170
Figure 7-8 Reachability graph for negotiations.....	171
Figure 7-9 Petri-net model for establishing connection.....	174
Figure 7-10 Reachability graph for establishing connection.....	175
Figure 7-11 Interaction diagram for a monitoring agent.....	178
Figure 7-12 Petri-net model for monitoring email.....	179
Figure 7-13 Reachability graph for monitoring email.....	180
Figure 7-14 Itinerary graph for upgrading a service.....	182
Figure 7-15 Petri-net model for upgrading a service.....	183
Figure 8-1 Converter hierarchy.....	188
Figure 8-2 Connector hierarchy.....	189
Figure 8-3 Device agent hierarchy.....	190
Figure 8-4 Setup Interface.....	190
Figure 8-5 Calendar Agent interface.....	191
Figure 8-6 Notify Agent interface.....	192
Figure 8-7 Relationships between agents and classes.....	193
Figure 8-8 Scenario for message handling.....	195

LIST OF TABLES

Table 3-1 Network management agent vendors.	62
Table 5-1 Satisfying user's requirements.	111
Table 5-2 Supporting multi-vendor market.	112
Table 5-3 Satisfying Service manageability requirements.	112
Table 5-4 Satisfying provider requirements.	113

Chapter 1

Introduction

Research is the art of seeing what everyone else has seen, and doing what no one else has done.

CHAPTER 1

INTRODUCTION

In this chapter, we give an overview of our research. The research motivations and approach are then discussed. We also present a summary of research objectives and contributions. We conclude this chapter by presenting the organization of the thesis.

1.1 PROBLEM OVERVIEW

The purpose of our research is to define a systematic process for the development of agent-based systems (static and mobile) based on existing object-oriented analysis and design methods. Then, we apply this process to develop an agent-oriented framework (AgenTINA) to enhance TINA (Telecommunications Information Networking Architecture)-compliant architectures.

It can be argued that agents are very well suited for the new world of distributed heterogeneous environments. The agent-oriented paradigm is a natural extension to the object-oriented paradigm, yet very little work has

been reported on analysis and design methods in the mobile agent community. After surveying a number of well-known object-oriented analysis and design methods, we have found that none provides an adequate model for the design and analysis of mobile agent systems. The initial goal of this research is to define a systematic process for the development of agent-based systems. Related work can be found in: [Og-92], [Con-92], [Rz-94], [Kgr-96]. However, they all deal with agents from an AI-oriented approach and none of them gives emphasis to mobility issues.

One of the most promising candidates for a future telecommunications system architecture is the “Telecommunications Information Networking Architecture” (TINA) developed by the international TINA consortium (TINA-C). Although there are possibilities for dynamic configuration and object migration, TINA is at its core still based on the traditional client/server concept and distributed objects standards. Configuration of a new service is relatively hard, even extending an existing service is not an easy task. Due to their autonomy, mobility, persistence and ability to communicate, agents appear to be a highly suited basis for dynamic telecommunications environments. They lend themselves easily to the task of representing service providers and users in an electronic service market, and seem almost predestined for this particular field. Therefore, the aim of the second part of our work is to develop an agent-oriented framework (AgenTINA) to enhance TINA-compliant architectures. Related work that deals with automating networks configurations can be found at: [Ten-97], [Yem-96], [Har-96], [Bha-96]. However, none of them deals with TINA-compliant architectures.

1.2 MOTIVATIONS

1.2.1 AGENT-BASED SOFTWARE PARADIGM

Agent-oriented paradigm is likely to be the next significant breakthrough in software development process. The evolution phases in the past two decades can be illustrated by Figure (1-1). The Ethernet era of client/server saw a file-centric application wave (the Netware wave) followed by database centric wave (the Oracle wave); TP monitors and GroupWare generated minor ripples. Active objects and components and Internet programming represent the next wave. We can argue that distributed objects and mobile agents combined with the Internet are essential for making the intergalactic client/server version.

Figure 1-1 Evolution waves.

Agents have the advantages of objects since they provide a uniform approach throughout the development life-cycle. The agent-oriented

paradigm is a natural extension to the object-oriented paradigm. Very little work has been reported on analysis and design methods in the agent community. After trying to apply object-oriented analysis/design methods to model agents, it was found that they provide a good starting point to model agent-based systems. However, some modifications and extensions are needed to provide an adequate model for the design and analysis of agent-based systems due to the differences in nature between agents and objects. These differences will be discussed in more detail in Chapter 4 (Agent-based Software Engineering). In fact, the motivation for the initial part of our work is the need for a systematic process for developing agent-based systems for real world applications.

1.2.2 FUTURE TELECOMMUNICATIONS ARCHITECTURES

Today's telecommunication service and management architectures are insufficient for the highly dynamic electronic service market of the future. It is obvious that telecommunications and information technologies will continue to evolve, merging in some respects (see Figure 1-2). This will create new possibilities for applied technologies and standardization. Main computing trends, such as object technology and distributed computing, have been indeed recognized as the basis for the future telecommunications applications. TINA-C (Telecommunications Information Network Architecture Consortium) is working on the TINA standardization, which challengingly incorporates (see Figure 1-2):

- Distributed telecommunications technology standardized in the ITU-T (International Telecommunications Union) ODP (Open Distributed Processing) recommendations,

- Telecommunications network management technology standardized in the ITU-T TMN (Telecommunications Management Network) recommendations,
- Object technology standardized in the OMG (Object Management Group) OMA (Object Management Architecture) standards,
- Broadband technology standardized in ITU-T B-ISDN (Broadband Integrated Services Digital Network) recommendations and incorporated with Broadband IN.

Broadband technologies are another major trend in telecommunications. With broadband technologies such as ATM (Asynchronous Transfer Mode), it will be possible to create advanced interactive telecommunications services which, up until now, have been impossible with present telecommunications networks because of insufficient bandwidth. It is expected that IN will incorporate ATM switching technology to form Broadband IN (BIN). A further, concurrent, trend in telecommunications networks is towards the separation of switching and call control technology.

Mobile communications have already revolutionized the way people communicate. The rapid increase in the number of subscribers to the GSM (Global System for Mobile communications) and other cellular systems all around the world has clearly indicated that mobility will be the basis for future personal communications. In addition, standards like UMTS (Universal Mobile Telecommunications System) and MBS (Mobile Broadband System) are incorporating B-ISDN with third generation mobile technologies and

fixed telecommunications networks to form seamless universal telecommunications networks (see Figure 1-2).

Figure 1-2 The evolution of telecommunications standards.

Despite the fact that TINA is one of the most promising future architectures, it is not flexible enough to handle services and components configuration in a plug-and-play fashion. TINA can be enhanced in many areas. Therefore, the

motivation for the second part of our work is to realize the practical applications of TINA. To do so we developed AgentTINA which is an enhancement for TINA using agents (static and mobile).

1.3 RESEARCH APPROACH

In this section, we discuss the steps we followed to conduct our research in order to achieve our goals. Figure (1-3) illustrates an overview of the research approach.

1.3.1 DEVELOPING AN AGENT-BASED ANALYSIS/DESIGN PROCESS

The following are the steps we followed to develop an agent-based software engineering process:

- Survey existing Object-oriented (OO) analysis/design methods to pick relevant modeling techniques.
- Survey agent-based systems and the nature of their architectures to discover the differences between agents and objects.
- Develop an agent-based analysis/design process. The process includes guidelines for finding agents and enough tools to model the static and dynamic behavior of agents.
- Use the process to develop illustrative agent-based applications to prove that it covers the whole life-cycle. Illustrative applications include electronic shopping assistants and schedule organizers.
- Update the process if needed, try to generalize the concepts by developing agent-based applications.

1.3.2 APPLYING THE METHOD TO DEVELOP AGENTINA

The following are the steps we followed to develop an agent-based framework that enhances TINA (AgentTINA):

- Analyze the TINA architecture, more specifically its service architecture.
- Prepare analysis and design of AgentTINA using the software engineering process that we proposed in the initial part of our thesis.
- Verify that the framework (AgentTINA) is safe and free of deadlocks using Petri-Nets.
- Use AgentTINA to model a unified messaging application.

Figure 1-3 Research approach.

1.4 OBJECTIVES AND SUMMARY OF FEATURES AND CONTRIBUTIONS

1.4.1 MAIN FEATURES OF AGENT-BASED SOFTWARE ENGINEERING PROCESS

The first part of our research is dedicated to develop an analysis/design process for agent-based development. The process includes guidelines to find and model agents (mobile and static). The contribution of this part is to introduce software-engineering practices in the new growing field of mobile agents.

The main features of the agents-based software engineering process we propose are:

- The process has step-by-step guidelines from analysis to design to verification.
- The process contains notations to model static and mobile agents.
- The process contains notations to model both static and dynamic behaviour of agent-based systems.
- The process puts emphasis on mobility and security issues, since the two issues are not adequately handled by existing OO methods.
- Petri-nets are used to verify the reliability of the design.

1.4.2 MAIN FEATURES OF AGENTINA

The second part of our research is to prepare an Analysis/design for an agent-oriented framework (AgentTINA). AgentTINA's main goal is to improve and realize the TINA architecture by adding a set of agents and classes. The main goal of AgentTINA is to make TINA more adaptable to the nature of the

multi-vendor, heterogeneous telecommunications market. AgenTINA facilitates service configuration, shopping and connection management.

The main features of AgenTINA are:

- Introducing new services, users and devices is done with minimal human effort utilizing agents.
- AgenTINA introduces a set of agents that negotiate for services on behalf of users.
- Shopping for services and devices is provided using mobile agents.
- AgenTINA contains a set of utilities agents that can be used in a variety of applications. A Text-to-speech agent is an example of utilities agents.
- AgenTINA is TINA-compliant.

1.5 THESIS ORGANIZATION

The thesis is divided into two main parts: part I includes a literature survey and background and part II includes our approach and solutions for the problems. Figure (1-4) shows the relationships between thesis chapters and shows which chapters are related.

Part I, Chapters 2 and 3 (Background and related work): In these chapters, TINA and agents' technology will be briefly introduced. Moreover, a survey of the closely related research in both fields will be included.

Part II, Chapter 4 (Software engineering for agent-based systems): In this chapter, the research methodology will be introduced including the steps to model and develop agent-based systems.

Figure 1-4 Relationships between thesis chapters.

Part II, Chapter 5 (Requirements Specifications): In this chapter, the requirement specifications of AgenTINA are introduced.

Part II, Chapter 6 (AgenTINA Architecture): In this chapter, AgenTINA framework for TINA-based services and components will be described. Modeling will be done using the process described in Chapter 4.

Part II, Chapter 7 (Verification of AgentTINA design using petri-nets): In this chapter, the technique for verification described in Chapter 4 is used to verify freedom of deadlocks of the design.

Part II, Chapter 8 (Applications): In this chapter, AgentTINA will be used to develop a unified messaging application.

Part II, Chapter 9 (Conclusion): In this chapter, we conclude our thesis by highlighting the unique features of our research. Finally, some suggestions for future research are proposed.

Appendix A contains a sample of the code generated by AgentTINA demo for the Microsoft Agent.

Appendix B contains Petri-Net analysis of the scenarios presented in Chapter 7.

Abbreviations list contains the abbreviations used in the thesis with their meanings.

Chapter 2

Agents

Literature Survey & Related Research

To succeed, it is necessary to accept the world as it is and rise above it.

Micahel Korda

CHAPTER 2

AGENTS: LITERATURE SURVEY AND RELATED RESEARCH

In this Chapter we introduce the notion of agents and their applications. We also discuss various types of agents. Then we introduce a brief discussion of existing agent-based development environments. The current agent standardization efforts are then discussed. We conclude our discussion by a literature survey of current agent-based software engineering research.

2.1 BACKGROUND

The term “Intelligent Agent (IA)” has become a buzzword recently. Today, agent technology is utilized in a wide range of application areas, such as intelligent user interfaces, distributed problem solving, smart messaging and network management. IA ideas and technologies have been influenced by a wide variety of disciplines and practices. The origin of IAs is in the field of artificial intelligence, in particular, distributed artificial intelligence. Agents were based on the idea of creating “objects that think”, i.e. intelligent autonomous software entities that behave in accord with self-contained

intelligence. Today “Agent Oriented Technology (AOT)” can be considered as an interdisciplinary research field, influenced by object-oriented programming, distributed systems concepts, telecommunications, knowledge-based systems, decision theory and organization theory [Mag-95].

2.1.1 DEFINITIONS

There is no single unifying definition of “agenthood” in the literature. There are, however, several widely accepted concepts which characterize agent systems. Webster’s dictionary defines an agent as an entity which acts for or in the place of another by authority from him, either as a representative...or as an instrument by which a guiding intelligence achieves a result.

According to Jennings and Wooldridge [Ina-95] three classes of agents can be identified. The first type of agents is **gopher agents** which can execute straightforward tasks based on pre-specified rules. The second type is **service-performing agents** which execute a well-defined high-level task at the request of the user. The third type is **predictive/proactive agents** which volunteer information or services to a user without being asked.

Hyacinth and Nwana [Hwa-96] define an agent as a component of software and/or hardware which is capable of acting in order to accomplish tasks on behalf of its user.

Pattie Maes of the software agents research group at MIT has another definition. For Pattie Maes, an agent is a computational system that inhabits a complex, dynamic environment. The agent can sense, and act on, its

environment, and has a set of goals or motivations that it tries to achieve through these actions [Maes-94].

FIPA (Foundation for Intelligent Physical Agents) [FIPA-99] uses a strawman definition of agent for all their specifications. They define an agent as:

"An entity that resides in environments where it interprets **sensor** data that reflect events in the environment and executes **motor** commands that produce effects in the environment. An agent can be purely software or hardware. In the latter case, a considerable amount of software is needed to make the hardware an agent."

For AI researchers, the term "agent" has a stronger and more specific meaning. These researchers generally mean an agent to be a computer system that is either conceptualized or implemented using concepts that are more usually applied to humans. For example, it is common in AI to characterize an agent using human notions, such as knowledge, belief, intention, emotion and obligation.

2.1.1.1 Fundamental Attributes of Agents

The previous definitions of software agents imply that the agent possesses the following minimal characteristics:

- *Persistence:* Agents maintain a consistent internal state over time, which is not changed.

- *Autonomy*: Agents exercise exclusive control over their internal state and behavior.
- *Reactivity*: Agents perceive and react to changes in their environment.
- *Ability to communicate*: Agents are able to exchange information with other entities (agents, humans, objects and their environment).
- *Ability to cooperate*: Agents can interact with other agents and/or humans.

2.1.1.2 Other Attributes of Agents

In addition to the fundamental attributes, agents may possess the following characteristics:

- *Initiative and data-directed execution*: An agent exhibits goal-directed, opportunistic behavior and does not simply react to its environment.
- *Mobility*: Agents can possess the ability to move from one location to another while preserving their internal state.
- *Reasoning*: An agent can possess the ability to infer and extrapolate based on current knowledge and experiences - in a rational, reproducible way.
- *Ability to Plan*: An agent can synthesize and choose between different courses of action intended to achieve its goals.

- *Learning and Adaptation:* An agent may be able to accumulate knowledge based on past experience, and consequently modify its behavior in response to new situations.

In addition, there are many other auxiliary attributes.

2.1.2 TYPES OF AGENTS

In this section we present the different types of agents that can be identified by combining some of the attributes described in the previous section.

- *Autonomous agents* inhabit some complex, dynamic environments, sense and act autonomously in this environment and by doing so realize a set of goals or tasks [FIPA-99].
- *Entertainment agents* serve entertaining purposes (e.g. games, film/video production), rather than strictly utilitarian ones.
- *Information agents* have access to potentially many information sources. Moreover, they are able to collate and manipulate information obtained from these sources to answer queries posed by users and/or agents [FIPA-99]. Some people refer to these as Internet agents as such agents may roam about the Internet in order to collect information.
- *Intelligent Agents* carry out some set of operations on behalf of a user or another program with some degree of independence [FIPA-99].
- *Interface agents* support and provide assistance, typically to a user. The agent observes and monitors actions taken by the user in the interface,

learns from it, and suggests better ways of doing the task [Nwa-96]. Interface agents learn typically to better assist in four different ways. First, they observe and imitate the user. Second, they learn through receiving positive or negative feedback from the user. A third possible way is by receiving explicit instructions from the user. Fourth they ask other agents for advice.

- *Collaborative agents* emphasize autonomy and cooperation with other agents in order to perform tasks for their owners. Their key attributes include autonomy, social ability, responsiveness and proactiveness. In order to have a coordinated setup of collaborative agents; they may have to negotiate in order to reach mutually acceptable agreements in some matters [Jen-95].
- *Mobile agents* are computational software processes capable of roaming wide-area networks, such as the WWW, interacting with foreign hosts, gathering information on behalf of its owner and coming back home after having performed the duties set by its user [Nwa-96]. The attribute of mobility has introduced the concept of remote programming [Whi-94] where agents interact as peers and each agent can act as both a client and server. Some of the most important issues to be dealt with while implementing mobile agents are that of security, secrecy, transport mechanism and authentication. The system has to be protected against such hazards as viruses and endless loops that consume all the CPU cycles. Some notable mobile agent implementation platforms are: AgentTcl from Dartmouth University, IBM Aglets Workbench [Aglets-99], Voyager [Voyager-99] from Objectspace Inc., and Concordia from

Mitsubishi Electric [Conordia-99]. Mobile agents will be discussed in more details in section 2.1.5.

- *Reactive agents* represent a special category of agents. They do not possess internal, symbolic models of their environments; instead they act/respond in a stimulus-response manner to the present state of environment in which they are embedded [Mul-94]. The important point of note with reactive agents is that these are relatively simple and they interact with other agents in simple ways. Nevertheless, complex patterns of behavior can emerge from these interactions when an ensemble of agents is viewed globally. Pattie Maes [Maes-91] highlights three key ideas which underpin reactive agents. She claims that the dynamics of interaction between these simple unintelligent agents should lead to emergent complexity. She adds that a reactive agent is usually viewed as a collection of modules that operate autonomously and are responsible for specific tasks (e.g. sensing, motor control, computations etc.). Communication between these modules is usually minimal and low-level in nature. Maes concludes that a reactive agent tends to function on representations that are close to raw sensor data, in contrast to high-level representations used by other types of agents. Reactive agents often give rise to NMAS (Natural Multi-Agent Systems) which are often modeled as artificial life systems. NMAS are colonies of agents which demonstrate cooperative behavior to produce integrated system behavior.
- *Hybrid agents* combine two or more agent philosophies within a single agent to maximize the strengths and minimize the deficiencies of the

most relevant technique for a particular purpose. These philosophies may include a mobile philosophy, an interface agent philosophy, collaborative agent philosophy, etc. One interesting implementation of such a hybrid approach is InterRRap Agent Architecture which has been developed by the German Artificial Intelligence Research Institute (DFKI) [Int-94].

2.1.3 AGENTS COMMUNICATIONS

Communication enables the agents in a multi-agent system to exchange information. On the basis of this exchange, agents coordinate their actions and cooperate with each other. This raises the important question of what communication protocols and mechanisms are conducive to enhance collaboration between communicating agents. In a multi-agent system, several ways have been proposed for agents to exchange information with each other. Agents can directly exchange messages, or they can organize themselves into a federated system and communicate through special facilitator agents [Gen-94], or they can broadcast the messages. Another popular approach used to enable agents to intercommunicate is through a shared data repository (called a blackboard) in which information can be posted and retrieved [Cha-96].

Directed communication involves establishing direct physical links with other agents using a protocol such as TCP/IP which promises safe arrival of message packets by implementing end-to-end acknowledgments. Physical link implies that the agent has to be aware of the other agent in the system. Agent addresses may be obtained either as part of received broadcast messages from other agents or from a centralized object which is like a directory service where all the agents joining the system register. A sender agent can access the

addresses of the receiver agents by looking at this centralized object. Registering is like a start-up process for the directory to learn about all the agents in the system. The FIPA 1997 Specification, V 1.0 [FIPA-99], specifies that any multi-agent system compliant with their specification should have an Agent Directory, which contains information about all the agents in a particular environment and facilitates identifying and accessing of agents. A directed communication mechanism is used in most existing agent building languages and platforms. In situations where an agent is engaged in a dialog with a particular agent, and knows exactly who to send the message to, directed communication makes sense.

Figure 2-1 Federated systems

When the number of agents in a system becomes very large (e.g. in a setting like Internet) the cost and processing involved in directed communication is prohibitive. A popular alternative to directed communication that eliminates these difficulties is to organize the set of agents into a federated system. Figure (2-1) illustrates the structure of such a federated system. As suggested in the diagram, agents do not directly communicate with each other. Instead, they communicate through special facilitator (mediator) agents. Here a set of agents has a facilitator who is kept informed about their individual needs and abilities. Agents can also send application-level information and requests to these facilitators and accept application-level information and requests in return. Facilitators use the information provided by these agents to transform these application-level messages and route them to appropriate places. In effect, the agents form a "federation" in which they surrender their autonomy to their facilitators and their facilitators take the responsibility of fulfilling their needs. FIPA 1997 Specification V 1.0 [FIPA-99] defines a specialized Domain Facilitator agent for each domain whose job is to maintain an Agent Directory for that domain and facilitate communication between agents of that domain.

There are two choices in situations where a message has to be communicated to all the agents in the environment, or the sender agent does not know who the recipient will be (like when it announces a task and has to choose from all possible agents that can perform that task). The agent can either physically broadcast the message to all the agents in the system, or it can maintain individual communication links with all the agents in the system and send each one of them a directed message (using the TCP/IP protocol).

When the message length is substantial, and there are a large number of agents in the system, then the network bandwidth used to transmit the message is significant. Maintaining individual links implies that multiple copies of the same message have to be sent to the receiving agents. Broadcasting, on the other hand, prevents network overloading by doing away with the need of making multiple copies of the same message and transmitting them to different agents [Til-96]. It helps in implementing a totally autonomous and flexible multi-agent system as agents can leave or join a system without needing to inform anyone, provided they have completed all the tasks they were engaged in. It is not uncommon to implement a hybrid approach by using broadcast communication to find the identities and addresses of agents in the community, and then use that information to engage in directed communication. Two main popular approaches in broadcast communication are: *the contract-net approach* and *the specification-sharing approach*. *The contract-net approach* was proposed by Davis & Smith [Smi-80]. In the contract-net approach to interoperation, agents in need for service distribute requests for proposals (broadcast messages) to other agents. The recipient of these messages evaluates these requests and, if capable, submit bids (directed message) to originating agents. The originators use these bids to decide which agent gets the contract (directed message) for the broadcasted request. In *the specification-sharing approach*, agents broadcast their capabilities and needs and other agents use this information to coordinate their needs and actions. Broadcast communication over the web is popularly referred to as webcasting [Fri-94]. The strength of webcasting lies in the fact that the Internet can be used to

transmit highly complex video, audio and other multimedia signals to any number of users all over the world.

In AI, the blackboard is a commonly used model of shared memory [Cha-87]. It is a repository on which agents write messages, post partial results, and obtain information. It is usually partitioned into several levels of abstraction appropriate for the problem at hand, and agents working at a particular level of abstraction have access to the corresponding blackboard level along with the adjacent levels. In that way, data that has been synthesized at any level can be communicated to higher levels, while higher-level goals can be filtered down to drive the expectations of lower level agents.

2.1.4 MOBILE AGENTS

2.1.4.1 Approach

Mobile agents are based on the Remote Programming (RP) paradigm. The RP paradigm views computer-to-computer communication as enabling one computer not only to call procedures in another, but also to supply the procedures to be performed. The main advantage is that the computer must be connected to the network only long enough to send the agent on its way, and later, to welcome it home (not while the agent carries out its assignment). Thus RP lets computers connected only occasionally do things that would be impractical with traditional Remote Procedure Call (RPC). Figure (2-2) shows the difference between a mobile agent approach and the traditional remote procedure calls.

Figure 2-2 Mobile Agents Approach

2.1.4.2 What is a mobile agent?

A mobile agent is not bound to the system where it begins execution. It has the unique ability to transport itself from one system in a network to another. The ability to travel lets a mobile agent move to a system that contains an object with which the agent wants to interact, then take advantage of being in the same host or network as the object [OMG-99].

Along with mobility, agents have the following unique and important computational characteristics:

- **Object-passing:** When a mobile agent moves, the whole object is passed; that is, its code, data, execution state, and travel itinerary are passed together.
- **Autonomous:** The mobile agent contains sufficient information to decide what to do, where to go, and when to go.

- ***Asynchronous:*** The mobile agent has its own thread of execution and can execute asynchronously.
- ***Local interaction:*** The mobile agent interacts with other mobile agents or stationary objects locally. If needed, it can dispatch messenger agents or surrogate agents, which are all mobile agents, to facilitate remote interaction.
- ***Disconnected operation:*** The mobile agent can perform its tasks whether the network connection is open or closed. If the network connection is closed and it needs to move, it can wait until the connection is reopened.
- ***Parallel execution:*** More than one mobile agent can be dispatched to different sites to perform tasks in parallel.

There are many technical advantages of mobile agents, and there is no single alternative to all of the functionality they provide. In addition to supporting existing network services, mobile agents are also likely to make possible new services and thus new businesses.

2.1.4.3 Conceptual architecture for mobile agents

A client computer consists of an application environment, for example, OS/2 or Microsoft Windows, which contains one or more applications for interaction with a remote server. These applications may include information searching and retrieval or transaction front-ends. These applications are bound to an execution environment for mobile agents. Via the Application Programming Interface (APIs), the application can pass parameters to various

classes (not necessarily object-oriented classes) of agent programs, and likewise the agent programs can return parameters to the application programs. In principle, there may be no application programs, the agent programs can themselves perform a presentation on the client device's user interface and collect information directly from a keyboard or other input device; in this case the agent programs must bind to the user interface libraries of the client device. The agent execution environment will also need to bind to various operating systems functions, such as the memory manager, the timer, the file system and so forth. In particular the agent execution environment needs to bind to the message transport service in order to send and receive mobile agents via the communication infrastructure.

When an application needs to perform a transaction, it will assemble the required information and then pass this via the API into the agent execution environment. This will initiate the execution of an instance of a particular class of agent as a process within the agent execution environment. This may correspond to an operating system process or an operating system thread or it may be managed by a threads package within the agent execution environment. The agent execution environment will have access to many different agent programs, which provide different services to the client applications. For example, one may act as a delivery agent for electronic mail, another may deliver a database retrieval request to a server, submit the request and return the result to the client application, yet another may navigate its way among multiple servers, asking each in return for updates on a particular topic.

The agent program may be built from procedural components or from classes of objects. In either case the agent has bindings to functions within the agent execution environment, including functions imported from the operating system, the application or other sub-systems, as well as other agent programs.

The program may be executed in either machine language or an interpreted (virtual machine) language like Java. In order to support heterogeneity, it is often preferable to express the agent in an interpreted language. There is a performance penalty for this, but since most of the agent processing is done not in the agent itself, but rather in the functions to which it binds, this may be acceptable. Interpreted languages also have the advantage of late binding; this enables the agent to contain references to functions or classes not present on the system at which it is launched, but which are available at its destination. Interpreted languages are also easier to render secure than machine language, since the language developer explicitly controls what system resources are accessible.

The information assembled by the application is accepted by the agent as part of its initialization and at a certain point in its execution, the agent will execute an instruction, which has the following effects:

- Either the current agent process is suspended in the agent execution environment, or a new agent daughter process is created.
- The suspended process or the new process, including its process state, stack, heap and all external references is collected and processed into a message expressed in a machine independent form. This step is facilitated if the agent is built from object classes and in an interpreted

language. In particular, if it is known that the identical classes are resident at the destination, the agent may be reduced to object references, instance data and process state data. If the agent is expressed in an interpreted language, the state data is captured on the stack and there is no need to save registers.

- The message may be addressed explicitly to a final destination, or it may be directed to a post office function, which can perform address resolution, or to immediate destinations, which route the agent on the basis of its content.
- The message is handed to message sub-system and routed directly or indirectly to the destination server, where it is delivered by the server's message subsystem to the agent execution environment.
- In the agent execution environment the received message is reconstituted into the executable, and the process or thread is dispatched.
- Execution continues at the next instruction in the agent program.

During execution at the server, the agent passes the information it received from the client application to server application functions and perhaps receives other information in return. At the completion of this stage, it may perform one or more of the following functions:

- It may terminate its execution.
- It may simply suspend at the server, waiting for some event to be delivered. We then say that it has become a 'resident' agent at the

server. Resident agents may become permanently resident if there is some repeated service desired by the user.

- It may repeat the migration process, either by forking a new daughter process or by suspending and migrating itself. This second migration may return the agent to its originating client or it might continue to another server or another client.

Figure 2-3 Conceptual model for mobile agent computing

The agent should be able to perform a recovery action and visit another server if the required service is not available. Figure (2-3) shows the conceptual model for mobile agent computing.

2.1.4.4 Problems introduced due to mobility

The utilization of mobile agents appears to offer certain advantages for client-server computing, but it also raises some difficult issues:

- *Efficiency:* Does the agent execution environment require significant computational resource? Does the transmission of a transaction or other request via a mobile agent result in more, or less, network traffic than alternate methods?
- *Flexibility:* Can the use of mobile agents provide a more flexible and robust method of communication than other methods? Is it likely that agent execution environments would be rapidly deployed on network servers?
- *Security:* Is there a useful compromise between the desire to isolate the agent execution environment from the system and application functions and the need to provide access in order to accomplish the users' tasks? Is it possible to define a language that provides sufficient expression for client-server interaction, while being sufficiently restricted that the server and other agents can not be compromised?

2.1.4.5 Alternatives to mobile agents

For completeness, alternatives to mobile agents for client-server interactions are discussed briefly in this section. The dominant methods are *messaging*, *simple datagrams*, *sockets*, *remote procedure call*, and *conversations*. The

primary distinction among these is between asynchronous protocols, for example, messaging, and synchronous protocols, for example, RPC. Mobile agents employ messaging frameworks for transport, and hence are asynchronous.

2.1.5 EXAMPLES OF AGENTS

There are numerous agents on the Internet that provide specialized services to users. These range from shopping agents which find the best price on a CD to matchmaking agents that notify users when a suitable candidate joins a matchmaking service. In this section, several examples of existing agents will be introduced briefly.

- *BargainFinder* is an experimental virtual shopping agent for the Web developed by Andersen Consulting. For instance, *BargainFinder* uses a parallel query architecture similar to meta search engines in order to query pricing and availability of user-specified music CDs [Cag-97].
- *Julia* is a conversational agent for Multi-User Dungeons (MUDs). *Julia* is a C program that incorporates a great number of tricks which plays an imitation game by answering questions with questions [Cag-97].
- *NewsHound* is a personalized newspaper that searches the stories in the San Jose Mercury News as well as several other newspapers to find the articles that match a user's profile. Selected articles matching the user's profile are sent to the user's Internet address by email [Cag-97].

2.1.6 AGENT DEVELOPMENT ENVIRONMENTS

Most network protocols address the issue of heterogeneous hardware platforms, including byte order of multi-byte integer representations. But an emerging technique for distributing applications involves “mobile code”—code that can be transmitted across the network and executed on the other end. First we will talk very briefly about Java since we will categorize mobile agents systems to be Java-based and Java-independent.

2.1.6.1 *Java*

Java introduces a new model of client/server interaction in the Web. It lets programmers write small component-like programs called applets that can be downloaded into a browser that is Java-compatible. Applets allow programmers to distribute executable content across the Web along with data. Java owes its popularity to its portability; latest version of an application can be accessed by simply clicking on the mouse. Applications can be distributed to millions of clients by putting them on a Web server.

2.1.6.2 *Aglets from IBM*

IBM Japan is developing a framework for Java “Agent Applets.” Some of their work has been submitted to the Object Management Group (OMG) for consideration in regard to OMG's request for a Mobile Agent Facility (MAF). Aglets Workbench is a visual environment for building network-based applications that use mobile agents to search for, access and manage corporate data and other information. IBM Aglets are mobile Java programs, which may travel and execute in specialized nodes in the network. The Java Aglet Application Programming Interface of the framework defines the methods necessary for Aglet creation, message handling in the network and

initialization, dispatching, retraction, deactivation/activation, cloning and disposing of the Aglet. Aglets workbench includes an Agent Web Launcher named Fiji and a Visual Agent Manager named Tahiti. Fiji is a Java applet based on the Aglets Framework and therefore capable of creating an Aglet and retracting an existing Aglet into a client's web browser. Tahiti uses a unique graphical user interface to monitor and control Aglets executing on a given computer. It also implements a configurable security manager that provides a fairly high degree of security for the hosting computer system and its owner. Aglets Workbench is a very versatile tool for creating secure mobile agent-based applications; however it does not deal with the important issue of implementing coordination, cooperation and coherence in agent-based applications. IBM Aglets can only engage in directed communication as they use the TCP/IP protocol.

Figure 2-4 Mobile code, the Java style

2.1.6.3 Concordia from Mitsubishi

Mitsubishi Electric Information Technology Center of America has developed a Java-based framework for development and management of network-efficient mobile agent applications for accessing information anytime, anywhere, and on any device. Concordia offers a flexible scheme for dynamic invocation of arbitrary method entry points within a common agent application. It provides support for agent persistence and recovery and guarantees the transmission of agents across a network. Concordia has also been designed to provide for fairly complete security coverage from the outset. Within Concordia, an agent's travel plans are specified by its Itinerary. The Itinerary is a completely separate data structure from the agent itself. Concordia provides two forms of asynchronous distributed events: selected events and group-oriented events. The event selection paradigm enables agents to define the types of events they wish to receive. By contrast, group-oriented events are distributed to a collection of agents (known as an event group) without any selection. Though Concordia provides a useful set of services for implementing agent mobility, security, persistence and transmission, it does not provide any methodology to specify how agents in a multi-agent system coordinate, cooperate and negotiate to bring about a coherent solution. Emphasis here is on the communication aspect in an agent-based application. Moreover, the fact that the agent Itinerary is outside the agent implies that where the agent travels is maintained in a separate logical location than what the agent does. This results in Concordia agents not being totally autonomous. Like Aglets, Concordia agents can communicate only through the TCP/IP protocol.

2.1.6.4 Voyager from ObjectSpace

Voyager is a Java-based agent-enhanced Object Request Broker (ORB) developed by ObjectSpace Inc. It allows Java programmers to create quickly and easily sophisticated network applications using both traditional and agent-enhanced distributed programming techniques. It provides for the creation of both autonomous mobile agents and objects. Voyager agents roam a network and continue to execute as they move. Voyager can remotely construct and communicate with any Java class, even third party libraries, without source. It allows seamless support for object mobility. Once created, any serializable object can be moved to a new location, even while the object is receiving messages. Messages sent to the old location are automatically forwarded to the new location [Voyager-99].

Voyager is a very efficient tool for constructing agent-based distributed applications, but it suffers from the same drawbacks as Aglets and Concordia. It does not provide any classes for defining the social behavior of agents. Voyager does not support broadcast communication and speech-act messaging. It does not pay any specific attention to providing security.

2.1.6.5 Microsoft Agents

The Microsoft Agent is a set of programmable software services that support the presentation of interactive animated characters. Developers can use characters as interactive assistants to introduce, guide, entertain users. Microsoft Agent includes an ActiveX control that makes its services accessible to programming languages that support ActiveX, including Web scripting languages such as Visual Basic Script [Msa-99].

2.1.7 AGENTS STANDARDS BODIES

The main bodies working in the areas that affect computer agents are as follows:

- *Worldwide Web Consortium (W3C)* was founded in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. It is also interesting to note that virtually all recognized standards in use on the Web come from this body.
- *The Object Management Group (OMG)* was founded in May 1989. OMG was formed to create a component-based software marketplace by hastening the introduction of standardized object software. The organization's charter includes the establishment of industry guidelines and detailed object management specifications to provide a common framework for application development.
- *Foundation for Intelligent Physical Agents (FIPA)* was founded in 1996. FIPA was formed to create set of standards for agents. Currently, FIPA released a set of specifications for several agent applications such as: personal travel assistants, human-agent communication and network management.

2.1.8 FUTURE OF MOBILE AGENTS

The individual advantages of mobile agents can be summarized as follows:

- High bandwidth remote interaction
- Support for disconnected operations

- Support for weak clients
- Ease of distributing individual service clients
- Robust remote interaction

It can be argued that while none of the individual advantages of mobile agents given above is overwhelmingly strong, the aggregate advantage of mobile agents is overwhelmingly strong. Mobile agents can provide a pervasive, open, generalized framework for the development and personalization of network services. While alternatives to mobile agents can be advanced for each of the individual advantages, there is no single alternative to all of the functionality support by a mobile agent framework. In addition to providing an efficient support for existing services, a mobile agent framework also enables new, derivative network services and hence new businesses. Mobile agents are expected to appeal strongly to the Internet community, since they can provide an effective means for dealing with problems of finding services and information and since they empower the individual user.

Agents will need a common software framework within which they can easily and safely migrate from one computer system to the next. Emerging object oriented software management technologies like the CORBA should provide the ability to manage object migration. Once free to roam under ethical guidelines, Intelligent agents will represent the most pervasive popularization of the computer to date: the computer as an appliance [Aub-95]. New methods, languages, standards will continue to appear in the market in the coming years because the field is still developing and it has a wide range of applications like telecommunications, electronic

commerce, information gathering, network management and active messaging.

2.2 RELATED RESEARCH

So far, very little work has been reported in the area of the systematic and methodological analysis and design approaches for building mobile agent systems. Most of the current research deals with modeling static multi-agent systems that are briefly discussed here.

An approach that was inspired by the KADS-model is mainly concerned with knowledge acquisition aspects [OG-92]. No hints are given on how to come to an agent-oriented model of the domain.

Some general criteria for a modular system design from OO techniques as defined in [Mey-88] were transferred to multi-agent system design by [OW-92].

The project CONSENSUS [CON-92] proposed a very high-level two step approach. The first step being a 'normal' requirement analysis as done for any software system. The second step is the design of a multi-agent system. However, again no instructions were given as to how this may be done.

The approach described in [Dor-93] discusses the decisions that have to be made when developing a multi-agent system, and in this respect is closer to our approach. The decisions concern the definition of the agents, the description of their beliefs and capabilities and finally the specification of generic interactions and communication contents. However, no guidelines are

provided as how appropriate structures and models may be developed to facilitate making “good” decisions.

Recent work of Kinny and his colleagues [KGR-96] starts from a specific OO technique, namely OMT (Object Modeling Technique); they propose a methodology and modeling approach for multi-agent systems. They distinguish between an external and an internal model. The external model consists of an agent model and an interaction model, and is independent of chosen agent architecture. The internal model is specific to the BDI (Belief-Desire-Intention) agent architecture and describes the agents by a plan model, a goal model and a belief model.

In the work of Rosenschein et al., [RZ-94], the main focus is on the cooperation and interaction aspects of multi-agent systems. They describe the definition of a negotiation process (as a special form of cooperation), as a two-step task: the negotiation process (as seen from the outside) and the negotiation strategy (for each agent). These instructions can be seen as a special form of building the cooperation model.

Recent research at Carleton University utilizes use case maps to handle the modeling of static agents [Buh-98]. The approach is suitable for high level design and does not handle mobile agents and their related issues.

Wooldridge proposes a general approach for specifications of agent-based systems and how the specifications can be refined or transformed into efficient implementation [Ow-92].

2.3 SUMMARY

In this Chapter, we introduce a brief overview of agents and their applications since we propose developing an analysis/design process to model agents in chapter 4. We give a detailed review of mobile agents and their applications. We also present related research projects that aim to handle modeling agents in general.

We can argue that most of the current research handles agents from an AI-oriented view. Also, current research is more oriented towards static agents, nothing much has been reported on modeling mobile agents.

Chapter 3

IN & TINA

Literature Survey & Related Research

I start where the last men left off

Thomas Edison

CHAPTER 3

INTELLIGENT NETWORKS AND TINA: LITERATURE SURVEY AND RELATED RESEARCH

In this Chapter, we introduce a brief background discussion of Intelligent Networks (IN) and The Telecommunications Information Networking Architecture (TINA). We also introduce a literature survey of related research to our thesis at the end of this chapter.

3.1 BACKGROUND

3.1.1 INTELLIGENT NETWORKS (IN)

Intelligent networks emanated from the first introduction of the 800 service (freephone) in the United States in the 80s. The term "intelligent networks" was introduced by BellCore around 1984 and initiated a new era in telecommunications. At that time, it became clear that centralized intelligence for controlling services nationwide could provide extensive benefits that it was worthwhile to build a separate node Service Control Point (SCP) exclusively for that.

3.1.1.1 What is IN?

Intelligent Network (IN) is a distributed architecture that allows telecom companies to create and customize services. IN aims to ease the introduction of new services based on more flexibility and new capabilities. IN architecture promotes a rapid service introduction through standard interfaces. Clearly, these interfaces should reduce the interval of time to support new services.

The IN Conceptual Model (INCM) was defined by International Telecommunications Union-Telecommunications (ITU-T) as a methodology for specifying the IN architecture. It is not an architecture in itself but it is a framework for the design and the description of the IN. The INCM consists of four planes (Figure 3-1). Each plane represents a different abstract view of the capabilities provided by an IN-structured network. The planes represent the service view, the global functional view, the distributed functional view and the physical view.

- *The service plane* gives an abstract view to the end-users on the services provided by an IN-structured network. A service can be decomposed into one to more service features.
- *The global functional plane* defines the Service Independent Building blocks (SIBs) with the following principle: the network is viewed as a single virtual machine. It turns out that the distribution is transparent at that level. A Service Feature (SF) is built of several SIBs (Figure 3-2). SIBs are used to build the Global service Logic (GSL). A service may be composed of one or several GSLs. A GSL is a series of SIBs starting with a Point Of Initiation (POI) and finishing on a Point Of

Return (POR). It is presently assumed that GSL for service processing has no parallelism. A specific GSL is performed through the BCP (Basic Call Processing) that provides all processing capabilities. At that time 14 SIBs have been defined by ITU-T. This number is insufficient and has to be enlarged by new capabilities.

Figure 3-1 The intelligent network conceptual model

- *The distributed functional plane* models a distributed view of an IN-structured network. This view consists of Functional Entities (FEs). A

FE is a unique group of functions in a single location and a subset of the total set of functions required to provide the service feature performed by the SIB. A group of centralized or distributed FEs is mapped onto a SIB, then, a SIB is realized by the interaction between corresponding distributed FEs. The advantage of this representation is in allowing reusable FEs, i.e., an FE can be used by several SIBs.

Figure 3-2 The global service logic.

- The physical plane identifies the different Physical Entities (PE). A Physical Entity consists of one or more Functional Entities. One or several FEs may be mapped onto the same PE, but a FE can not be split between two PEs. This plane takes into account physical considerations, optimization aspects and protocols.

3.1.1.2 IN Elements

Figure 3-3 gives an overview of the IN elements according to[Amb-89]:

- Service *user* takes advantage of the service by, for example, dialing a set of digits.
- Service *subscriber* provides the user's end service via the network operator's service access.
- *Network operator* provides the control logic and network (service) enabling the service user and subscriber to do the business. The service subscriber and network operator can be the same entity.
- *Network product supplier* supplies products enabling the network operator to provide service control.

Figure 3-3 IN elements.

3.1.1.3 The attractions of IN

The strong interest among fixed and mobile telecom network operators in Intelligent Networks stems from the potential of IN techniques to enhance the range, quality and flexibility of the services that can be offered to subscribers, and the speed with which new services can be developed and introduced. With IN, network operators can create new business opportunities by offering new services that are popular among business and residential subscribers. These services can be used as a way of increasing the network operator's market share.

3.1.2 TINA

3.1.2.1 What is TINA?

The Telecommunications Information Networking Architecture (TINA) Consortium is an international collaboration aiming at defining and validating an open architecture for telecommunications systems for the broadband, multi-media, and information era.

The architecture is based on distributed computing, object orientation, and other concepts and standards from the telecommunications and computing industries. The TINA architecture is very broad in its scope and deep in its detail. The TINA architecture provides a set of concepts and principles to be applied in the specification, design, implementation, deployment, execution, and operation of software for telecommunication systems.

The range of services provided by TINA systems to customers is potentially wide, and includes voice-based services, interactive multi-media services, information services and management services.

In addition to the obvious questions regarding network prerequisites and the evolving platform model serving as a framework for the architecture, the researchers are also addressing services, resource management, distributed processing platforms, and other key issues. Development of these architecture standards is not just an academic exercise. In fact, the consortium plans to follow through with field trials based on real applications to verify the appropriateness of the specifications.

TINA-C has now been joined by leading carriers and communications equipment and computer vendors from 32 countries around the globe. The technical work of the consortium is pursued on a volunteer basis by a core team of researchers on temporary assignment from among the consortium members, and is provisionally being conducted in facilities provided by Bellcore in the U.S. The Consortium will have fulfilled its goal in 2000 with large-scale deployments of TINA-conforming products and TINA-based services.

3.1.2.2 Basic Definitions

Architecture is the discipline of designing and constructing systems, and consists of a set of concepts, principles, rules and guidelines to be applied during design and construction. The TINA service architecture (by analogy) defines the concepts and basic principles necessary in constructing TINA services. Also, for certain types of services, additional principles may be defined that are more problem domain specific.

The term “*service*” is commonly used in multiple senses. A common-sense definition is that a service is a set of goods or valuable functions provided by

a service provider to a customer. In the telecommunications industry a service can be seen as a packaged set of capabilities that is perceived by a human user when interacting with a telecommunication network or a service provider and for which separate billing can be arranged. The definition that will be used in this document is :

“A TINA service is a meaningful set of capabilities provided by an existing or intended system to all business roles who utilize it.”

TINA services are classified in three main categories:

- *Telecommunications services* are services based on the transport of bits of information between terminals attached to a telecommunication network. The telecommunication service is responsible to the establishment of connections and processing of information related to the connections. A telecommunication network is transparent to the information that is carried between network endpoints.
- *Management services* are responsible for the management of TINA resources. It includes fault, configuration, accounting, performance and security functionalities, as well as service life-cycle management, service instance management and user life cycle management.
- *Information services* are services that are able to handle information resources such as movie, sound or document. It includes the storage (information content) and the visualization (the application that is able to interpret the resource). The services needed between storage and

visualization such as billing, caching and all such services specific to the information resource are also considered as information services.

3.1.2.3 Organization of concepts and principles

TINA is intended to be applied to all parts of telecommunications and information systems such as: terminals (personal computers, etc.), transport servers (switching systems, routers, etc.), service servers (Video on Demand (VoD), web, etc.) and management servers (authentication, billing, etc.). Reference Points are defined to specify conformance requirements for TINA products.

The architecture is based on four principles: Object-oriented analysis, distribution, decoupling of software components and separation of concerns. The purpose of these principles is to ensure interoperability, portability and reusability of software components and independence from specific technologies, and to share the burden of creating and managing a complex system among different business stakeholders, such as consumers, service providers, and connectivity providers.

Object-oriented analysis captures the complexity of a system from different angles and breaks it down into a set of models. (Refer to the Model and Reference Point Section) Distribution of service software components over different parts of the network accommodates traffic characteristics, network load or survivability, and specific customer demand. The distribution is supported by the Distributed Processing Environment (DPE).

Software components are decoupled from each other so that a change in one component due to a change in underlying technology (standards, languages, programs, materials, networks, etc.) would not affect other components.

TINA provides for two measures of separation of concerns. The first separation is between applications and the environment (i.e., DPE) on which they run. The second is separation of applications into the service-specific part and the generic management and control part. The latter interacts with transport and other elements.

3.1.2.4 Decomposition of the architecture

The TINA architecture is decomposed into four main subsets (Figure 3-4):

- *The service architecture* defines a set of concepts and principles for the design, specifications and management of telecommunication services.
- *The network architecture* defines a set of concepts and principles for the design, implementation and management of transport networks.
- *The management architecture* defines a set of concepts and principles for the design, implementation and management of software systems. These software systems are used to manage services, resources, software, and underlying technology.
- *The computing architecture* defines a set of concepts and principles for designing and building distributed software and the software support environment.

Figure 3-4 The TINA architecture.

3.1.2.5 Relationships between the architectures

The previously mentioned architectures interact and cooperate as follows:

- The *computing architecture* must be consistently used within the service, network and management architectures. This will ensure that software is constructed following the same basic principles, and will result in an interpretable and portable software.
- The *service architecture* requires service-oriented abstractions of network resources. The *network architecture* provides these abstractions.
- The *management architecture* defines the generic concepts and principles suitable for the management of services, networks, and computing infrastructures.

Figure 3-5 Relationships between Tina's sub-architectures.

3.1.2.6 Conformance

The TINA architecture is expressed as a set of concepts and principles. Moreover, it defines the style and nature of software in a TINA system. The concepts of TINA are classified as being one of two types: External and Internal. The *external* concepts define those concepts that must be adhered to in order to claim that a product is TINA conformant. The *internal* concepts are used and applied by the TINA-C team.

3.1.2.7 Overall framework of TINA systems

Software

- *Telecommunications software* is the software that implements the capabilities provided by the system.
- *Distributed Processing Environment (DPE)* is the software that supports the distributed execution of telecommunications applications.

Figure 3-6 Basic structure of telecommunication software in a TINA environment.

Layers approach

The computing structure consists of different layers. At the bottom, there are the *hardware resources*, such as processors, memory, and communication devices. Above the hardware resources is a *software layer* which is called *Native Computing and Communications Environment (NCCE)*. NCCE

contains the operating systems, communications, and other support software found in computing systems. Above the NCCE is the *DPE layer*, followed by the *telecommunications applications layer*. A TINA system may inter-work, at different levels, with non-TINA systems (Figure 3-6).

3.1.2.8 Main separations

Looking at TINA's application level, one can identify three main separations (Figure 3-7):

- *Element layer*

The first layer is the Element layer that is characterized by the following characteristics:

- It contains objects that represent atomic units of physical or logical resources.
- It is defined for allocation control, usage and management purposes.
- It is technology dependent.
- Elements do not directly interact with each other.
- Elements can be either software or hardware.
- Examples include: switching fabrics and transmission equipment.
- The second layer is the *Resources layer*. This layer manipulates collections of elements. Anything that is represented in the element layer should be handled by the resource layer.

Figure 3-7 Main separations in TINA.

- The third layer is the *service layer*. This layer contains objects involved in the provision of services to customers. Objects in this layer are service independent.

Interaction between the previously mentioned layers are characterized by the following constraints:

- An object can be in more than one layer provided the purpose is different.
- Objects are expected to access only in their same layer or in a contiguous layer.

3.1.2.9 Service architecture

The Service Architecture defines a set of principles for providing services. The notion of *session* is used. It is an important original concept of TINA. A *session* represents the information used by all processes involved in the provision of a service for certain duration. For example, in a multi-point conference, the information about connections, charging and user profiles may change during the conference as participants join and leave. The session helps keep such information coherent throughout the conference. A session can also be very simple (e.g., a web search). The notion of session is further refined to allow for separation of access, service usage, and communications.

Objects are separated into generic objects (common to all services) and service-specific objects (representing service logic, data, management, etc.). Several sessions are defined, corresponding to different types of activities (Figure 3-8):

- The *access session* corresponds to the establishment of the terms and conditions of the session (e.g., authentication, selection of service profile) during the connection of a user to system. It allows the user to start service sessions, combine sessions, and participate in several services.
- The *service session* corresponds to the provision of the service itself (e.g., moderated multi-party conference with information retrieval capabilities) and insures overall coherence of control and management. It is divided into:

- The *user service session* manages the state of each user's activity (local view) and resource attributes (e.g., charging context, current page),
- The *provider service session* contains the service logic and offers the functions allowing the user to join a session, to be invited in a session, etc. (global view).
- The *communication session* provides an abstract view of the actual transport network connections.

Figure 3-8 The session concept in TINA.

3.1.2.10 Advantages of TINA

The TINA service architecture fosters the rapid introduction of new service capabilities to satisfy market demands. Moreover, it enables interoperability between heterogeneous service software applications in the network. In addition, it provides independence between services and the networks on which they are provided.

3.1.2.11 Open issues for research

One of the main issues concerning TINA is that it will not exist in isolation. There are many other systems in existence, and these will not disappear overnight. Thus, TINA systems may have the ability to interact with other non-TINA systems, at the network or service levels. Currently, no architectural aspects for inter-working have been defined.

Another important issue is that TINA does not consider the specific means of dynamically specifying services or service components. It will not invent scripting languages or other mechanisms as these are in the application realm. It will consider the means of supporting such composition by location, communication, and other services. A service and service component type browsing are required. Such a service could have organizational as well as technical impacts and requirements.

3.2 RELATED RESEARCH

3.2.1 ACTIVE NETWORKS

Active networks are a novel approach to network architecture in which the switches of the network perform customized computations on the messages flowing through them. This approach is motivated by both lead user

applications, and the emergence of mobile code technologies. These mobile code technologies make network service innovation attainable. Work on active networks is currently being investigated at a number of different sites (mainly academic research centers):

3.2.1.1 Massachusetts Institute of Technology

The MIT team is prototyping an architecture based on the capsule approach and studying issues related to component specification. Capsules use the built-in constructs of a programming language to perform packet processing [Ten-97].

3.2.1.2 University of Pennsylvania

The SwitchWare project is developing a programmable switch approach that allows digitally signed type-checked modules to be loaded into the nodes of a network. The basic idea is to raise the level of abstraction of the switch functionality to be closer to that of a Turing machine [Smi-96].

3.2.1.3 Columbia University

The NetScript project, lead by Yemini and Da Silva, consists of a programming language and execution environment. The language provides a means to script the processing of packet streams. NetScript agents can be sent to remote systems including intermediate network nodes, such as routers and switches. The goal is to enable programming of these nodes as easily and quickly as end-systems [Yem-96].

3.2.1.4 University of Arizona

Active network concepts are being applied to network congestion by allowing applications to request those specific node algorithms. At the

University of Arizona, Peterson is developing “liquid” software, a suite of mobile code technologies that includes rapid compilation of intermediate code [Bha-96].

3.2.2 NETWORK MANAGEMENT USING AGENTS

There are several research centers and industrial applications that utilize agents in network management. In this section, we give an overview of some of the state-of-the-art development in this area since we are aiming at enhancing TINA using agents.

3.2.2.1 *University of Berlin*

The MAGNA project aims to investigate the impacts of mobile agents on the future of telecommunications architecture. One of its long-term goals is to develop a complete environment for agent-based development, including tools and components [MAGNA-99].

3.2.2.2 *Carleton University*

Pagurek leads The Perpetuum Mobile Procura Project at Carleton University. The overall goal of the Perpetuum Mobile Procura Project is to investigate the use of mobile code for managing networks. One of the project’s goals is to develop a mobile agent tool kit on top of Java. There are several other goals of the project including building a network simulator. The ultimate goal of the project is to utilize mobile agents to have self-repairing networks [Procura-99].

3.2.2.3 *EURESCOM*

The European Institute for Research and Strategic Studies in Telecommunications (EURESCOM) leads a variety of agent related research

projects. One of the main goals is to demonstrate the potential of agent technology in practice. Several prototypes and demonstrations have been already implemented. One of the main projects is called: Intelligent and mobile agents and their applicability to service and network management. The objective of that project is to assess and make recommendations on the applicability of intelligent and mobile agent technology to telecommunications service and network management [Eurescom-99].

3.2.2.4 Network agent vendors

The market for agent software is rapidly expanding. Table 3-1 contains a sample of products and vendors. No one vendor dominates the market, and it is open to competition.

Product	Vendor
TME-10 modules-Tivoli	IBM
IT/O Agent Openview	HP
Patrol Agents	BMC group
TNG Agents	Computer Associates
Bull ISM Agent Integrator	Groupe Bull
Command Center	Candle
Ecotools	Compuware
Pro Vision	Platinum Technology

Table 3-1 Network agent vendors.

3.3 SUMMARY

In this Chapter, we discussed the architectures of IN and TINA pointing to open issues for research. Also, we introduced a brief survey of related research in the area of agents and network management since we propose an agent-based architecture for TINA-based architectures in Chapter 6.

We can argue that the TINA-architecture can be enhanced by introducing agents (mobile and static). Although there are several relevant research projects that utilize agents in network management, very few investigate enhancing and realizing TINA using agents.

Chapter 4

SWE for Agents

Agent-based Software Engineering

Great results can not be achieved at once; and we must be satisfied to advance in life as we walk step by step.

Samuel Smiles

CHAPTER 4

AGENT-BASED SOFTWARE ENGINEERING

In this Chapter, we introduce an agent-based software engineering process based on existing object-oriented software engineering methods. We introduce samples of the notations to be used throughout analysis and design with examples to illustrate the ideas and notations. Finally, we introduce a brief background about petri-nets as we will use their concepts in the verification of the design.

4.1 AGENT-BASED DEVELOPMENT VERSUS OTHER DEVELOPMENT PARADIGMS

While most software provides users with significant value when used in isolation, there is an increasing demand for programs that can interoperate in order to exchange information and services with other programs and thereby solve problems that cannot be solved in isolation. Part of what makes interoperation difficult is heterogeneity. Programs are written by different people, in different languages, on different operating systems and on different machines-as a result they often provide different interfaces. Agent-based

software engineering is intended to facilitate the creation of software that is able to interoperate in heterogeneous setting [Gen-94]. It represents a novel and powerful way of approaching large-scale software engineering problems [Par-96]. In this approach to software development, applications are written as software agents, i.e. software components that communicate with their peers by exchanging messages in an expressive agent communication language. Software agents usually have some sort of persistent control (e.g. distinct control threads within a single address space, distinct processes on a single machine, or separate processes on different machines). The salient feature of an agent communication language is its expressiveness, allowing the exchange of data and logical information, individual commands and scripts (i.e. programs). The general evolution of autonomous agent based systems is shown in Figures (4-1) and (4-2).

In Figure 4-1, Pont and Moreole [Pont-96] view the various forms of software development as a logical progression, where methodologies have evolved in parallel with generations of programming languages. Thus, third generation languages (3GLs), like C, are compatible with a process-oriented approach to software development; 4GLs, like SQL, support entity-oriented style; 5GLs, like C++, are compatible with an object-oriented style of development; while 6GLs, like Java and Ada 95, provide good support for agent-oriented development.

As shown in Figure (4-1) and (4-2), the agent oriented (AO) approach builds upon the well-established object oriented (OO) techniques. However AO and OO approaches differ in a number of ways.

Figure 4-1 Evolution of software development paradigms [Pont-96].

Binary data types	Octal, hexadecimal
Basic data types	Real, integer, character
Structured data types	Strings, arrays, records
Abstract data types	Queues, sets, trees, lists
Objects	Windows, palettes, tool bars
Agents	Radar stations, factory machinery

Figure 4-2 Evolution of autonomous-agent systems [Bak-96].

4.1.1 AOP VERSUS OOP

AO (Agent-Oriented) technology builds upon and extends the now widely accepted Object-oriented (OO) approach to system design. The OO paradigm views an application as being composed of entities called objects that interact with each other by sending and receiving messages. Objects are grouped into classes. Then, class relationships can be captured in separate hierarchies. Objects are reactive entities as they receive messages and react to specific external events.

The AO paradigm goes one step further by viewing an application as being made up of one or more autonomous agents. Each agent continuously receives input. Agents respond to the environment where they reside by taking actions that affect the environment. Agents are pro-active entities, evaluating situations and then acting accordingly. Agents do not require specific instruction messages. Current applications being developed based on AO technology include Spacecraft Fault Diagnosis, Management and Business Process Management, Air Traffic Management, Air Mission Simulation and Telecommunication Network Management.

Objects in Object-Oriented Programming (OOP) represent real-world counterparts such as inventory items, documents, financial transactions and business processes. Objects are an intuitive mapping of the problem space to a computational model. They are characterized by state and defined behavior. They interact by passing messages since object internals are obscure.

Agent-oriented programming deals with both object and agent primitives. In some ways, agents may be thought of as objects, because they are also

characterized by state and behavior. Unlike objects, agents have goals and proactively work to achieve them. On one hand, object methods are simply procedures coded by a programmer to deal with different messages. For example, a dog object can bark when pinched. On the other hand, an agent can synthesize plans to deal with contingencies not foreseen by the programmer and can learn from experience. For example, a dog agent can discover that insistent barking gets him fed a lot quicker.

There are three fundamental principles of object-oriented programming. First, *Encapsulation* which is the hiding of data structures used to implement the object model. The second principle is *Inheritance* which is the reuse of code for objects which share similar implementations. The third main principle is *Polymorphism* which is the dynamic binding of object methods to incoming messages. As for agent-oriented programming, there are two main principles: goal-directed execution and transparent object management.

4.1.2 AOP VERSUS CLIENT/SERVER

Client/server techniques rely on the partitioning of the space into service consumers (clients) and suppliers (servers), and the management of communication channels between these clients and servers.

Agent-based computing builds on this metaphor. In addition it introduces a greater degree of parallelism and asynchronicity to the overall picture. For example, a distributed database working on the client/server model allows remote clients to submit queries to the database server for processing. It then returns these results over the network, the clients parse the output and extract the useful information, and formulate new queries to narrow their search. This

entire process is rather bandwidth-intensive. A more efficient method would be to make the clients send retrieval agents to the server and perform the extraction there. Such agents could bring the client's intelligence with them and move the processing closer to the information; thus saving time and bandwidth.

4.1.3 OO ANALYSIS AND DESIGN METHODS

There are many OO analysis/design methods. The Unified Modeling Language (UML) is considered the most widely accepted modeling language. Responsibility-Driven Design (RDD) method is one of the early OO methods, however, it provides a good starting point to develop agent-based systems. In this section, we will give a brief description of both RDD and UML as we will use their concepts in defining our agent-based development process.

4.1.3.1 Responsibility-Driven Design

Responsibility-driven design supports the basic concepts of object-orientation, such as classes, objects and inheritance. For each class, different responsibilities are defined which specify the responsibilities or the roles of the object. To fulfill these responsibilities, the classes need to collaborate with other classes using collaborations. In this method identifying and specifying responsibilities are the dominant starting point. A class inheritance hierarchy is built up using the responsibilities identified. Collaborations among objects are defined by contracts and protocols (i.e., the formal specification of method calls). Finally the system is structured by subsystems consisting of closely collaborating objects [WWW-90].

4.1.3.2 Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of engineering practices that have proven successful in the modeling of large and complex systems. The UML is a language, it is not just a notation for drawing diagrams. The UML is not a process, it provides a modeling language without enforcing a lifecycle. The UML contains the following models:

- *Structural models* contain diagrams that describe the static behavior of the system like class diagrams and object diagrams. Figure 4-3 shows some of the main UML notations used in structural models.
- *Behavioral models* contain diagrams that describe the dynamic behavior of the system like sequence charts and collaboration diagrams. Figure 4-4 shows a sample of a UML sequence chart. Figure 4-5 is a sample of a class state chart.
- *Implementation models* describe the organization and dependencies among software implementation components. Figure 4-6 shows a sample of UML implementation models.
- *Deployment models* describe the configuration of processing resource elements and the mapping of software implementation components onto them.

CLASS DIAGRAM Shows the existence of classes and their relationships in the logical view of a system

Figure 4-3 Classes notations in UML[Rat-98].

Figure 4-4 Sequence charts in UML.

Figure 4-5 Statecharts in UML.

Figure 4-6 Deployment models in UML.

The UML can be extended to model new elements using the concepts of stereotyping and tagged values.

Stereotypes

Stereotypes are a mechanism for introducing new types of modeling elements. Stereotypes are strings enclosed in (`<< >>`), preceding the name of an element. They may have an associated graphic icon or graphic marker. The new modeling elements that can be introduced using stereotypes must be subclasses of existing modeling elements [Alh-98].

Tagged values

Tagged values are a mechanism for specifying properties of model elements. Tagged values have an equal sign followed by a value following the keyword.

4.1.3.3 Conclusion

After a thorough investigation of OO analysis and design techniques it was found that they are not directly applicable to the development of multi-agent systems. This is basically due to the conceptual differences between objects and agents:

- Agents have a more complex behavior and structure than objects, and in this respect they are more comparable to subsystems in some OO methodologies, (e.g., [WWW-90]). Their internal structure differs from objects since agents have a more complex underlying functional architecture such as the belief-desire-intention (BDI) architecture [RG-92]. In this respect they are on a higher level of abstraction than objects.
- The internal states of objects are defined in terms of arbitrary attributes, while the internal states of agents are defined in terms of mental notions.
- In contrast to objects that are rather passive entities, agents are active. On one hand, objects immediately become active through messages and in this respect they are benevolent. On the other hand, agents act on their own behalf by following their goals, and can decide whether they act and respond to the events and messages received from other agents.
- Agents behaviors are described as scripts or plans by some graphical means that resemble state transition diagrams used to describe object life-cycles in some OO techniques. Agents' plans can be directly implemented using an agent programming language or an appropriate tool or

environment such as dMARS [Kin-93]. Specifically using such tools, implementing an agent is mainly specifying its plans.

- The communication of objects only looks at single messages. In AO not only messages are characterized by message types; but also agents dialogues with respect to specific contexts are pre-structured into cooperation protocols.
- Another difference between objects and agents is that objects usually do not have any autonomy as opposed to agents, which are characterized by their aims. Each agent follows its aims autonomously and independently, without being given instructions from outside.
- Mobility modeling is not handled by existing OO methods.
- The Responsibility Driven Design (RDD) method [WWW-90] for the design of object-oriented systems comes closest to the concepts in Agent Oriented Technology (AOT). But as was stated earlier, due to the conceptual differences between agents and objects, OO methods are not immediately applicable to the design and analysis of agent-oriented systems.
- The notations of UML can be extended to model agents (static and mobile) using stereotyping and tagged values.

4.2 SYSTEMATIC PROCESS FOR AGENT-BASED DEVELOPMENT

The first goal of this thesis is to develop a software engineering process for agent-based systems. The process consists of concepts, techniques, and steps to implement agent-based systems. Our process contains concepts from RDD and UML. We also added some new modeling elements and notations. We followed a similar approach to UML-RT (UML for Real Time) extensions [Selic-98]. A summary of the extensions to UML is included at the end of this chapter. Design verification is done utilizing Petri-Nets.

4.2.1 DEVELOPMENT LIFE-CYCLES

Since the early 1970s, system professionals have tried to describe the way they develop systems. The first effort was the classic waterfall life-cycle [Boehm-81]. It proved to be highly successful with traditional systems. Ed Yourdon [Your-92] points out problems with the waterfall model. Waterfall model does not promote software reuse and prototyping. In addition, waterfall model makes it difficult to trace requirements to code. Boehm modified the waterfall life-cycle, producing a series of spirals that explicitly includes prototyping. The spiral life-cycle is a strategy that reduces risk [Boehm-81]. The main advantage of the spiral life-cycle is that it is iterative. However, the iterative nature of the spiral life-cycle makes project deliverables harder to define. Developers can have 80% of the work done, but not completely finished with analysis or design. Another difficulty associated with both the waterfall and the spiral life-cycles is that they reflect the steps of structured development.

Second-generation OO methods suggest new life-cycle models. The MOSES method [Hen-94] uses what is called the fountain life-cycle (Figure

4-7). In this model, the spiral model is implicit and it emphasizes that phases overlap, that layers and subsystems emerge at various points in the development and that the development of individual components is uneven. The simile includes the idea of intermediate catchment basins. New components rise up through testing and program use into maintenance and further development. The fountain model is a useful metaphor but contains insufficient details to guide the development process [Hen-94].

Semantic Object Modeling Approach (SOMA by Graham, 1995), another second-generation method, enhances the fountain model and gives emphasis to projects interactions. Figure 4-8 shows the SOMA's life-cycle model.

Figure 4-7 The fountain life-cycle.

Figure 4-8 SOMA's life-cycle.

Due to the nature of agent-based development, we chose to use an iterative life-cycle since agent-based development is dynamic by nature. Figure 4-9 shows our proposed life-cycle in the form of a state chart. In the remaining of this Chapter the steps of our proposed process will be introduced.

Figure 4-9 Life cycle model for agent-based development process.

4.2.2 ANALYSIS

Analysis aims to clarify what a system is supposed to do, it mainly aims at answering “what” the system does rather than “how” to do it. The proposed analysis process is illustrated in Figure (4-10). The first step is to define the expected tasks of the system. The second step is to identify “places”, agents and classes. Then, a static model of the identified agents, classes and places is constructed. The analysis process is concluded by filling a data dictionary.

Figure 4-10 Analysis process.

4.2.2.1 Define tasks

As discussed earlier in section 4.1, agent-oriented paradigm is goal-oriented. Consequently, the first step should be defining expected tasks because agents will be identified based on these tasks. Figure (4-11) gives the syntax of the

task template. A task template will be filled for each task. The template includes the task name, its super-task (if it has any), sub-tasks (if it can be sub-divided), alternatives to achieve the task and exceptions.

The screenshot shows a window titled "Agent Based SWI" with a "Task Template" section. The form contains the following fields and text:

- Name:** Book a plane ticket.
- Super Tasks:** Booking
- Sub Tasks:** Find travel agencies.
- Exceptions:** No available tickets
- Failure Reasons:**
 - Unauthorized agent
 - Occurs if the requesting agent is not authorized.
- Alternatives to achieve the task:** Get the best available price.

Figure 4-11 Task template.

4.2.2.2 Define Places

The second step is to define “places”. The concept of places is associated with mobile agents. When a mobile agent transfers itself, it travels between execution environments called *places*. A *place* is a context within an agent system in which an agent can execute [OMG-99]. This context can provide functions such as access control. The source place and the destination place

can reside on the same agent system, or on different agent systems that support the same agent profile. A place is associated with a location, which consists of the place name and the address of the agent system within which the place resides. When a client requests the location of an agent, it receives the address of the place where the agent is executing.

In the electronic marketplace, a mainframe computer might function as a shopping center. For instance, a small shopping center might house a ticket place, where agents can purchase tickets to theaters and sporting events; a flower place, where agents can order flowers; and a directory place, where agents can learn about any place in the shopping center. The network might encompass many independently operated shopping centers, as well as many individually operated shops, many of the latter on personal computers.

Servers provide some places, and user computers provide others. For example, home place on a user's personal communicator might serve as the point of departure and return for agents that the user sends to server places.

4.2.2.2.1 Model relations between places

A place template will be filled for each identified place. The template contains place name, purpose and super-place (if it has any). Figure (4-12) shows the syntax of a place template. The following level describes the communications among places, which places are safe to accept agents from, which places are authorized to send agents to. The last level of the template contains the attributes and methods of the place.

The next step is to model relationships among places. Relationships among places include aggregation, inheritance and association. *Aggregation* can be defined as any mechanism for structuring place model whereby a new place is constructed from several other places and relationships. *Association* is any semantic relationship between two or more places. *Inheritance* is a mechanism that permits places to share characteristics. Section 4.2.2.5 includes the notations used to model places.

Figure 4-12 Place template.

4.2.2.3 Identify Agents

Agents and their characteristics were discussed in detail in Chapter 2. Agents can be considered similar to active objects in UML. In order to be able to have secure environments for mobile agents, the concept of agent's owner is utilized. The owner of an agent can be defined as the place where it

was created or originated from. The owner can later be used as a security identifier for the mobile agent. As for mobile agents there are three main actions that they can do. Mobile agents can *go to a place, meet another agent who resides in the same place and connect to another agent on another place.*

A simple starting point to identify candidate agents is to assign an agent for each task identified earlier in the first step. Moreover, agents can be identified for other purposes such as representing users, devices and databases. Figure 4-13 illustrates the syntax of the agent template.

The next step is to classify the identified agents. First, classify them in terms of mobility, whether they are static or mobile. Then agents will be classified according to their functionality:

- *Interface agents* are agents that interact with the user receiving user specifications and delivering results. They acquire, model and utilize user preferences to guide system coordination in support of the user's tasks. The main functions of an interface agent include: collecting relevant information from the user to initiate a task, and asking the user for additional information during problem solving.
- *Task agents* are agents that support decision making by formulating problem solving plans and carrying out these plans through querying and exchanging information with other software agents. A task agent receives user delegated task specifications from an interface agent and forms plans to satisfy these goals. Task agents are then classified into three categories:

- ☐ *Watcher agents* are agents that continuously monitor a stream of information and notifies its customer when it finds out ,
 - ☐ *Searcher agents* are agents that actively search,
 - ☐ *Analyst agents* are agents that make conclusions out of given data.
- *Information agents* are agents that answer a one-shot query about associated information sources.

The screenshot shows the 'Agent Template' dialog box in the JADE (Java Agent Development Environment) software. The dialog is titled 'Agent Template' and has a standard Windows-style title bar. It is divided into several sections:

- Name:** A text field containing 'PCA'.
- Role:** A dropdown menu set to 'Personal Communication Agent'.
- Super-Agent:** A text field containing 'Agent'.
- Characteristics:** A section with multiple checkboxes for defining agent capabilities. The checked options are 'P' (Proactive), 'S' (Social), and 'M' (Mobile). Other options include 'I' (Initiative), 'T' (Team), 'E' (Emotion), 'O' (Other), 'A' (Autonomous), 'C' (Cognitive), and 'D' (Dynamic).
- Behaviors:** A section with two text fields: 'Overs' and 'Main behavior to load'.
- Details:** A section with two text fields: 'Name' and 'Initialize'.

The dialog box is set against a dark, textured background.

Figure 4-13 Agent template.

The last step of this stage is to model the relationships among different agents. Relationships among agents include aggregation, inheritance and association. Section 4.2.2.5 includes the notations used to model agents.

4.2.2.4 Identify classes and objects

This step can be done using any of the existing OO methods. The Responsibility Driven Design (RDD) [WWW-90] approach can serve as a good starting point. However, any well-known analysis/design method can be used. Relationships between classes are modeled in this step. Again, relationships can be *association*, *aggregation* and *inheritance*. We chose Unified Modeling Language (UML) notations introduced earlier in section 4.1 [Rat-98] since UML is the most commonly used set of notations.

Figure 4-14 Notations for Agents, Mobile Agents and Places

4.2.2.5 Model relationships between agents, classes and places

In this step, a static view of the whole system is constructed. The aim is to describe the relationships between agents, classes and places in one global view. Figures (4-14,15) illustrate the notations that will be used throughout the thesis. It includes the notations for static and mobile agents, classes and places. It also shows the notations used for relationships.

Figure 4-15 Notations for relationships between component.

4.2.2.6 Create data dictionary

The concept of a data dictionary has been used in traditional software engineering and database tools for many years. A data dictionary is a central repository of definitions of terms and concepts. Without it, the process has

little semantic content. A data dictionary is a vital tool by virtue of its role as a single place to look up definitions. It provides clarity of thought and transition from phase to phase. A further important reason for having a data dictionary is that it helps someone unfamiliar with the development to understand it. Without a data dictionary the unfamiliar reader is often forced to rely on decoding meaningful identifiers. Figure (4-16) shows the syntax of the data dictionary. A set of shared vocabulary and meaning for concepts (often called an ontology) is then defined to conclude the analysis process.

<u>Name</u>	<u>Kind</u>	<u>Description</u>	<u>Comments</u>
Travel Agency Booking Agent User Interface Ticket	Place Mobile Agent Static Agent Class		

Figure 4-16 Data dictionary.

4.2.3 DESIGN

Design deals mainly with how to implement the system to produce the expected outcomes as stated in the analysis phase. The proposed design process is illustrated in Figure (4-17). Design starts by defining authorization rights for places. After that, state machines for agents, places and classes are constructed. Then, a life-cycle model (section 4.2.3.3) is constructed for each major task. Related places are then grouped in neighborhoods. The next step is to describe the dynamic behavior of the system by defining mobile agents itineraries and modified interaction diagrams (based on Jacobson's software engineering process[Jac-92]). The design phase is concluded by detailed

design for methods (operations) and attributes of classes, agents, and places. Finally, the data dictionary is updated.

4.2.3.1 Define authorization rights

The concept of authorization rights for each defined place is introduced because security issues have to be taken into consideration when dealing with mobile agents. Authorization rights indicate which agents are allowed to visit a certain place (their owner must be trustworthy). Places templates will be updated accordingly.

Figure 4-17 Design process.

4.2.3.2 Define State charts for agents, classes and places

State diagrams render the states and responses of a component. These diagrams describe the behavior of a class in response to external actors. These diagrams contain states and transitions. We use the UML notations for state diagrams (previously introduced in Figure 4-5).

4.2.3.3 Life-cycle model

The life-cycle model describes the behavior of agents and how they communicate with the environment from creation to death. A life-cycle expression defines the allowable sequences of interactions that an agent may participate in, over its lifetime. Figure (4-19) shows syntax and semantics of life-cycle expressions. The idea is applied in the Fusion method [Col-94]. Life-cycle expressions define patterns of communication. As an example consider a simple banking system. A very simple pattern occurs in response to balance inquiries. A check-balance event or task is always followed by the output of a current balance event. An example would be Inquiry=check_balance. #current_balance.

4.2.3.4 Define subsystems and neighborhoods

The next step is to start defining subsystems and neighborhoods. Subsystems are groups of agents, classes and places that collaborate together to perform a set of related tasks. First, we have to introduce the concept of a “Neighborhood”. Two places are in the same “neighborhood” if they are physically on the same host; i.e. agents do not have to travel from one place to another to perform a task if the two places are in the same neighborhood. Neighborhoods are useful to keep track of mobile agents. The main goal is to minimize agents roaming over the network. A neighborhood diagram (Figure

4-18) is constructed then for each group of places. Neighborhood diagrams are based on the deployment diagrams of the UML.

Figure 4-18 Neighborhood diagram.

Alphabet: Any input or output event maybe used in an expression. Output events are prefixed with #.

Operators: Let x and y be life-cycle expressions, then:

.x.y denotes x is followed by y.

.x|y denotes either x or y occurs.

.X* denotes zero or more occurrences of x.

.X+ denotes one or more occurrences of x.

.[x] denotes that x is optional.

.X||y means arbitrarily interleaving the elements of x and y.

Operators precedence: In decreasing order the precedence is: [], *, +, .., ||

Expressions may be bracketed to override default precedence.

Figure 4-19 Life-cycle notations.

4.2.3.5 Describe the dynamic behavior of the system

An itinerary graph is constructed for each scenario a mobile agent participates in. The itinerary graph shows places where the mobile agent passes by to achieve a certain task. Figure (4-20) shows an example of the itinerary graph of a personal travel agent. The example shows a mobile agent (Personal Travel Agent) that travels between 3 places (Personal PC, Flight Status Place and Paging Place) in order to book a ticket for a customer. The character indicates where the mobile agent started its itinerary. The number on each place indicates the number of the detailed interaction diagram that describes the interactions between the mobile agent and agents/classes in that place. For each identified task, an interaction diagram will be constructed. The interaction diagram describes how each task is achieved. Figure (4-21) illustrates a sample of our proposed interaction diagram. The interaction diagram is based on Jacobson's software engineering process [Jac-92]. Also, we added the arrow to indicate connecting to another agent in a different neighborhood. Connecting to agents in other neighborhoods should be minimized as it may result in bottle-necks. Meeting with other agents in the same neighborhood and sending messages to classes are both done using the simple arrow .

4.2.3.6 Detailed design of methods, attributes of each place, class and agent

The last step is the detailed design of the methods and attributes of each agent, class and place. Agent templates, place templates, class templates are then updated accordingly.

As mentioned previously, the process is iterative. Consequently, the analysis/design steps will be repeated till the whole system tasks are defined.

Figure 4-20 Itinerary graph.

Figure 4-21 Interaction diagram.

4.2.4 DEPLOYMENT

Deployment diagrams describe the configuration of processing resource elements and the mapping of software implementation components onto them. These diagrams contain nodes, components and their relations. We chose to use communication relationships diagrams from UML. Communication relationships are associations between nodes. They are used to model communication paths or connections between nodes. Figure 4-6 is an example of communication relationship diagrams.

4.2.5 VERIFICATION USING PETRI-NETS

The next step after design is to verify the design using Petri-Nets (PN). A Petri-Net will be constructed for each life-cycle model identified in the design process. The constructed Petri-Nets should be live and free of dead-locks. In the rest of this section, the principles of Petri-Nets will be introduced.

Petri-Net models have emerged as a very promising performance modeling tool for systems that exhibit concurrency, synchronization, and randomness [Dav-92]. This thesis uses Petri-Nets to analyze coherency and coordination of agent conversations. Petri-Nets (or place-transition nets) are bipartite graphs that provide an elegant and mathematically rigorous modeling framework for discrete event dynamic systems [Vis-92]. They manipulate events according to certain rules. This section explains some of the main concepts of a Petri-Net and how they can be used to analyze system coherency and coordination.

A PN is a four-tuple (P, T, IN, OUT) where

$P = \{p_1, p_2, p_3, \dots, p_n\}$ is a set of places

$T = \{t_1, t_2, t_3, \dots, t_m\}$ is a set of transitions

$P \cup T \neq \Phi, P \cap T = \Phi$

$IN: (T \times P) \rightarrow N$ is an input function that defines directed arcs from places to transitions, and

$OUT: (T \times P) \rightarrow N$ is an output function that defines directed arcs from transitions to places.

Figure 4-22 An example of a Petri-Net.

Pictorially, places are represented by circles and transitions by bars. In some models, places represent conditions or resources in the system while transitions model the activities in the system. Places and transitions are

connected by arcs. Places represent an “OR” situation, where only one of the arcs starting from it can be activated. Transitions represent an “AND” situation. When a transition fires, all the arcs starting from the transitions get activated.

Figure (4-22) shows a Petri-Net with five places and four transitions. In this Petri-Net

$P = \{p_1, p_2, p_3, p_4, p_5\}$ is the set of places,

and $T = \{t_1, t_2, t_3, t_4\}$ is the set of transitions.

4.2.5.1 Marking

A marking of a Petri-Net with n places is an $(1 \times n)$ row vector. It associates with each place a certain number of tokens or marks represented by dots. A marking represents the state of a Petri-Net model. We always associate an initial marking M_0 with a PN model. The marking of the Petri-Net in Figure (4-22) is a (1×5) vector. As, in the beginning only place p_1 is associated with a token, the initial marking of this Petri- Net is given by:

$$M_0 = [1 \ 0 \ 0 \ 0 \ 0]$$

4.2.5.2 Transition firing and reachability

A transition t_i of a PN can only be fired if each of the input places of this transition contains at least $IN(t_i, p_j)$ tokens. The transition is then said to be enabled. Firing of a transition t_i involves withdrawing the number of tokens spent by the input function, from each of the input places of transition t_i and adding them to each of the output places of transition t_i . An enabled transition

t_i can fire at any time. When a transition t_i enabled in a marking M fires, a new marking M' is reached according to equation

$$M'(p_j) = M(p_j) + \text{OUT}(t_i, p_j) - \text{IN}(t_i, p_j) \quad \forall p_j \in P$$

We say marking M' is reachable from M through firing of transition t_i . In Figure 4-22, transition t_1 can be fired as its input place p_1 is associated with a token. If t_1 fires, a token moves to p_2 and the new marking is given by $M_1 = [0 \ 1 \ 1 \ 0 \ 0]$. Marking M_1 is said to be reachable from marking M_0 . For a marked PN $G = \{P, T, \text{IN}, \text{OUT}, M_0\}$, a marking M is in the reachability set, $R(M_0)$, of G with respect to M_0 , iff there exists some valid sequence of transition firings $\{t_a t_b \dots t_i\} \subset T$, with which M can be reached. A reachability graph of a PN shows all the possible reachable markings of a PN and the associated transitions whose firing results in them. Figure (4-23) shows the reachability graph for the PN showed in Figure (4-22).

Figure 4-23 Reachability graph.

4.2.5.3 Incidence matrix

IN: $T \times P \rightarrow \{0, k\}$ is the input incidence matrix

OUT: $T \times P \rightarrow \{0, k\}$ is the output incidence matrix

Here, $k \geq 1$ is an integer and denotes the weight of the associated arc. $IN(t_i, p_j)$ is the weight of the arc $p_j \rightarrow t_i$. $OUT(t_i, p_j)$ is the weight of the arc $t_i \rightarrow p_j$. This weight is 0 if the arc does not exist.

The incidence matrix is

$$W = OUT - IN = w_{ij}$$

Row i of this matrix corresponds to the marking modification made by the firing of transition t_i . For example, the incidence matrix for the PN in Figure (4-22) is

$$W = \begin{bmatrix} P1 & P2 & P3 & P4 & P5 \\ -1 & +1 & +1 & 0 & 0 & t1 \\ 0 & -1 & 0 & +1 & 0 & t2 \\ 0 & 0 & -1 & 0 & +1 & t3 \\ +1 & 0 & 0 & -1 & -1 & t4 \end{bmatrix}$$

The incidence matrix is used to find out the change in the marking of a Petri-Net upon firing a given transition and the characteristic equation of the change in the marking is given by $M = M_0 + T.W$, where T is the firing vector stating which transition has fired, M_0 is the initial marking, M is the modified

marking, and W is the incidence matrix. In Figure (4-22), if transition t_1 is fired, the firing vector is $[1\ 0\ 0\ 0\ 0]$, and the modified marking is given by

$$M = [1\ 0\ 0\ 0\ 0] + [1\ 0\ 0\ 0\ 0] \begin{bmatrix} -1 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & -1 & -1 \end{bmatrix} = [0\ 1\ 1\ 0\ 0]$$

4.2.5.4 Safe Petri-Net

A PN is said to be safe for an initial marking M_0 if for all reachable markings, each place will contain at most one token. The marking matrix for the Petri-Net in Figure 4-22 is:

$$R = \begin{bmatrix} M0 & M1 & M2 & M3 & M4 \\ 1 & 0 & 0 & 0 & 0 & P1 \\ 0 & 1 & 0 & 1 & 0 & P2 \\ 0 & 1 & 1 & 0 & 0 & P3 \\ 0 & 0 & 1 & 0 & 1 & P4 \\ 0 & 0 & 0 & 1 & 1 & P5 \end{bmatrix}$$

It can be seen that in all possible marking of the Petri-Net, the number of tokens at any place never exceeds one. Thus, the Petri-Net is safe.

4.2.5.5 Liveness and Deadlock

The marking of a PN evolves by the firing of transitions. When certain transitions are no longer fireable and when all or part of the PN no longer 'functions,' there is most probably a problem in the design of the system. A transition t_i is live for an initial marking M_0 if for every reachable marking M_i from M_0 a firing sequence S from M_0 exists, which contains transition t_i . A

PN is live for an initial marking M_0 if all transitions are live for M_0 . In other words, if a PN is live, regardless of the evolution, no transition will become unfireable on a permanent basis. A deadlock is a marking such that no transition is enabled. Liveness of a PN implies absence of any potential deadlock states. It can be seen from the reachability graph in Figure (4-23), that transitions are always fireable. Thus it can be concluded that the Petri-Net in Figure (4-22) is live and deadlock free.

4.3 STEREOTYPES (EXTENSIONS TO UML LANGUAGE)

In this section, we include a list of the stereotypes we added to the UML (Figure 4-24). Stereotyping is the formal method to extend UML as mentioned before in section 4.1.3.2.

Name	Application
<<Agent>>	Class
<<Mobile Agent>>	Agent
<<Place>>	Class
<<Neighborhood>>	Package
<<Agent Template>>	Package
<<Itinerary graphs>>	Package
<<Place Template>>	Package
<<Life Cycle>>	Package
<<Task Template>>	Package
<<Interface Agent>>	Agent
<<Task Agent>>	Agent
<<Info. Agent>>	Agent
<<go>>	Operation
<<meet>>	Operation
<<dispose>>	Operation
<<connect>>	Operation

Figure 4-24 Extensions to UML.

4.4 SUMMARY

In this Chapter, we introduced the analysis/design process that we propose for modeling agent-based systems. The process is an enhancement to object-oriented methods. We argued that agent-based systems are the next step after object-oriented systems. Hence, object-oriented methods provide a valid starting point to model agent-based systems. The process we proposed is based on the UML notations, RDD concepts and some other notations and concepts. The process will be used in Chapter 6 to model the design of AgentTINA. Petri-Nets were introduced as a verification tool for agent-based systems.

Chapter 5

AgenTINA Specifications

Requirement Specifications For AgenTINA Architecture

Realize what you really want. It stops you from chasing butterflies and puts you to work digging gold.

William Moulton Martson

CHAPTER 5

REQUIREMENTS SPECIFICATIONS OF AGENTINA

In this chapter, we present the requirements specifications of AgentTINA. The requirements are mainly extracted from the TINA architecture requirements; accordingly, AgentTINA will be TINA-compliant. The requirements are classified in terms of users, network providers, service providers and third-party providers. Satisfying the requirements utilizing agents will then be discussed.

5.1 OVERVIEW

The functional requirements describe high level implementation independent requirements. These requirements are independent from the notion of an agent; this concept is introduced later. The system requirements will be derived from these functional requirements.

The following parties (Figure 5-1) are involved in the provisioning of an AgentTINA service and use their own negotiation strategies to meet their internal goals (neither of which will necessarily be publicly known):

- The initiating *user* will negotiate with the Service Provider about the terms and conditions of the service to be provided. The user is thought to be interested in satisfying his requirements at minimum cost.
- *The Service Provider* will negotiate with the user about terms and conditions as stated above. The Service Provider will also negotiate with its network provider in order to find the optimal solution for the provisioning of the service to the customer. The Service Provider has an interest in maximizing his profit.

Figure 5-1 AgentTINA involved parties.

- *The Network Provider* will negotiate with the service provider about terms and conditions as stated above. The Network Provider will also negotiate with other network providers (Third Parties) for parts of the connection it cannot deliver itself, or that can be offered more cheaply than the Network Provider can deliver. The Network Provider has an interest in maximizing his profit. The Network Provider will notify the receiving customers that their participation is required once the service has been established.
- *Third Party network providers* negotiate with the Network Provider as stated above. The Third Parties will also notify the receiving customers once the connection has been established.

5.1.1 USER REQUIREMENTS

5.1.1.1 Connection to be set at any place any time

- The service should have no restrictions for time and locality for the provisioning of the AgentTINA service.
- The user can issue a request anywhere in the network at any time.
- The users to be connected can be located anywhere in the network.

Example: The user may request the AgentTINA service at 2 am from a moving taxi using his Mobile phone to contact a local agent platform that resides in the Base Station of the mobile telephony operator.

5.1.1.2 Dynamic reconfiguration

The service parameters (e.g. QoS, Price, User List, Bandwidth) and the number of participating users can be changed dynamically during the life time of the service.

Example: The user may wish to change the bandwidth to allow video conferencing any time when the AgentINA service is active.

5.1.1.3 Reliability

The service should be reliable in the sense that the agreed quality of service is met, and that the risk of unexpected termination of the service is minimized.

Example: All parties jointly providing the service have measures in place to guarantee 99% availability of the service.

5.1.1.4 Fault Tolerance

The service is robust in the sense that it can recover from most exceptions.

Example: When a link that is part of the connection can no longer be provided because of a hardware fault in the switch, an alternative link is automatically set up to keep the connection alive.

5.1.1.5 On line billing

The service should be able to make billing information available on-line / real-time.

Example: The user decides to change bandwidth and is informed that this cannot be done within its current budget.

5.1.1.6 Security Levels

The service shall support different levels of security (authentication and confidentiality).

Example: A malicious user wants to use an established AgentTINA service and is informed that he/she is not a valid member of the user list.

5.1.1.7 Intelligent/flexible customer care

The service should provide enhanced customer support. It delivers intelligent responses on request of the user about the service provisioned.

Example: The user wants to know how much it will cost to add more participants (recipients) to the service. The AgentTINA service should be able to deliver the correct answer.

5.1.2 SERVICE PROVIDER REQUIREMENTS

The Service Providers are responsible for the provisioning of the service as required by the user, and have a goal to maximize profit. During the life time of the service, the Service Providers will be able to re-negotiate contracts with network providers in order to further optimize the service that is delivered to the user in terms of quality and cost. The dynamic re-negotiation and re-configuration will be invisible to the user.

5.1.2.1 Rapid service development and provisioning

AgentTINA should support the rapid development and deployment of services in order to respond promptly to market needs and at the same time reduce development costs. Accordingly, a common approach for the design

and management of all kinds of services is required in order to maximize the reuse of service components.

5.1.2.2 Tailored services

AgentTINA services should be readily customizable in order to satisfy specific requirements of a variety of customers (ranging from large companies to residential users). Service subscribers and end-users should be offered some direct control in managing their services.

5.1.2.3 Support for a multi-player environment

AgentTINA should fit in a multi-supplier/provider/operator environment. The coexistence of a number of stakeholders, performing various business roles, must be supported. In addition the architecture should define a flexible framework with respect to changes imposed by regulatory bodies. Accordingly, the AgentTINA service architecture should define an open environment enabling the introduction and modification of services, the introduction and modification of software and hardware components from different vendors and organizations and the interoperability among such services and components.

5.1.2.4 Profit Maximization

The service provider strives to maximize profit. This means that the service provider has a negotiation strategy that maximizes revenue, and minimizes cost for the deployment of the service. Negotiations will be undertaken within the constraints of required Quality of Service (QoS) and cost as specified by the customer / user.

5.1.2.5 Negotiation position with customer

AgentTINA should allow the service provider to effectively negotiate about terms of conditions and the cost of a dynamic service with the customer. The result of the negotiation will be a contractual agreement between the service provider and the user.

5.1.2.6 Negotiation position with network provider

AgentTINA should allow the service provider to effectively negotiate about terms of conditions and the cost of a dynamic AgentTINA service with the network provider. The result of the negotiation will be a contractual agreement between the service provider and the network provider.

5.1.2.7 User satisfaction

AgentTINA should allow the service provider to satisfy the requirements of the user during the entire life-time of the service in terms of cost and quality. This requirement implies that a dynamic AgentTINA service allows the Service Provider to dynamically change network provider when a better deal can be made elsewhere.

5.1.3 THIRD PARTY (NETWORK OPERATOR) REQUIREMENTS

5.1.3.1 Profit Maximization

The Network Operator strives to maximize profit. This means that the service provider has a negotiation strategy that maximizes revenue, and minimizes cost for the delivery of the connections over his network infrastructure. Negotiations will be undertaken within the constraints of required QoS and cost as specified by the service provider.

5.1.3.2 Negotiation Position

AgentTINA should allow the third party network operators to effectively negotiate about terms of conditions and the cost for a dynamic AgentTINA service. The result of the negotiation will be a contractual agreement between the network operator and the service provider.

5.2 SATISFYING THE REQUIREMENTS

The major high level requirements of the roles and actors in the AgentTINA architecture are the capabilities to negotiate about service conditions and configurations, and to notify (or be notified) accordingly. Service negotiation in this context will have the following objectives:

- Satisfaction of the requirements from users/customers.
- Optimization of the service conditions and configurations, e.g. minimal costs, maximum profits.

With traditional negotiation mechanisms, a user can only select the service features offered by the provider. The interface between the negotiation partners is fixed by specifications. A user can only modify the service parameters if such modifications are allowed in the interface specification. The possibility of dynamically optimizing the service conditions and configurations is limited.

Agents using a proper Agent Communication Language (ACL), can significantly enhance the possibility of dynamic negotiation and optimization. For example:

- The provider can change the knowledge (or inform such changes) of the user (e.g. the customer care component at the user site) about the service provisioning. In this way the provider can dynamically change the form of the service features or even the service itself in response to new user/provider requirements.
- The user can express wishes/preferences, inform the provider about the new requirements, and request new service features. With such information, the provider can infer the user characteristics and offer appropriate support.
- Service negotiation can have several phases following a contract net protocol in order to reach the optimal agreement between the involved parties.
- The involved parties can also modify their negotiation strategy dynamically, depending on the intermediate negotiation results.

Therefore agents provide a highly flexible, robust and user-friendly framework for service negotiations in the context dynamic AgentTINA services.

In this section, we try to fulfill AgentTINA's main requirements using agents. Each table will contain the main previously defined requirement, objectives to meet the requirement and how it will be fulfilled utilizing agents.

5.2.1 USER REQUIREMENTS

Objective	Fulfillment
Connection to be set at any place, any time	With agents, the requests and preferences of the users can be coded in the ACL message to the responsible service provider. The user can send the message from anywhere in the network, and can even disconnect itself from the network after sending the message (Section 6.5).
Personal assistants	Personal Communication Agents (PCAs) are created to act on behalf of users (Section 6.3).
Reliability	Negotiation based on ACL can treat exceptional situations more intelligently and support robust negotiations. Using composite messages, like mobile agents, we can encapsulate the negotiation steps or management actions within the messages. With such encapsulation we can reduce the number of messages transmitted over the global network and the dependency of provisioning on the underlying remote network for management traffic. This can further increase the reliability/fault tolerance of the provisioned service (Section 6.6).
Multimedia connections	Provisioning of the connections can be affected by many QoS parameters. Agents can provide enhanced support for negotiating such parameters, resulting in very flexible and user-oriented provisioning of the connections (Section 6.6).
Dynamic reconfiguration	Agent communication enables re-configuration of the agent's knowledge about service configuration and the corresponding functionality (Section 6.6).

Table 5-1 Satisfying user's requirements

5.2.2 SUPPORT FOR A MULTI-VENDOR MARKET

Objective	Fulfillment
Smooth multi-vendor interaction	Can be achieved by definition of standards and security measures. Mobile agents can be classified by their owners to prevent 'malicious' agents attacks.
Allow for service shopping	Can be achieved by a mobile agent that matches user requirements and gives an advice regarding choosing a service or a component (Section 6.4).

Table 5-2 Supporting multi-vendor market

5.2.3 SERVICE MANAGEABILITY

Objective	Fulfillment
Definition of principles and mechanisms to support service availability, reliability and performance	Can be achieved by agents that monitor the performance (according to some metrics) and availability of services (Section 6.6, 6.5)
Definition of mechanisms for service and components deployment and removal	Can be achieved by mobile agent that contact the concerned parties to notify them of the removal of the service and perform necessary modifications (Section 6.6).

Table 5-3 Satisfying Service manageability requirements

5.2.4 PROVIDER REQUIREMENTS

Objective	Fulfillment
User satisfaction	Agent-based approach allows the provider to dynamically configure the service features to meet the user requirements (Section 6.6).
Tailored services	Customization is supported by the user specific information objects. The customization will be supported in the computational model through the personal user agent (Section 6.3, 6.6).
Negotiation with customers	This will be supported by ACL messages and the corresponding contract net protocol (Section 6.6).

Table 5-4 Satisfying provider requirements

5.3 AGENTS REQUIREMENTS

In this section, we discuss some of the basic requirements for all agents. A well-behaved agent must fulfill its purpose, but it also must exhibit reasonably good judgment about handling the unexpected. Plus, some instances will require that agents be sensitive to security issues, such as verifying the identity of users or systems.

5.3.1 RELIABILITY

Agents should be reliable in the sense that the risk of unexpected failure of the services offered by an agent is minimized. For example, a personal agent should be capable of reconnecting itself after the connection has been temporarily disabled.

5.3.2 FAULT TOLERANCE

When a link that is part of the connection can no longer be provided because of a hardware fault in the switch, an alternative link is automatically set up to keep the connection established.

5.3.3 TRACEABILITY

For the purpose of dynamic testing, an agent should be able to keep track of all its activities which requires keeping a log. A user should always know what an agent is up to.

5.4 SUMMARY

In this chapter, we discussed the requirements of AgentTINA. The requirements are extracted from TINA [TINA-99] general requirements and based on FIPA [FIPA-99] general guidelines for agent-based dynamic service environments. We also showed how agents can fulfill the requirements. AgentTINA and its subsystems will be described in Chapter 6.

Chapter 6

AgenTINA Architecture

Major Components of AgenTINA

*There is no such things as a problem without a gift . . you seek problems
because you need their gifts.*

Richard Bach

CHAPTER 6

AGENTINA ARCHITECTURE

In this Chapter, we introduce the architecture of AgentTINA. AgentTINA's architecture satisfies the requirements discussed in chapter 5. We start by giving an overview of AgentTINA's architecture and components. Then, the main agents and classes of each sub-system are introduced and modeled using the process described in chapter 4.

6.1 MAIN SUBSYSTEMS AND RATIONALE

The AgentTINA architecture is divided into the following environments: Personal Communication Agent (PCA) Management Environment, Connection Management, Conversion and Utilities Management, Service Management and Shopping Environment. The main goal of AgentTINA's design is to enhance and realize TINA utilizing agents (static and mobile). An overview of AgentTINA is introduced in the remaining of this section. Figure 6-1 illustrates the major sub-systems of AgentTINA.

Figure 6-1 Main subsystems of AgentTINA.

6.1.1 PCA MANAGEMENT

The PCA management environment deals mainly with initiating and modifying personal communication agents (PCAs). Through this environment, users are supposed to fill in their calendars and personal preferences in general. The environment then launches the PCA to act on behalf of users in their interactions with other agents and classes in the environment.

6.1.2 SHOPPING ENVIRONMENT

The Shopping environment interacts with users and vendors to give users advice regarding buying devices, utilities and subscribing for new services. It

gets requests for shopping from the PCA and it responds by giving recommendations to users according to their preferences.

6.1.3 CONNECTION & NOTIFICATION MANAGEMENT

The main function of the connection management is to allow users to make a connection. Connection examples include connecting to the www, sending faxes and emails, etc.

6.1.4 SERVICE MANAGEMENT ENVIRONMENT

The Service management environment manages the following activities: monitoring equipment and services, negotiating for new services and updating service files with minimal human effort.

6.1.5 CONVERSION UTILITIES MANAGEMENT

The Utilities management environment manages conversion agents. The main task of the conversion agents is to convert files or messages from one format to another. Examples include agents that convert from text to speech and vice versa.

6.2 GENERAL COMPONENTS

In this section, we introduce some of the common components that will be used in many of AgenTINA's subsystems.

6.2.1 FACILITATORS

In the AgenTINA framework, the Facilitator agents play a key role. When an agent is added to the application, it registers its capabilities with the Facilitator. Part of this registration is the natural language vocabulary that can

be used to talk about the tasks that the agent can perform. When an agent needs some work done by other agents within the application, it sends a request to the Facilitator, which then delegates it to an agent, or agents, that have registered that they can handle the needed tasks. The ability of the Facilitator to handle complex requests from agents is an important attribute of the AgentTINA design. The goal is to minimize the information and assumptions that the developer must embed in an agent, thereby making it easier to reuse agents in other applications.

AgentTINA supports direct communication between application agents and blackboard communication through facilitators. The Facilitator (and supporting agents) handle the translation of the user's model of the task into the system model (analogous to how natural language interfaces to databases handle transforming the user's model into the database's schemas). The Facilitator simplifies reusing agents in new applications. The Facilitator is responsible for decomposing complex requests and translating the terminology used. This translation is typically handled by delegating it to another agent. The Facilitator is a potential bottleneck if there is a high volume of communication between the agents. Our focus has been on supporting a natural user interface to a community of intelligent agents, and this environment produces relatively low volume through the Facilitator.

6.2.2 TRIGGERS

In an increasing variety of conventional applications, users can set triggers (also called monitors or watch-dogs) to take specific actions when an event occurs. However, the possible actions are limited to those provided. AgentTINA supports triggers in which both the condition and action are parts

of a request that can cover the full range of functionality represented by the agents dynamically connected to the network. Triggers fire when a certain condition is satisfied in the form when (condition, action). Triggers may also fire if their timers fire. When the condition is satisfied or the timer fires, the action is performed. The use of triggers is shown in the email and www monitors (Section 6.5).

6.2.3 NOTIFICATION AGENTS

In this section, we introduce notification agents. The typical reasons, methods and types of notifications according to [Pal-99] will be discussed in this section.

6.2.3.1 *Typical Events*

There is no end of the different types of events related to interacting with intelligent networks in general. However, the ones listed in this section are commonly used in the design of intelligent agents.

6.2.3.1.1 *Activity and inactivity*

Activity means the user is working on the computer, using the keyboard or mouse. Activity events are useful for operations that should be performed only while the user is present. A real world stock market agent might constantly acquire and present information, but only when the user is present.

Inactivity is the realization that a certain amount of time has passed without any user activity. Inactivity events are useful for tasks best handled in the user's absence. This might include agents that perform disk maintenance, backups, or functions that are likely to monitor system resources.

6.2.3.1.2 New Email

The arrival of an email message is an event. Most email applications include agents for taking actions on incoming messages that meet certain criteria. Depending on its purpose, an agent might take action in response to any new email, email from certain parties, or email with certain content or attachments.

6.2.3.1.3 Web page changes

A change in a web page can be considered an event. An agent might have instructions to notify a user when a web page changes. To detect changes in a web page, the agent must poll the web page periodically until it detects a change in the page. Two methods can accomplish this: If the web server's HTTP header includes a date-modified field, the agent can retrieve the page's header periodically. Alternatively, the agent can retrieve the full page periodically and compare to a previous copy. When the contents differ, the agent realizes that the page has changed. Section 6.5 contains a web pages monitor that utilizes both methods suggested by [Pal-99].

6.2.3.1.4 Network additions and deletions

The addition or removal of systems to or from a network can be detectable events. This can be done by periodically attempting to access (ping) the range of possible IP addresses for your network. This will tell who is on line, but not who is physically connected, since off-line systems will not respond.

6.2.3.1.5 New hardware in the system

The addition of new hardware to a system can be a detectable event. This event can be detected by checking the system hardware configuration registry settings from time to time to see what has changed.

6.2.3.2 Reasons for notification

The following are the four most common reasons for notifications:

- *Task completion*: reporting that a process has completed successfully.
- *Error reporting*: informing the user about an error condition.
- *Alert notification*: reporting that something of interest to the user has been found or has occurred.
- *Status updates*: communicating progress or a change in status.

6.2.3.3 Methods of notification

6.2.3.3.1 Email

Email is a powerful method of notification, and it is the most common notification method available for agents. Email also is the only common notification method that can reach users who are away from their computers.

6.2.3.3.2 Faxing

Faxing by computer has been popular since the 1980s, almost all PCs now have fax modems. Many email servers, such as Microsoft Exchange, support fax servers that transform email messages into faxes.

6.2.3.3.3 Paging

Paging is a popular way to reach users who are away from their computers or on the road. Email is without question the simplest way to implement paging. Like email faxing, email paging avoids extra programming and purchasing an additional tool.

6.2.3.4 Types of notification

Passive notification does not force the user to receive the communication. *Active* notification forces the user to receive and acknowledge the communication. It is appropriate to use active notification when your agent must communicate critical information to the user. Message boxes and pager messages are examples of active notification, while email messages can be considered passive notification.

6.2.4 THE MICROSOFT AGENT

The Microsoft Agent API provides services that support the display and animation of animated characters. Implemented as an OLE Automation (Component Object Model [COM]) server, Microsoft Agent enables multiple applications, called clients or client applications, to host and access its animation, input, and output services at the same time. A client can be any application that supports the Microsoft Agent's COM interfaces [Msa-99]. The Microsoft agent (Genie) is used to notify users in different subsystems. In service management subsystem (Section 6.6), Genie notifies users that a new service or a hardware device has been added. In communication subsystem, Genie is used to read email messages for users. The utilities subsystem provides an interface to generate Microsoft agent animation sequences. Figure (6-2) shows a snapshot of Genie, the agent that will be used throughout AgentTINA's demo.

Figure 6-2 Microsoft agent (Genie).

6.3 PCA MANAGEMENT

6.3.1 INTERFACE AND OVERVIEW

The screenshot shows the 'AgenTINA Demo' window. The 'Personal' tab is active, displaying the following information:

- Name:** Zeid
- Office:** 5555
- Email:** masrooe@sprint.ca
- Phone:** (613)224-5351
- Fax:** (613)224-6629
- Pager:** (613)277-2647

On the right side, there is a 'Calendar' section showing a monthly calendar for June 1999. The date 'Today: 06/08/99' is highlighted. Below the calendar, there are fields for 'Time' (10:00) and 'Location' (1234 Herzberg). At the bottom, there are buttons for 'Cancel' and 'OK'.

Figure 6-3 PCA Management Interface.

The PCA management environment is designed to provide the user with a tool to create and personalize his Personal Communication Agent (PCA). The user is provided with an interface to enter his personal information and schedules. Figure (6-3) is a snapshot of the PCA management interface.

In addition, the user is provided with an option to view the current running PCAs in the environment (Figure 6-4).

A screenshot of a web browser window displaying a table titled "Current PCAs". The table has four columns: Name, Location, Phone, and Fax. It lists three entries: Zed, Simon, and Jordan, each with their respective location, phone number, and fax number. The browser's address bar shows "http://192.168.1.100:8080/".

Name	Location	Phone	Fax
Zed	1234 Herzberg	6132772647	6132246629
Simon	Meeting room	4185553344	5143433434
Jordan	Rowing	6132343434	6132258645

Figure 6-4 Current PCAs.

6.3.2 MAIN AGENTS, PLACES AND CLASSES

6.3.2.1 PCA

The PCA represents the customer/user in his dealings with Service Providers. Some of the main functions of the PCA were discussed in [Gar-98], while some were introduced in [FIPA-99]. The key functions performed by the PCA during service provisioning are as follows:

- Obtain and validate local service resources such as local telecommunication resources and terminals.
- Obtain and validate service partners destination and service constraints - starting time, duration and cost.
- Negotiate with Service Providers Agents (SPAs) the best cost-benefit balance for service provisioning in terms of the constraints and preferences defined by the user
- Suggest the best bid to the user. This is achieved by maintaining a user profile of the user and using this to judge the SPAs bids.
- Provide the user with monitoring capabilities to supervise the negotiation process.
- Enhance user profile through dialogue input. The PCA should use the user profile to minimize the dialogue with the user, asking questions like “would you contact the same people that you contacted the last time you were here” [FIPA-99].

The PCA has a wishlist, a list of items it wants to negotiate for and desired purchase price; the inventory, which contains all of the items it already succeeded to get; the negotiations, a list of negotiations in progress.

6.3.2.2 NamedPCA

A named Personal Communication Agent (namedPCA) is a service independent component that represents a user. The namedPCA acts as a single contact point to control and manage the life cycle of service sessions and user service sessions. It also manages the user's preferences on service access and service execution, and allows the user to use services from different types of terminals including support of user registration at different terminals to receive invitations.

The namedPCA supports the following capabilities:

- Act as a single contact point to control and manage (create/suspend/resume/delete) the life-cycle of service sessions and user service sessions, taking into account restrictions posed by the subscriber and the user.
- Suspend/resume existing user service sessions. This includes support for session mobility.
- Resolve the service execution environment for the user, allowing him/her to use services from many different types of terminals. This requires resource configuration information of the user system (which includes terminals and their access points being used by or available for the user). This includes support for personal mobility.

- Register user at a terminal to receive invitations. This includes support for personal mobility.
- Accept invitations from users to join a service session.
- Deliver invitations to a terminal, previously registered by the user with the namedPCA. No access session would be required to allow this delivery of invitations.

6.3.2.2.1 Main operations

- `listAccessSessions( )` - allows the user in this access session to find out about other access sessions that he has with this provider. (e.g. A user is at work, but has an access session set up at home which runs an active security service session.)
- `endAccessSession( )` - allows the user to end a specified access session, either the current one, or another, found using `listAccessSessions( )`. The user can also specify some actions to do if there are active service sessions.
- `getUserInfo( )` - gets the user's username, and other properties.
- `listSubscribedServices( )` - lists the services to which the user is subscribed. Scoping of subscribed services can be done using property lists. The operation returns sufficient information for the user to start a particular (subscribed) service.
- `discoverServices( )` - lists all the services available from the provider. The user can scope the list by supplying some properties that the service should have, and a maximum number to return.

- **listServiceSessions()** - lists the service sessions of the user. The request can be scoped by the access session, and session properties, (e.g. active, suspended, service type, etc.).
- **getSession** - retrieves information on a particular session.
- **listSessionInvitations()** - lists the invitations to join a service session that have been sent to the user.
- **listSessionAnnouncements()** - lists the service sessions which have been announced. It can be scoped by some announcement properties.
- **startService()** - allows the user to start a service session.
- **endSession()** - allows the user to end a service session.
- **endMyParticipation()** - allows the user to end his participation in a service session.
- **suspendSession()** - allows the user to suspend a service session.
- **suspendMyParticipation()** - allows the user to suspend his participation in a service session.
- **resumeSession()** - allows the user to resume a service session.
- **resumeMyParticipation()** - allows the user to resume his participation in a service session.
- **joinSessionWithInvitation()** - allows the user to join a service session, to which he has been invited.

- **joinSessionWithAnnouncement()** - allows the user to join a service session, which has been announced.
- **replyToInvitation()** - allows the user to reply to an invitation. It can be used to inform the service session to which they have been invited, that they will/will not be joining the session, or to send the invitation somewhere else. (It does not allow the user to join the session.)
- **participationSuspended()** - allows its clients to notify the namedPCA of the suspension of the user's participation in a service session.
- **participationEnded()** - allows its clients to notify the namedPCA of the end of the user's participation in a service session, and provide it with the relevant accounting information for the ended service session.
- **sessionSuspended()** - allows its clients to notify the namedPCA of the suspension of a service session where the user was taking part.
- **sessionEnded()** - allows its clients to notify the namedPCA of the end of a service session where the user was taking part, and provide it with the relevant accounting information for the ended service session.
- **sessionResumed()** - allows its clients to notify the namedPCA that a previously suspended service session, where the user was taking part, has been resumed.
- **getUserServiceProfile()** - It returns the service profile defined for the user for a specific service.
- **setUserServiceProfile()** - It allows to define the user service profile for a specific service. If it is consistent with the SAG Service Profile (profile for

the group the user belongs to) the operation will succeed. If not, an exception is returned.

- `getUserInformation( )` – retrieves users information.
- `setUserInformation( )` – sets user information.
- `Notify( )` - allows its clients to notify the namedPCA of new services added to the user's portfolio, or modifications or withdrawal of existing ones. It includes the list of services to which the notification refers and their corresponding service profiles.

6.3.2.3 AnonPCA

An anonymous Personal Communication Agent (anonPCA) is a service independent agent that represents a user. The anonPCA is a specialization of PCA for users who do not wish to disclose their identity to the provider. The anonPCA acts as a single contact point to control and manage the life cycle of service sessions and user service sessions. This management is possible during the anonymous user's current access session only. The anonPCA provides no support for personal or session mobility.

The anonPCA supports all of the capabilities which are defined for PCA. In addition, it supports the following capabilities:

- Suspend/resume existing user service sessions and service sessions within an access session;
- Manage the user's preferences (choices or constraints) on service access and service execution. (These would have to be determined during the access session, and could not be re-used in a separate access session.);

- Provide access to a user's contract information with the provider. (This contract would be defined at the start of the access session and terminated at the end of the access session.);
- Define user private /public policies. (This may be supported by starting a provider specific service session. This information would only be maintained during this access session.);

6.3.2.4 PCAF

A PCA factory is a service independent component that creates and initializes namedPCA and anonPCA upon request.

6.3.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES

Figure (6-5) illustrates the relationships between agents and classes. NamedPCA and anonPCA are sub-agents of the general case PCA. PCAF creates and initialize PCAs.

6.3.4 DATA DICTIONARY

Name	Type	Description	Comments
PCA	Static agent	General personal communication agent.	
NamedPCA	Static agent	PCA for a certain user.	
AnonPCA	Static agent	PCA for users who wish to hide their identity.	
PCAF	Static agent	Creates PCAs.

Figure 6-5 Relationships between PCA agent and classes.

6.4 SHOPPING ENVIRONMENT

The Shopping environment creates a shopping helper that displays available products from stores located at a virtual mall. Users can select a product and send an agent to the virtual mall to find its best available price. If users do not want to wait for the shopper agent to do its work, they can stop the applet and check for the result later. The Shopping environment contains static and mobile agents. The shopping environment demo is implemented using Voyager Mobile Agent framework [Voyager-99].

6.4.1 INTERFACE AND OVERVIEW

The main interface of the shopping environment is illustrated in Figure (6-6). The interface allows users to create a virtual mall. Also, it allows users to create stores. Stores are added to the mall automatically. Finally, the interface allows users to create and personalize a mobile shopper. Mobile shoppers then roam around available stores to find the required product with the best price. Figure (6-7) is a snapshot of the mobile shopper applet. In AgentTINA's demo, stores mainly contain communications tools like cell phones and faxes.

Figure 6-6 Shopping Demo Interface.

Figure 6-7 Mobile agent shoppers.

6.4.2 MAIN AGENTS, PLACES AND CLASSES

6.4.2.1 Shopper

A Shopper is a mobile agent responsible for locating and buying a product with the best available price. Products can be cell-phones, pagers, or any communications device. The shopper moves to the available malls, then checks available stores in each mall keeping track of the best price. Each Shopper has an owner, which is the name of the user. The Shopper also has an itinerary that contains the list of stores he will visit. Also, each shopper keeps track of the product it is trying to locate, its price and the store that offers that product.

6.4.2.1.1 Main operations

- **find:** the purpose of this method is to find the best price of a given product.

- **AtMall:** once at the mall place the shopper agent tries to locate the available stores in that mall, then starts moving to them according to the itinerary.
- **AtStore:** once at store, the Shopper agent tries to get the best price of the product it is looking for.

6.4.2.2 Store

A Store is a place that contains products. It contains a table of products and their corresponding prices. Each store has a name. Stores keep track of their stock. Stores are responsible for selling products to shoppers. Once a store is created, it notifies the mall about the products it offers.

6.4.2.2.1 Main operations

- **addProductToStock(product, price):** adds new products to existing stock.
- **getPrice(product):** gets price of a given product.
- **getProducts():** gets a list of all products to be added to the mall list of available goods.

6.4.2.3 Mall

A Mall is a place that contains virtual stores. It has a list of all products in its stores. A Mall accepts mobile shoppers (agents). New stores can be added to the Mall. When a new store is added, the Mall adds the new products of that store to its list.

6.4.2.3.1 Main operations

- **register (store):** adds a new store to the existing list of stores in the mall.

- `getProducts()`: gets all available products in all stores.
- `enter (name)`: allows a shopper mobile agent to enter the mall.

6.4.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES

Figure (6-8) displays the relationships between mobile agents, places and classes of the shopping environment. Each mall has a set of stores. Stores keep a list of products they sell. The shopper agent is authorized to move to malls and check on the existing stores to locate a certain product.

Figure 6-8 Relationships between shopping components.

6.4.4 DATA DICTIONARY

Name	Type	Description	Comments
Shopper	Mobile Agent	Finds best price of a given product by visiting available stores in the mall.	
Store	Place	Keeps track of products it sells	
Mall	Place	Keeps track of its stores and products.	
Products	Class	Keeps information about product including price and description.	

6.5 NOTIFICATIONS & CONNECTION MANAGEMENT

The Connection management subsystem is responsible for establishing email, phone and www connections when needed.

6.5.1 INTERFACE AND OVERVIEW

The main interface of the connection management is illustrated in Figure (6-9). The interface allows users to dial a given phone number, browse the www using a simple browser and send/receive emails. In addition, the AgentTINA demo includes an agent that monitors www pages and notifies the user when certain pages that interest him change.

Figure 6-9 Communication management interface.

6.5.1.1 Dialer

AgentTINA's dialer (Figure 6-10) provides a tool to dial any given phone number. It can be used to negotiate for best deals before dialing. Negotiation for carriers can be done in a similar way like services negotiation (Section 6.6). The dialer can be used together with the emailer to notify users by phone when they receive emails.

Figure 6-10 Dialer interface.

6.5.1.2 Browser

AgentTINA's www browser (Figure 6-12) has all the basic browsing functions. It is able to establish a connection automatically if the computer is not currently connected to the Internet (Figure 6-11). The main purpose of the browser is to provide users with a small-sized browser that provides all the basic functions. Also, it is used to test agents scripts (section 6.7).

Figure 6-11 Establishing a connection

Figure 6-12 Browser interface.

6.5.1.3 Email

AgentTINA email (Figure 6-13) provides all regular email functions and adds a smart monitoring agent (Figure 6-14). The email monitoring agent acts on behalf of its user. The monitoring agent can respond to any condition by performing an action. Conditions may include the subject of the message or the sender. Actions vary from performing a phone call, forwarding an email, to even reading the email to the user. Chapter 8 provides an application for such monitoring agents.

Figure 6-13 E-mailer interface.

Figure 6-14 Email monitor.

6.5.1.4 WWW pages monitor

The www pages monitor (Figure 6-15) monitors a list of favorite web pages. When a page changes, the agent automatically launches the user's browser to show the page.

Figure 6-15 Webpages monitor.

6.5.2 MAIN AGENTS PLACES AND CLASSES

6.5.2.1 Email monitor

After initialization, the agent starts monitoring the mailbox. The timer fires to check the user's mailbox frequently (every two minutes). The agent accesses the new unread messages and compares their contents against the user's preferences. When the contents match, the agent responds by performing the action.

The main tasks of the agent:

- Allow the user to assign an action to a message.
- Check users mailbox every 2 minutes for incoming messages.
- Perform the user's action when a matching message arrives.

6.5.2.1.1 Main operations

- **GetMessages** : checks the mailbox and retrieve new unread messages.
- **Compare**: matches unread messages with predefined conditions.
- **DefineActions**: enables users to associate actions to a message.

6.5.2.2 www monitor

After initialization, the agent must make a decision. If a list of URLs already exists, the agent will go to work. If no list of URLs exists, the agent keeps its dialog box visible so that the user can configure it for the first time. The timer fires to check the web sites every five minutes. The agent accesses each site and compares its contents against the site's previous contents. When the contents differ and the old contents are not empty, a web change is detected. The agent's response is to launch the default browser and display the web page.

The main tasks of the agent:

- Allow the user to enter a list of web pages.
- Check the list of web pages periodically and detect when they change.
- Launch the user's web browser to open a web page when the page has changed.

6.5.2.2.1 Main operations

- **UrlGet(URL):** Gets the contents of a given URL.
- **UrlCompare:** Compares the contents of two given web pages.
- **AddaURL:** Adds a URL to the list of URLs the agent keeps.
- **DeleteaURL:** Deletes a URL from the URLs list.

6.5.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES

Figure (6-16) shows the relationships between agents and classes for the email monitoring agent. Each monitoring agent has a timer and an action to perform when a predefined condition is satisfied. The email monitor is associated with the mailbox as it checks the mailbox when the timer fires. Figure (6-17) shows the relationships between agents and classes for the web pages monitor. The monitor has a timer and a list of URLs. It launches a www browser when changes happen.

Figure 6-16 Relationships between components for email monitor.

Figure 6-17 Relationships between components for webpages monitor.

6.5.4 DATA DICTIONARY

Name	Type	Description	Comments
Email monitor	Static agent	Monitors mailbox and performs actions when a message arrives	
Homepages monitor	Static agent	Launches browser if any of the URLs it keeps track of change.	

6.6 SERVICE MANAGEMENT

The service management subsystem provides the users with facilities to negotiate for services and networks. It also provides a remote service update facility. In addition, it provides a monitoring facility for network equipment and services.

6.6.1 INTERFACE AND OVERVIEW

Figure 6-18 Service management interface.

The main service interface view (Figure 6-18) provides interface to some of the implemented functions. The negotiations options lead the user to a negotiation application, while the monitoring view provides a description of the equipment-related activities.

6.6.1.1 Monitoring a network addition

Monitoring in AgentTINA's demo contains three main options. The Service Detection Agent (SDA) monitors the network and reports any addition, removal or change in the configuration of any of the installed services and equipment. The following is the sequence that happens when a new network printer is added. The printer is added using an Add printer wizard (Figure 6-

19,6-20). When the printer is installed successfully, Genie (The Microsoft agent) will appear in the Monitor Equipment view saying: “A printer was added”. The activities script message will be updated accordingly with the date, time and the event (Figure 6-21).

Figure 6-19 Adding a network printer.

Figure 6-20 Adding a network printer.

Figure 6-21 The agent notifies users that a network printer was added.

6.6.1.2 Negotiation

The negotiation view provides an interface to show the transactions happening between Service Provider Agents (SPAs) and Personal Communication Agents (PCAs). Figure (6-22) illustrates a sample of PCA/SPA negotiation process. The upper script window shows the routing of messages from PCA to SPA through the Facilitator. The lower script view shows the transactions between the agents during the negotiation process.

Figure 6-22 Negotiation between SPA and PCA.

6.6.2 MAIN AGENTS, PLACES AND CLASSES

6.6.2.1 Facilitator

The Facilitator acts as a blackboard for agents negotiations. It contains two tables, one which is a registry for all SPAs and PCAs, and one which contains groups of interest. For example, there may be a group of SPAs which provides certain services, and another which sells utilities. SPAs are grouped into domains. SPAs are added to the matching domain by advertising their ability to provide a service in this domain. PCA may ask the Facilitator to recommend an SPA for a certain item. Negotiations could proceed between

SPA and PCA resulting in a contract. Some of the main tasks of the Facilitator are:

- SPA registration.
- PCA registration when it changes its location.
- Provision of the list of service providers.
- Provision of location information of service participants' PCAs.

After all agents are registered with the Facilitator and the SPAs advertise their services the following sequence takes place:

1. PCAs ask Facilitator to recommend an SPA that offers a certain service, Netmeeting for example.
2. Facilitator tells the PCA the name of the SPA.
3. PCA asks SPA if it has Netmeeting for sale.
4. SPA either make offer to the PCA, or will deny it has the item.
5. PCA can either accept the offer or make a counter offer to the SPA.
6. SPA can either accept the offer, make a counter offer or reject the offer.
7. If the SPA accepts, it sends a message to the PCA and the transaction is complete.
8. If the SPA rejects the offer, the negotiation is over.

6.6.2.1.1 Main operations

- Recommend: checks to see if there are any agents in the domain of interest. If so, the Facilitator determines how many agents have advertised and then select one to recommend to the PCA.

- Advertise, unadvertise: either add or remove the sending agent from a community.

6.6.2.2 Service Provider Agent (SPA)

The SPA represents the interests of the Service Provider and supports the provisioning of telecommunication services to customers. It adopts two distinct roles:

- Client of network services offered by Network Provider Agents (NPA)
- Provider of a variety of telecommunication services to end customers represented by the PCAs

A SPA can also support management activities such as customer profile management and billing of the customers.

Some of the main functions of the SPA were discussed in [Gar-98], while some were introduced in [FIPA-99]. The key functions performed by the SPA are as follows:

- Validating the service calls from the PCAs and generates appropriate service proposals to the customers.
- Negotiation with the NPA for the service connections agreed between the SPA and the PCA.
- Requesting connections from the NPA.
- Informing the PCAs upon activation of the agreed service according to the service scheduler.

- Receiving service management requests (configuration, termination) from the PCAs, validating them and performing such management operations by operation requests to the NPA.
- Informing the initiating PCA about the service management results.

The SPA has a wishlist, a list of items it wants to negotiate for and desired purchase price; the inventory, which contains all of the items it already succeeded to get; the negotiations, a list of negotiations in progress.

The SPA has an inventory list, a table of items the agent offers and their prices, and the negotiations, a table of negotiations in progress. After the SPA advertises its services to the Facilitator, it waits for a prospective PCA to contact the Facilitator looking for a recommendation. When the Facilitator passes the SPA name to the PCA, the PCA sends a message to start the negotiations.

The SPA can respond to an initial request from a PCA or respond to an offer from the PCA. When the PCA asks about a service, the SPA must check to see if it can offer the desired service. Assuming the SPA can offer the service, it then generates an offer.

6.6.2.3 Network Provider Agent (NPA)

The NPA represents a network domain. Its major responsibility is the provisioning of network connectivity upon requests from a SPA. For this purpose, the NPA has to interact with the SPA representing the customer, the Network Management System (NMS) representing the local network domain

and with other NPAs representing other network domains in the global environment [FIPA-99].

To obtain the network connection from the NPA, the SPA will first negotiate with the associated NPA and inform the NPA the requirements on the connection. This negotiation can consider an already existing long-term contract between the two parties, but has to support the specific requirements of the current session. The knowledge needed by NPA in this interaction includes the “Service Description/Knowledge” and the “In Service Requirements”.

Some of the main functions of the NPA were discussed in [Gar-98], while some were introduced in [FIPA-99]. The key functions performed by the NPA during service provisioning are as follows:

- respond to call for proposals (cfp) for connections
- negotiate with the SPA for connections
- Receive network management requests (configuration, termination) from the SPAs, validate them and perform such management operations
- provide connections for participating partners (PCAs)

The NPA has an inventory list, a table of items the agent offers and their prices, and the negotiations, a table of negotiations in progress. After the NPA advertises its services to the Facilitator, it waits for a prospective SPA to contact the Facilitator looking for a recommendation. When the Facilitator

passes the NPA name to the SPA, the SPA sends a message to start the negotiations.

6.6.2.4 Service Upgrading Agent (SUA)

The purpose of the Service Upgrading Agent (SUA) is to update remote files by replacing all occurrences of one specified word in the files with another specified word. The philosophy behind SUA is as follows: for files larger than a certain size, it is considered beneficial (it saves network bandwidth) to perform a distributed update of the files. Distributed update saves downloading and uploading of the file and distributes the load of update to multiple servers. The idea of the SDA was inspired from a technical presentation about the Aglets workbench [Aglets-99]. Check section (7.6) for the scenario that deals with remote service upgrading.

6.6.2.5.1 Main operations

- **OnArrival:** This operation is automatically invoked when the SUA arrives at the new host. The method will perform the service update, followed by the disposal of the SUA.

6.6.2.6 Service Detection Agent (SDA)

The purpose of the Service Detection Agent (SDA) is to detect if a new hardware or a service has been added to the environment. This event can be detected by checking the system hardware configuration registry settings from time to time to see what has changed. The SDA then invokes Genie, the Microsoft agent, to inform the user of a hardware addition. Similarly, the SDA invokes Genie in case of a hardware removal and change of configuration.

6.6.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES

Figure (6-23) shows the relationships between services agents. SPAs and NPAs communicate through the Facilitator. SUA is a mobile agent that is associated with services. Finally SDA monitors services and hardware and invokes Genie when a change takes place.

Figure 6-23 Relationships between service agents.

6.6.4 DATA DICTIONARY

Name	Type	Description	Comments
Facilitator	Static Agent	Acts as a blackboard for PCA/SPA negotiations.	
SPA	Static Agent	Negotiates with NPAs in behalf of users.	
NPA	Static Agent	Negotiates with SPA for service provisioning.	
SDA	Static Agent	Detects changes and invokes Genie.	
SUA	Mobile Agent	Upgrades service files.	

6.7 UTILITY MANAGEMENT

The Utilities management's main function is to manage the active conversion agents. In AgenTINA's demo, the main utility agent is the text-to-speech agent.

6.7.1 INTERFACE AND OVERVIEW

The AgenTINA text-to-speech view is designed to provide the user with a tool to control one of Microsoft's interface agents. The purpose is to generate the code in Javascript for a set of agent's actions. Actions include animation, speech and reactions. The text-to-speech agent is just an example of a utility agent, other agents may include fax-to-text, voice-to-text, etc. Figure (6-25) is a snapshot of the text-to-speech interface. When the "Generate Java Script" button is pressed the agent actions are interpreted into a Javascript file that can be run using any browser. When the "Test" button is pressed, the sequence of the agent's actions will be played using the browser interface (See Figure 6-24).

Figure 6-24 Testing the agents actions.

Figure 6-25 Text-to-speech interface.

6.7.2 MAIN AGENTS, PLACES AND CLASSES

Conversion agents are provided by service providers. They are mainly used to transform files or messages from one format to another.

6.7.2.1 Microsoft agent

6.7.2.1.1 Animation Services

The Microsoft Agent's animation services manage the animation and movement of a character's image in its own window on the screen. An animation is defined as a sequence of timed and optionally branched frames, composed of one or more images. Specifying the Play statement with the name of an animation plays that animation. Animation names are specific to a character definition. As an animation plays, the shape of its window changes to match the image in the frame. This results in a movable graphic image, or sprite, displayed on top of the desktop and all windows.

Each client application can display and animate its own character. You can also share a character between multiple client applications. Microsoft Agent also supports clients using multiple characters displayed at the same time. The animation services enable users to animate characters independently or synchronize their animation.

To access a character, the Load method to load the character's data is used. Microsoft Agent's services include a data provider that supports two formats for loading character and animation data: a single structured file and separate files. Typically, users would use the single file format (.ACS) when the data can be stored locally. The multiple file format (.ACF, .AAF) works best when

you are downloading animation individually, such as when accessing animation from a Web page script [Msa-99].

6.7.2.1.2 Audio Output Support

Microsoft Agent enables users to use audio files for a character's spoken output. Users can record audio files and use the `Speak` method to play that data. Microsoft Agent animation services automatically support lip-syncing the character mouth by using the audio characteristics of the audio file. Microsoft Agent also supports a special format for audio files, which includes additional phoneme and word-break information for more enhanced lip-sync support.

6.7.2.1.3 Word Balloon Support

Spoken output can also appear as textual output in the form of a cartoon word balloon. This can be used to supplement the spoken output of a character or as an alternative to audio output. Word balloons support only captioned communication from the character, not user input. Therefore, the word balloon does not support input controls.

6.7.2.2 Code generator

The code generator agent is concerned mainly with generating Java script files that match the agent animation sequences generated by users. The generated script can run directly using any www browser. Users can personalize the generated script and use it in their applications. Appendix A contains a sample of the files generated by the code generator. The code includes initialization and loading of the Microsoft Agent. In addition, the

code generator produces Java script statements that match the animation and speech acts created by the user.

6.7.3 RELATIONSHIPS BETWEEN AGENTS, PLACES AND CLASSES

Figure (6-26) shows the relationships between agents and classes. A converter is the general conversion agent. Text-speech, Fax-text and Video-audio agents are examples of sub-agents of the converter. The Code-generator agent is associated with converters and Genie. The Code-generator creates the JavaScriptOutput file. The Code-generator agent is associated with converters and Genie. The Code-generator creates the JavaScriptOutput file.

Figure 6-26 Relationships between utilities agents.

6.7.4 DATA DICTIONARY

Name	Type	Description	Comments
Text-to-speech	Static Agent	Converts written text to speech.	
Genie	Static Agent	Provides animation and speech facilities for users.	
Code generator	Static Agent	Generates Java script files for Genie's animation and speech.	
Fax-text	Static Agent	Converts Fax format to text format.	

6.8 SUMMARY

In this Chapter, we introduced the architecture of AgentTINA. The architecture consists of five main subsystems: PCA management, Connection management, Shopping environment, Service management and Utilities (conversion) management. AgentTINA's architecture utilizes mobile and static agents to provide a TINA-compliant architecture that satisfies the requirements specifications (Chapter 5). AgentTINA contains a set of tools and agents that can be used to develop a variety of messaging applications. We used many tools to implement AgentTINA's demo: Java, Voyager mobile agent environment, Visual C++, Visual Basic and Microsoft Agent 2.0 SDK. Chapter 8 contains a practical application designed using AgentTINA.

Chapter 7

AgentTINA Verification

Verification of AgentTINA Design Using Petri-Nets

I think and think for months and years. Ninety-nine times the conclusion is false. The hundredth time I am right.

Einstein

CHAPTER 7

VERIFICATION OF AGENTINA DESIGN USING PETRI-NETS

In this Chapter, we apply the methodology described in Chapter 4 to verify the design of AgentTINA (described in Chapter 6). The main purpose of this Chapter is to illustrate the process we propose to verify the reliability of AgentTINA design. We present some main scenarios and test cases for AgentTINA to illustrate the verification process. Petri-Net analysis is then presented for each scenario.

7.1 PROCESS OF VERIFICATION

The process of verification is illustrated in Figure 7-1. Each case we present in this chapter will start by a scenario description in plain English. The life-cycle model of the scenario is then constructed. Next step is to construct itinerary graphs for mobile agents. Interaction diagrams for the scenario are then presented. Finally, we transform Life-cycle models to Petri-Nets. Details of petri-net analysis can be found in Appendix-B.

We chose Petri-nets as a verification tool due to the following reasons:

- Petri-nets provide the benefits of formal modeling.
- Petri-nets provide a relatively simple formalism.
- Petri-nets have an intuitive graphical representation with many tools for analysis and design.
- Petri-nets can help in visualizing the dynamic behavior of agents.

Figure 7-1 Verification process.

We then generate reachability graphs for petri-nets. The following properties of petri-nets are checked:

- Deadlocking states in which an agent is no longer able to support interaction with its environment or its autonomous behavior.
- Safety and reliability properties, meaning that agents ensure results for the actions they have committed themselves to achieve and thus are trustworthy.

The notations we introduced in Chapter 4 will be used to describe the scenarios.

7.2 SHOPPING USING MOBILE AGENTS

7.2.1 SCENARIO DESCRIPTION

This scenario describes in general the process of dispatching a mobile agent to another 'place'. The mobile agent is interpreted in the destination place then it performs its intended action. After the action is performed the system will reset to its initial state. This scenario is a general case for all shopping mobile agents (section 6.4).

7.2.2 LIFE-CYCLE MODEL

The shopper moves to the virtual mall, following that it checks the stores (one or many) looking for the best available price. Then the shopper moves back to the user's host with the best price. The life-cycle model of the process is as follows:

(Move to mall).(Move to store)*. # (Get best price)*

7.2.3 INTERACTION DIAGRAM

Figure (7-2) shows the itinerary graph for a mobile shopper. It starts by moving to the virtual mall. Next, it passes by available stores. Figure (7-3) is the interaction diagram for the mobile agent shopper. It shows the interaction between the shopper agent, the mall and related stores.

Figure 7-2 Itinerary graph for a shopper.

Figure 7-3 Interaction diagram for a shopping scenario.

7.2.4 PETRI-NET MODEL

7.2.4.1 The net (places and transitions)

Places		Transitions	
ID	Description	ID	Description
P ₁	Agent ready to move	T ₁	Move to mall
P ₂	Agent initialization	T ₂	Move to mall
P ₃	Agent on the way	T ₃	Create shopper
P ₄	Agent on the way	T ₄	Move to store
P ₅	Ready to accept agent	T ₅	Move to store
P ₆	Agent at store	T ₆	Get best price

Figure 7-4 Petri-net model for a shopping scenario.

7.2.4.2 Reachability Graph

Figure 7-5 Reachability graph for dispatching mobile agents.

$$M_0 = [1 \ 0 \ 0 \ 0 \ 1 \ 0] \quad M_1 = [0 \ 0 \ 1 \ 0 \ 1 \ 0]$$

$$M_2 = [0 \ 0 \ 0 \ 1 \ 1 \ 0] \quad M_3 = [0 \ 0 \ 0 \ 0 \ 0 \ 1] \quad M_4 = [0 \ 1 \ 0 \ 0 \ 1 \ 0]$$

7.2.4.2.1 Reachability matrix

	M ₀	M ₁	M ₂	M ₃	M ₄
P ₁	1	0	0	0	0
P ₂	0	0	0	0	1
P ₃	0	1	0	0	0
P ₄	0	0	1	0	0
P ₅	1	1	1	0	1
P ₆	0	0	0	1	0

[i]

Equation [i] gives the marking matrix of the PN model. This matrix shows that at all times the number of markings in a place is never greater than one.

This implies that the petri-net system is safe. Consequently, clones of the shopper agents will not exist simultaneously in the same place. This verifies that the network will not be flooded with shoppers. Figure (7-5) shows the reachability graph for the petri-net model. We can see from the graph that from all the reachable markings of the system, a firing sequence (that contains every transition of the petri-net) exists. That implies that every transition of the petri-net is live and the system is deadlock free.

7.3 NEGOTIATION BETWEEN PCA AND SPA

7.3.1 SCENARIO DESCRIPTION

This scenario is a general scenario for the negotiation process between two agents. The general idea is that an agent makes a request for a service by sending a message to the Facilitator (Blackboard), other agents respond to the request by sending proposals to the Facilitator. The negotiation between the PCA(Personal communication agent) and the SPA (Service provider agent) for services can be used as an example of a general negotiation process.

7.3.2 LIFE-CYCLE MODEL

The life-cycle model starts by the PCA sending a request for proposal to the Facilitator. The SPA either fails to respond or starts checking offers. If the SPA fails the negotiation process terminates. If the SPA starts checking offers, it will search through the best available offers and report back to the PCA (via the Facilitator) to establish a contract then the process will terminate. The following is the life-cycle model:

(PCA sends a request for proposal).(SPA Fails|(SPA starts checking offers)')

(PCA sends a request for proposal).(SPA Fails).(Terminate and restart)

(PCA sends a request for proposal).(SPA starts checking offers)' .(SPA negotiations start).#(SPA gets the best offer, establish contract with PCA).(Terminate and restart).

7.3.3 INTERACTION DIAGRAM

Figure (7-6) shows the interaction diagram of the negotiation scenario. The main agents involved are the PCA (Personal Communication Agent), the SPA (Service Provider Agent) and the Facilitator.

Figure 7-6 Interaction diagram for negotiation between PCA and SPA.

7.3.4 PETRI-NET MODEL

7.3.4.1 The net (places and transitions)

Figure 7-7 Petri-net model for negotiation between PCA and SPA.

Places		Transitions	
ID	Description	ID	Description
P ₁	PCA ready	T ₁	PCA posts a request for offers.
P ₂	PCA waiting for a response from Facilitator	T ₂	SPA failed to provide offers.
P ₃	Facilitator has a request	T ₃	SPA starts checking available offers.
P ₄	Facilitator has no offers to match request	T ₄	Conversation failed (no offers from SPA).

P ₅	Facilitator has an offer.	T ₅	Negotiation between SPA and other agents starts.
P ₆	Failure in negotiation.	T ₆	SPA rejects PCAs bid.
P ₇	Negotiation is over successfully.	T ₇	SPA accepts PCAs bid.
P ₈	PCA waiting for PSA to accept/reject a bid.	T ₈	Negotiation ends.
P ₉	SPA ready.	T ₉	Establish contract.
P ₁₀	SPA checking PCAs bids.	T ₁₀	Reset, look for other offers.
P ₁₁	Contract established.	T ₁₁	Reset, negotiation succeeded.
P ₁₂	Conversation ended with no contracts.		

7.3.4.2 Reachability Graph

Figure 7-8 Reachability graph for negotiations.

$$\begin{aligned}
M_0 &= [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0] & M_1 &= [0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0] \\
M_2 &= [0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] & M_3 &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1] \\
M_4 &= [0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] & M_5 &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0] \\
M_6 &= [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0] & M_7 &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0] \\
M_8 &= [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0]
\end{aligned}$$

7.3.4.2.1 Reachability matrix

	M ₀	M ₁	M ₂	M ₃	M ₄	M ₅	M ₆	M ₇	M ₈	
P ₁	1	0	0	0	0	0	0	0	0	
P ₂	0	1	1	0	1	0	0	0	0	
P ₃	0	1	0	0	0	0	0	0	0	
P ₄	0	0	1	0	0	0	0	0	0	
P ₅	0	0	0	0	1	0	0	0	0	
P ₆	0	0	0	0	0	0	1	0	0	[ii]
P ₇	0	0	0	0	0	0	0	1	0	
P ₈	0	0	0	0	0	1	1	1	0	
P ₉	1	1	0	0	0	0	0	0	0	
P ₁₀	0	0	0	0	0	1	0	0	0	
P ₁₁	0	0	0	0	0	0	0	0	1	
P ₁₂	0	0	0	1	0	0	0	0	0	

Equation [ii] gives the marking matrix of the PN model. This matrix shows that at all times the number of markings in a place is never greater than one. This implies that the petri-net system is safe (1-bounded). Places represent conditions in this scenario. A condition is either true (represented by a token in the place) or false (represented by an absence of a token). A bounded petri-net could be realized in hardware and software, while a petri-net with an unbounded place could not in general be implemented. Figure (7-8) shows the reachability graph for the petri-net model. We can see from the graph that

from all the reachable markings of the system, a firing sequence (that contains every transition of the petri-net) exists. That implies that every transition of the petri-net is live and the system is deadlock free.

7.4 ESTABLISHING A CONNECTION FOR A MOBILE AGENT

7.4.1 SCENARIO DESCRIPTION

This scenario describes a mobile agent during one of its stops. It performs an action without the need for the connection to be established. When the mobile agent finishes its task, it may request to open a connection to proceed to its next stop. If the connection is already open, the mobile agent interrupts the current process and dispatches itself. If the connection is not established, the mobile agent opens a new one and dispatches itself.

7.4.2 LIFE-CYCLE MODEL

The life-cycle model starts by the agent requesting a connection. The agent interrupts the current process. If the connection is open, the agent moves to the next place, then the system resumes. If the connection is off, the agent establishes the connection then it moves to the next place. The following is the life-cycle model:

(Request a connection).(Interrupt while connection open|Interrupt while connection off).(Resume while connection open|Resume while connection off).

(Request a connection).(Interrupt while connection off).(Establish a connection).(Dispatch agent).(Resume while connection is off).

(Request a connection).(Interrupt while connection open).(Dispatch agent).(Resume while connection is open).

7.4.3 PETRI-NET MODEL

7.4.3.1 The net (places and transitions)

Figure 7-9 Petri-net model for establishing connection.

Places		Transitions	
ID	Description	ID	Description
P ₁	Preparing connection to be ready to dispatch the agent	T ₁	Dispatch agent
P ₂	Releasing the connection	T ₂	Interrupt
P ₃	Stopped while connection open	T ₃	Establish connection
P ₄	Stopped while connection off	T ₄	Resume connection off
P ₅	Checking if connection is needed	T ₅	Resume connection open

P ₆	Agent is done with its job and ready to move to the next place in its itinerary.	T ₆	Request a connection
P ₇	Connection off		
P ₈	Connection open		

7.4.3.2 Reachability Graph

Figure 7-10 Reachability graph for establishing connection.

$$M_0 = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1] \quad M_1 = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1]$$

$$M_2 = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0] \quad M_3 = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0]$$

$$M_4 = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0] \quad M_5 = [0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0]$$

$$M_6 = [1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \quad M_7 = [0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$M_8 = [1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \quad M_9 = [0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

7.4.3.2.1 Reachability matrix

	M ₀	M ₁	M ₂	M ₃	M ₄	M ₅	M ₆	M ₇	M ₈	M ₉	
P ₁	0	0	0	0	0	0	1	0	1	0	
P ₂	0	0	0	0	0	0	0	1	0	1	
P ₃	0	0	1	0	0	0	0	0	1	1	[iii]
P ₄	0	0	0	0	0	0	1	1	0	0	
P ₅	0	1	0	0	1	1	0	0	0	0	
P ₆	1	0	1	1	0	0	0	0	0	0	
P ₇	0	0	1	0	1	0	0	0	0	0	
P ₈	0	0	0	1	0	1	0	0	0	0	
P ₉	1	1	0	0	0	0	0	0	0	0	

Equation [iii] gives the marking matrix of the PN model. This matrix shows that at all times the number of markings in a place is never greater than one. This implies that the petri-net system is safe (1-bounded). Places represent conditions in this scenario. A condition is either true (represented by a token in the place) or false (represented by an absence of a token). A bounded petri-net could be realized in hardware and software, while a petri-net with an unbounded place could not in general be implemented. Figure (7-10) shows the reachability graph for the petri-net model. We can see from the graph that every transition of the petri-net is live and the system is deadlock free.

7.5 EMAIL MONITORING

7.5.1 SCENARIO DESCRIPTION

This scenario describes a monitoring process for email. A monitoring agent is installed to notify the user when he gets an email that satisfies a certain condition. The condition is set previously by the user. The action to be performed on the message is defined by the user as well. The monitoring agent checks the mailbox frequently. When the agent's timer fires the process

starts. If the new message matches the requirements, the specified action will be performed and the system will reset. Otherwise, the system will reset and wait for the timer to fire again.

7.5.2 LIFE-CYCLE MODEL

The life-cycle model starts when the timer fires. If the conditions the agent are monitoring are not satisfied, the timer will reset. If the conditions are satisfied then the agent will start to verify the conditions then it will carry out the associated action. The following is the life-cycle model:

(Timer fires).(Condition Satisfied|Condition unsatisfied)

(Timer fires).(Condition unsatisfied) .(Terminate)

(Timer fires).(Condition satisfied) .(Perform action) .(Finish & Restart)

7.5.3 INTERACTION DIAGRAM

Figure (7-11) illustrates the interaction diagram for the scenario. When the timer fires, the monitoring agent is activated. Then the monitoring agent accesses the new unread messages in the mailbox. If the messages match the requirements the agent performs the action.

Figure 7-11 Interaction diagram for a monitoring agent.

7.5.4 PETRI-NET MODEL

7.5.4.1 The net (places and transitions)

Places		Transitions	
ID	Description	ID	Description
P ₁	Condition	T ₃	Terminate
P ₂	Timer	T ₄	Timer fires
P ₃	Condition unsatisfied	T ₅	Proceed
P ₄	Checking condition	T ₆	Perform action
P ₅	Condition satisfied		
P ₆	Action		
P ₇	Ready for action

Figure 7-12 Petri-net model for monitoring email.

7.5.4.2 Reachability Graph

Figure 7-13 illustrates the reachability graph of the Petri-Net in Figure 7-12.

$$M_0 = [1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0] \quad M_1 = [0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0]$$

$$M_2 = [1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0] \quad M_3 = [0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0]$$

$$M_4 = [0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0] \quad M_5 = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1]$$

$$M_6 = [0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0]$$

7.5.4.2.1 Reachability matrix:

	M ₀	M ₁	M ₂	M ₃	M ₄	M ₅	M ₆
P ₁	1	0	1	0	0	0	0
P ₂	1	1	0	1	0	0	0
P ₃	0	0	0	1	0	0	1
P ₄	0	0	1	0	1	0	1
P ₅	0	1	0	0	1	0	0
P ₆	1	1	1	1	1	1	1
P ₇	0	0	0	0	0	1	0

[iv]

Figure 7-13 Reachability graph for monitoring email.

Equation [iv] gives the marking matrix of the PN model. This matrix shows that at all times the number of markings in a place is never greater than one. This implies that the petri-net system is safe (1-bounded). Places represent conditions in this scenario. A condition is either true (represented by a token

in the place) or false (represented by an absence of a token). A bounded petri-net could be realized in hardware and software, while a petri-net with an unbounded place could not in general be implemented. Figure (7-13) shows the reachability graph for the petri-net model. We can see from the graph that from all the reachable markings of the system, a firing sequence (that contains every transition of the petri-net) exists. That implies that every transition of the petri-net is live and the system is deadlock free.

7.6 UPGRADING A SERVICE

7.6.1 SCENARIO DESCRIPTION

This scenario describes a remote service upgrade mobile agent. The agent (SUA) was described in section 6.6. The mobile agent travels between places using his itinerary. When it reaches a place, it upgrades the service file by replacing all the occurrences of one specified term with another specific term.

7.6.2 LIFE-CYCLE MODEL

The life cycle-models start by distributing the upgrading agents. Each of the agents will carry out an upgrade file process, then it will kill itself. The following is the life-cycle model:

(Distribute agents)^{*}.(Start upgrading)^{*}.(Finish upgrading)^{*}. (Kill agents)^{*}.
Reset

7.6.3 INTERACTION DIAGRAM

Figure (7-14) is the itinerary graph for a mobile agent that reconfigures services. The provider creates clones and sends them in parallel to update the service. After the SUA is done with its job, it kills itself.

Figure 7-14 Itinerary graph for upgrading a service.

7.6.4 PETRI-NET MODEL

Places		Transitions	
ID	Description	ID	Description
P ₁	Agent at place A	T ₁	Distribute
P ₂	Agent at place C	T ₂ , T ₇ , T ₅	Start upgrading
P ₃	Agent at place B	T ₃ , T ₆ , T ₈	Finish upgrading
P ₅ , P ₈ , P ₁₁	File exists	T ₄	Dispose agents
P ₉ , P ₆ , P ₁₂	Performing upgrade		
P ₇ , P ₁₀ , P ₁₃	Idle		

7.7 SUMMARY

In this Chapter, we illustrated the idea of applying petri-nets to verify the safety and freedom of deadlocks of some main scenarios for AgentTINA. The scenarios presented are a subset of the actual scenarios. Petri-nets proved to be an appropriate tool to verify the correctness of agent-based systems. Also, Petri-nets provided a useful tool to visualize the dynamic behavior of mobile agents.

Chapter 8

Applications

Unified Messaging Using AgenTINA

***Raise new questions, explore new possibilities, regard old problems
from a different angle***

CHAPTER 8

APPLICATIONS: UNIFIED MESSAGING

In this Chapter, we use the AgentTINA architecture (Chapter 6) and the concepts defined in chapters 4 to develop a unified messaging application. The main goal of the chapter is to illustrate the practicality of using AgentTINA to develop real life applications.

8.1 WHAT IS UNIFIED MESSAGING?!

Today's businesses are continually flooded with a multitude of ways to communicate information--voice mail, fax mail, and electronic mail. Almost all workplaces use all three. As the pace of business increases, the number of types of messages gets larger. Finally, users process enough voice, fax, and electronic mail that technology is actually slowing them down.

The need for efficient ways to communicate information has resulted in a number of different messaging mediums. Each message type has its own

advantages and disadvantages and each requires a different terminal device for delivery and access--the telephone or desktop computer.

However, as the number of voice, fax, and electronic messages grows, the need for an improved way of managing them using a common set of management tools from a single terminal device grows. The main goal of a unified messaging application is to deliver users messages in the appropriate form at any place and any time.

8.2 WHY AGENTS IN UNIFIED MESSAGING?

There are many reasons that make agents a natural representation for unified messaging applications:

- **Flexibility:** as agents come and go at run time, users have the freedom of using endless combinations of message type and actions without hard-coding. A major advantage of using agents is that they provide simple mix-and-match in a seamless manner. For example, new message formats and devices can be added dynamically.
- **Robustness:** agents should be able to handle unexpected problems in a robust way. For example, if the monitor breaks down, the agent community should be able to find an acceptable way to deliver messages to users using the text-to-speech agent over the phone instead of email.
- **User delegation:** agents should be able to act on behalf of users in a smart way. For example, users want to be able to delegate a task to

the agent community without having to specify which agent will perform the task.

- **Parallel cooperation:** the unified messaging application requires a lot of parallel cooperation between its components. The agent community can perform certain tasks in parallel, if needed. An example to illustrate this issue will be presented in section 8.5.

8.3 MAIN AGENTS, PLACES AND CLASSES

In this section, we present the main agents of the unified messaging application. We use the process described in Chapter 4 to identify and categorize agents. Agents communicate through the Facilitator who knows the capabilities of each.

8.3.1 PCA

The Personal Communication Agent (PCA) manages the users interactions with the application. PCA runs on the user's laptop or PDA and here he may use the telephone to access his remote applications. Input can be done using voice or menu commands. PCA's main responsibilities are discussed in details in section 6.3.

8.3.2 UTILITIES (CONVERSION AGENTS)

Conversion agents are provided by service providers. They are mainly used to transform messages from one format to another. Conversion agents were described in section (6.7). Figure (8-1) illustrates the hierarchy of conversion agents. In the unified messaging application the following conversion agents are needed to accept and deliver files in text or voice format.

8.3.2.1 Text-Speech Agent

This agent converts a text string to an utterance through a speech synthesis facility. An example of Text-Speech Agent was introduced in section 6.7.

8.3.2.2 Fax-Text Agent

This agent takes an encoded fax message as input and decodes it to an image of 1's and 0's. The text on the image is then recognized through optical character recognition.

8.3.2.3 Video-Audio Agent

This agent streams the audio from an encoded video. The audio is represented as a WAVE file so that it may be used in standard messaging environments and be digitally encoded.

Figure 8-1 Converter hierarchy.

8.3.3 CONNECTION AGENTS

Connection agents are provided by service providers. They provide connections that the PCA may require to arrive at the user in a certain format.

Figure (8-2) shows hierarchy of connection agents. Examples include:

8.3.3.1 WWW Connector

This agent provides on demand access to the WWW. It may use either Netscape or Microsoft Explorer as the web connection based on the user's preferences.

8.3.3.2 Pager Connector

This agent provides access to a 2-way paging connection on demand. As a secondary function, the pager connector can also send a page command with a set of digits to an external paging service.

8.3.3.3 Phone Connector

This agent provides an interface to a phone dialer. Phone dialers were introduced in section 6.5.

Figure 8-2 Connector hierarchy.

8.3.4 NETWORK ELEMENTS AGENTS

Network elements agents are provided by network vendors. They are used to manage network resources since they represent network elements. The ultimate goal of this type of agents is to add robustness and ease of use to the system. Figure (8-4) shows a snapshot of the setup interface. Elements can be

added and removed from the network without affecting the system as a whole. Each element has an agent that handles its interaction with the messaging application. Figure (8-3) displays the hierarchy of device agents.

Figure 8-3 Device agent hierarchy.

Figure 8-4 Setup Interface.

8.3.5 FACILITATOR

The Facilitator's main function is to act as a blackboard for communication between agents. Facilitators were previously described in section 6.2. In this application, Interface agents send requests to the Facilitator and wait for the appropriate results. Examples include message delivery and formatting.

8.3.6 DATABASE AND CALENDAR AGENTS

The database agent interfaces with a directory database containing users information. This agent is classified as an information agent. Figure (8-5) gives a snapshot of the database agent. The calendar agent can report on where a person might be, or when they might be performing a particular action (see Figure 8-5).

Agenda Demo

Set Up Manage View Print

Notification File View Help

Personal

Name: Zeid

Office: 5555

Email: mazreese@sprint.ca

Phone: (613)224-5951

Fax: (613)224-6629

Pager: (613)277-2647

Calendar

June 1999

25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Today: 06/08/99

Time: 10:00

Location: 1234 Herzberg

View Calendar For Agent Update

Figure 8-5 Calendar Agent interface.

8.3.7 NOTIFY

The notify agent is responsible for sending information, either textual or audio, to a specific user. This agent has the knowledge of how a user is to be contacted; it may request help from the mail agent, text-to-speech agent, or the calendar agent. The notify agent might instruct the phone agent to call the user to read certain messages. Figure (8-6) illustrates a Notify agent interface.

Figure 8-6 Notify Agent interface.

8.3.8 TRIGGERS

Triggers fire when a certain condition is satisfied in the form: when (condition, action). When the condition is satisfied, the action is performed. The use of triggers is shown in the example in section 8.5.

8.4 RELATIONSHIPS BETWEEN AGENTS AND CLASSES

Figure (8-7) shows the relationships between agents described in section 8.3. The model is constructed using the proposed notation described earlier in Chapter 4 (Figure 4-14, 4-15). Figure (8-7) shows also the main classes and

their relations to agents. As shown in Figure (8-7) agents are associated to the Facilitator as it acts as a blackboard.

Figure 8-7 Relationships between agents and classes.

8.5 SCENARIO

A traveler is trying to reach back and access his calendar, database and email applications. These applications are running on different machines. In this scenario, the user orders the system to notify him once he gets an email with a certain subject. When the email arrives to the mailbox, the email agent detects it using a trigger, then it sends a request to the Facilitator agent who

consequently tries to locate the user by contacting the database and calendar agents in parallel. Finally, the phone number of the room where the user is located, is returned. The email message is then converted from text format to voice format using the text-to-speech agent. The phone agent calls the user and verifies his password. The password is acquired from the database through the database agent. Finally, the message is delivered in voice format.

The following is the sequence of messages and agents communication to achieve the scenario (Figure 8-7):

1. *PCA* Microphone button is pressed to request speech recognition.
2. *The Speech recognition agent* converts audio form to text.
3. *The Natural Language agent* converts text to the form: when(condition, action) and route the request to the Facilitator.
4. A *trigger* is added to the email agent in order to monitor or keep track of information in behalf of the user. The trigger is in the form: when mail arrives notify me. The syntax of the trigger is as follows: Condition: test arrives (mail,args(for ("Amir Zeid"), about (**thesis**))), Action: notify ("Amir Zeid").
5. When an email arrives at the mailbox of the user with the subject **thesis**, the *trigger* fires.
6. The *Notify agent* tries to locate the user by sending a request to the Facilitator. *The Calendar and database agents* respond in parallel to the request.

7. The required message format is determined according to the feedback of the calendar agent. In this example, it is an audio format. The telephone number where the user is located, is returned from the calendar agent.

Figure 8-8 Scenario for message handling.

8. The *Text-to-speech agent* converts the message to wave format.
9. The *Database agent* returns the phone number and password of the user.

10. *The phone agent* calls the user, verifies his password then reads the message to him.

8.6 RELATED WORK

Related work in unified messaging is being investigated in British Telecom labs where they lead agent-based telecommunications research projects. They are currently investigating and developing several different aspects of agent technology including collaborative agents, personal agents, information agents and mobile agents. They have developed ZEUS, a tool-kit of components which designers can extend and integrate to fit a specific application. ZEUS also provides advanced visualization, statistical and debugging facilities. Using ZEUS to develop various applications including messaging applications is the next logical step of British Telecom research labs [Crab-96]. OAA (Open Agent Architecture) is an agent-based framework that can be used to develop various applications integrating such diverse technologies as image processing, speech recognition, multi-user collaboration, text extraction and virtual reality [Martin-96]. One of the main goals of the SPIN(Seamless Personal Information Networking) group of the Canadian National Research Council is to develop messaging applications to achieve end-to-end personalized communication of information in distributed workspaces. The first generation was implemented using Lotus Notes with a gateway to the world wide web [Impey-98].

8.7 SUMMARY

As the world becomes more network reliant, we need new architectures for constructing applications. The unified messaging application, we introduced in this chapter, is a dynamic community of agents running on different machines, written in different programming languages interacting, cooperating with each other on behalf of the user to perform his request at a high level. We have seen that agents can be used to transparently access numerous remote systems either in a directed or delegated manner.

Chapter 9

Conclusion

Concluding Remarks & Suggestions for Future Research

Life is the art of drawing sufficient conclusions from insufficient premises
Samuel Butler

CHAPTER 9

CONCLUSION AND FUTURE DIRECTIONS FOR RESEARCH

In this Chapter, we conclude our thesis by summarizing the features of the software engineering process we proposed in Chapter 4. The main features of the AgentTINA architecture are then discussed. Finally, we conclude by proposing related future research to enhance this thesis.

9.1 CONCLUSION

To foresee the future of a technology is always a difficult task. To foresee the future of agent technology is even more difficult. In fact, the word "agent" denotes a wide set of concepts and no exact definition of agency can be described. The set of software agents appears to be a "fuzzy" set and, instead of saying whether a piece of software belongs or does not belong to this set, it makes more sense to say that a piece of software belongs "to a certain degree" to this set.

Agent technology is a fertile research area, and when such a research area is fashionable, it can easily become an overworked concept. Thus it sometimes happens that new "agent oriented" terms are used to express old meanings and that some agents are actually not very much more than objects or modules. Everyone who wants to approach the "agent world" should be aware of people taking advantage of the fashionable status of agents at the moment, and also of the over-hyping that often comes with agent advertising.

Agent-based systems have shown much promise for flexible, fault-tolerant, distributed problem solving. With an increasing number of new computers being added to the Internet every day, a paradigm shift for computing is well under way, one which moves away from requiring all relevant data and programs to reside on the user's desktop computer. The data now routinely accessed from computers spread around the world has become increasingly rich in format, comprising multimedia documents, audio and video streams, and with the popularization of Java, may include programs which can be downloaded and executed on the local machine. As the world continues to evolve towards a more networked computing model where remote computers take on an expanded role, not only for storing public data but also for providing services, we will need new programming models which allow for flexible composition of distributed processing elements in a dynamically changing and relatively unstable environment. Agents seem to be one of the possible paradigms that can meet future computing needs.

Our dissertation has aimed to tackle two main problems related to agent-based development. The first part was dedicated to develop a software engineering process for agent-based development, while the second part was

dedicated to investigate the application of agents to enhance one of the main future telecommunications architectures (TINA) and to design AgenTINA.

9.1.1 MAIN FEATURES AND CONTRIBUTIONS OF AGENT-BASED SOFTWARE ENGINEERING PROCESS

The first goal of this thesis is to develop a software engineering process for agent-based systems. The method consists of concepts, techniques, and steps to implement a process. We extended the notations of UML to provide adequate tools to model static and mobile agents. Design verification is done utilizing Petri-Nets.

The agents-based software engineering process we proposed represents a contribution based on the following features:

- The process has a step-by-step guidelines from analysis to design to verification
- The process has a modified set of notations based on the Unified Modeling Language (UML) to model agents (static and mobile). UML is the standard OO modeling notation.
- The process contains notations to model static and dynamic behavior of agent-based systems.
- The process is open and adaptable. New notations for types of agents can be added.
- The process maintains a clear and consistent mapping from analysis to design to verification.

- The process is easy to learn and use.
- Petri-nets are used to verify the freedom of deadlocks and safeness of the design.

9.1.2 MAIN FEATURES AND CONTRIBUTIONS OF AGENTINA

The main goal of AgentTINA's design is to enhance TINA utilizing agents (static and mobile). The AgentTINA architecture is divided into the following environments: Personal Communication Agent (PCA) Management Environment, Connection Management, Utilities Management, Service Management and Shopping Environment. AgentTINA represents a contribution based on the following features:

- AgentTINA is TINA-compliant. It is an enhancement to one of the future telecommunications standards. AgentTINA satisfies TINA's requirements and adds some features like:
 1. Negotiation for services is done using agents.
 2. Automatic shopping for services and devices is provided using mobile agents.
 3. Reconfiguration of services is done using mobile agents.
 4. Introducing new services, users and devices is done with minimal human efforts utilizing agents.
 5. Monitoring email and web pages can be easily done utilizing AgentTINA.
- AgentTINA is based on FIPA design guidelines. FIPA is one of the main agent standards bodies.
- AgentTINA provides code generation facility for the Microsoft agent.

- AgentTINA is flexible. Many applications can be modeled using the AgentTINA architecture.

9.1.3 NEW (UNIQUE) CONCEPTS

Although our research incorporates several ideas from other tools and methodologies, it is *unique* in the following ways:

- The notion of *place* is introduced to model a host that accepts and processes mobile agents.
- The notion of *Neighborhood diagrams* was introduced to group places that reside on the same network neighborhood.
- *Itinerary Graphs* are used to model the dynamic behavior of mobile agents. They are a visualization of mobile agents movements from one place to another.
- We introduced the *Lifecycle* model for agents. Life-cycle expressions define patterns of communication.
- We used *petri-nets* to verify and visualize mobile agent behavior. Petri-nets are well-known for modeling various types of systems. We adapted them to model agents behavior and communication.
- AgentTINA is a *realization* of TINA using *agents* (static and mobile).
- *Monitoring* email and associating various actions to monitoring like negotiation, shopping and reading email make use of the flexibility agents provide. New actions can be easily added and associated to the monitoring process.

- Monitoring *negotiations* between PCA/SPA provides a visualization of agents dynamic behavior.
- *Distributed service update* saves downloading and uploading of the files and distributes the load of update to multiple servers.

9.1.4 PROBLEMS AND OBSERVATIONS

Throughout our research we encountered many problems. We had some useful observations as well. Some of the problems and observations are:

- Agent technology is still relatively new. As with any new technology, before applying in business, the benefits and implications, including skill requirements and network infrastructure, must be understood. However, our research has shown that for particular types of applications the technology is ready. There is very little doubt that agent technology will be increasingly used in many areas of the industry.
- Many mobile Java-based agent platforms are now available. Many of these have been through several releases and are now adequately stable.
- Interoperability among the agents/heterogeneous platforms (i.e. agreement on basic communication protocols, register of all the agent addresses) is harder than it seems. Also, lack of technical support for the mobile agent environments was a real challenge.

- Although intuitively a natural progression from object-orientation, the agent paradigm is sufficiently different to make it difficult for traditional developers to think in an agent-oriented way.
- Agent technology allows the development of new lines of telecommunication services incorporating new features such as autonomy and negotiation, which improve substantially the functionality of present services. Service providers commercializing agent-based services incorporating intelligent interfaces might target their potential customers more effectively. By offering better products than their competitors they might increase the market share maximizing revenues.
- The ability for Agent-based system to negotiate network resources facilitates the optimization of network resources in a competitive framework. The user may obtain the best quality cost ratio for the service from different service providers while service providers may obtain the best cost from different network providers.
- Network providers may optimize network resources and profit adapting dynamically the fees of the offered resources according to demand.
- Petri-nets prove to be a good tool to model agents conversations and negotiations. Petri-nets provide a concrete model to prove the correctness of the design of agent-based systems.

9.2 DIRECTIONS FOR FUTURE RESEARCH

Some directions regarding agents technology are discussed in this section. Some of the future predictions are related to agent-based systems in general, while others are more specific to our research.

9.2.1 WEARABLE COMPUTERS

Currently, there is good deal of serious research being carried out at prestigious institutes such as IBM, Xerox, and MIT, in the area of wearable computers. Clothing-based computing will blur the boundaries between seeing and viewing and between remembering and recording. The application of agents in wearable/smart clothes will include many of the following:

- Coordinating all input/outputs: voice, video, sounds and so on.
- Interacting with the wearer's other computers, laptops, desktops, by sending and receiving data.

9.2.2 VIRTUAL CORPORATIONS

Five years in the future, the global corporate landscape will be totally different from that of today. We may define the virtual corporations as "a legal enterprise or corporation that conducts business providing services, products, or other goods with little or no physical presence and has a set of strategic alliances with other business entities." We predict that leading corporations will develop virtual intelligent agents that will be able to analyze, compile, and develop future products on the basis of the data they find within the corporation. To achieve this, some mobile agents will perform a number of processes such as the following:

- They will scan and collect data on their own internal processes, products and services to identify new opportunities, markets and potential risks.
- They will establish huge knowledge pools of products, in order to filter, interpret, and present data to management in new ways.

9.2.3 AGENT STANDARDIZATION

There are several areas of mobile agent technology that should be standardized now to promote interoperability like agent management, transfer, naming and location syntax. There are other aspects that should be standardized, but not until the industry is more mature. The issues for future consideration includes interoperability between mobile agents environments. Today's mobile agent systems use several different languages. Therefore, the effort to convert from one agent encoding to another is too complex. When the serialization formats for agent code and execution state are similar, it should be possible to build standard bridges between different agent system types. Security issues have to be standardized. It is vital that the security issue is resolved before mobile agents will be deployed widely.

9.2.4 AGENT-BASED SOFTWARE ENGINEERING

Some of the main issues that can be investigated in this area are:

- Agent-based analysis/design tools based on the process proposed in chapter 4 can be developed.

- Automatic verification of design can be done using a tool that converts life-cycle model to a Petri-Net and verify its correctness and freedom of deadlocks.
- Experiment with *metrics* for mobile agents. Since mobile agents are dynamic by nature, then dynamic metrics should be applied to prevent the abuse of using agents. An example would be to measure the injection rate of agents in an open environment to prevent agents from flooding the network.
- The notion of *design patterns* was introduced in the OO paradigm for many reasons. One of the main goals of patterns is to reuse design instead of just reusing code. Another goal of patterns is to provide a common design vocabulary. We believe that the same argument can be applied to the new paradigm of mobile-agents. There are some reported patterns for mobile agents [Lang-98], yet we believe the field is still open for further investigation.

9.2.5 AGENT-BASED TECHNOLOGY IN TELECOMMUNICATIONS MARKETPLACE

A possible model for agent-based technology in telecommunications marketplace can be divided into three parts:

9.2.5.1 Service Providers

The service providers are the companies, which will offer agent services to consumers and businesses. The service providers will typically configure and apply an agent-based/enhanced software package to a problem domain. Currently, the most popular application has been for building user profiles.

The services will be developed by connecting together knowledge from a set of agent-enabled resources. The value that the service provider will add will be the way in which these resources are mapped together.

9.2.5.2 Product Development

Product developers will wrap a particular resource in an agent shell to allow service providers to use their product. These developers are analogous to today's component programmers who develop API's for use by programmers. However, the agent technologies used will describe the components in such a way as to make it easier for service providers to tie them together. There is much research interest in ways of allowing service providers to access these components dynamically in order to provide the best service to their customers. This may mean that the components will need to scale well as demand rises and falls.

9.2.5.3 Agent Technology Developers

Supporting the above categories, the agent developers will provide the infrastructure and methodologies to facilitate product development. These methodologies will span a number of product developers, allowing components developed independently to work together. Mobile agents have received much attention over the last few years. Despite providing an intuitively flexible modeling paradigm, to date, situations where mobile agents give advantage have been very specialized. The reliable management of mobile agents and their security implications are also causes for concern. Once solutions to these concerns have been overcome, then mobile agents will become part of the methodologies and development environments.

9.2.6 AGENT ORIENTED SERVICE DEVELOPMENT

New agent-based services might be quickly developed by selecting agents and adapting the necessary components such as reasoning, communication, resources, etc. in achieving the service functionality. The cost of service maintenance and service evolution is minimized by the use of declarative control mechanisms.

Deliberative agent architectures based on symbolic processing are suitable for implementing intelligent agent functionality such as deduction, learning, explanation and fault tolerance. Agents provide a good model for negotiation and communication.

9.2.7 AGENTINA AND AGENT-BASED TELECOMMUNICATIONS ARCHITECTURES

Some of the main issues that can be investigated in this area are:

- More applications using AgentINA can be investigated. A multi-lingual messaging system is a possible interesting application. Translation agents can be added to provide interface and messages in different languages.
- New state of art equipment like palmtops can be investigated. Agents for such devices should have new characteristics.
- More intelligence and negotiation strategies can be added to the shopping and service environments.
- Generating code in different languages and scripts can be added in the Text-to-Speech agent (Section 6.7).

- Creating more agents to monitor different events in the service management subsystem can be added (Section 6.6).
- Automatic service provisioning can be investigated. We believe that the next generation of network services should be provided in a plug-and-play fashion.

References

References

Agents

- [Aglets-99] <http://www.trl.ibm.co.jp/aglets/>
- [Ash-96] www.ashland.edu/~beisert "Intelligent Agents"
- [Aub-95] Aubrey, David. "Nomads of the Net." *Computer Shopper* Dec 1995: 616-620.
- [Aus-96] www.aaii.oz.au "Agent oriented versus object oriented."
- [Bak-96] Baker, A.D., "Chapter 8 - Metaphor or Reality: A case study where agents bid with actual costs to schedule a factory," *Market-based Control*, Scott H. Clearwater (ed.), World Scientific, pp. 184-223, 1996.
- [Bec-85] Beck, C.L., "Modeling and Simulation of Flexible Control Structures for Automated Manufacturing Systems," Masters Thesis, Electrical and Computer Engineering Department, Carnegie-Mellon University, 1985
- [Big-98] Bigus, J. *Constructing Intelligent Agents with Java*. Wiley Computer Publishing, New York, 98.
- [Buhr-98] R.J.A. Buhr, M. Elammari, T. Gray, and S. Mankovski (1998), A High Level Visual Notation for Understanding and Designing Collaborative, Adaptive Behaviour in Multiagent Systems. In: *31st Annual Hawaii International Conference on System Sciences (HICSS'98)*, Hawaii, USA, January 1998.
- [Cag-97] Caglyan, A. *Agent Source Book*. Wiley Computer publishing, New York, 97.
- [Cha-96] Chaib-draa, B., Moulin, B., Mandiau, R. & Millot, P., "Chapter 1 - Trends in Distributed Artificial Intelligence," *Foundations of Distributed Artificial Intelligence*, G. M. P. O'Hare and N. R. Jennings (eds.), John Wiley & Sons Inc., pp. 3-55, 1996.
- [Cha-87] Chaib-draa, B., & Moulin, P., "Architecture for Distributed Artificial Intelligent Systems," *IEEE Proceedings*, Montreal, pp. 64-69, 1987.
- [Crab-96] Crabtree, B. *Agents, mobility and Multimedia information*. Proceedings of PAAM 96, London, England.
- [Dav-92] David R. & Alla H. (eds.), *Petri Nets and Grafset - Tools for modelling discrete event systems*, Prentice Hall, 1992.
- [Dib-96] Dibbell, Julian. "Smart magic" *Time* July 1, 1996: 18-24.
- [dMars-96] dMars, <http://www.aaii.oz.au/proj/dMARS-prod-brief.html>, by Australian Artificial Intelligence Institute (AII), 1996.

- [Eurescom-99] <http://www.eurescom.de/public/projects/P700-series/P712.HTM>
- [FIPA-96] FIPA Rationale, http://www.csel.stet.it/fipa/fipa_rationale.htm, 1996.
- [FIPA-97] FIPA 97 spec, <http://www.csel.stet.it/fipa/spec/fipa97reston.htm>, FIPA '97 specification - Reston draft (ver 1.0), 19 April 1997.
- [FIPA-99] <http://drogo.csel.stet.it/fipa/>
- [Fri-94] Frivold, R., Lang, R., & Fong, M., "Extending WWW for Synchronous Collaboration," Electronic Proceedings of Second WWW Conference, Phoenix, pp. 559-583, 1994.
- [Gai-96] Gaiti, D. "A proposal for intelligent management in the intelligent network"
- [Gen-94] Genesereth, Michael. "Software agents." *Communications of ACM* July 94: 48-53
- [Hur-95] Hurwicz, Mike. "Intelligent Management Agents." *Distributed Computing Monitor* July 95: 3-18.
- [Impey-98] R. Impey, S. Abu-Hakima, R. Liscano. A Common Multi-Agent Testbed for Diverse Seamless Personal Information Networking Applications. *IEEE Communication Magazine*. July 1998.
- [Jen-95] Jennings, N. R., "Chapter 6 - Coordination Techniques for Distributed AI," *Foundations of Distributed Artificial Intelligence*, G. M. P. O'Hare and N. R. Jennings (eds.), John Wiley & Sons, pp. 187-210, 1995.
- [KGR-96] D. Kinny, M. Georgeff, A. Rao. "A Methodology and Modeling Technique for Systems of BDI-Agents" in *Proceedings MAAMAW 96*, Springer, Berlin 1996.
- [Maes-91] Maes, P. (ed), *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, London, The MIT press, 1991.
- [Maes-94] Maes, P., "Agents that Reduce Work and Information Overload," *Communications of the ACM* 37 (7), pp. 31-40, 1994.
- [Mag-95] Magedanz, Thomas. "On the impacts of intelligent agent concepts on future telecommunication environments."
- [Martin-96] Martin, David. Development tools for open agent architecture in *Proceedings of PAAM 96*, London, England.
- [Mcg-96] www.ee.mcgill.ca "What is agents?"
- [Met-96] Metcalfe, Robert. "Future looks surprisingly bright for cross-platform, object oriented Java." *Infoworld* May 13, 1996: 54-55.
- [Mul-94] Muller, J. P., "A Conceptual Model for Agent Interaction," *Proceedings of the 2nd International Working Conference on Cooperative Knowledge Based Systems (CKBS-94)*, Deen, S. M. (ed.), Keele University: Dake Centre, pp. 213-234, 1994.
- [Nwa-96] Nwana, H.S., "Software Agents: An Overview," *Knowledge Engineering Review*, Vol. 11, No 3, Cambridge University Press, pp.1-40, Sept 1996.
- [Nob-69] Noble, Ben, "Chapter 4 -- Basis and Dimensions," *Applied Linear Algebra*, Prentice -- Hall, Inc., pp. 95-126, 1969.
- [NWW-96] Network World. "Top 10 Technologies that will shape the next 10 years"
- [OG-92] A. Ovalle, C. Garbay. "Towards a Methodology for Multi-Agent System Design" in: *Proceedings Expert Systems 92*, Cambridge, 1992.
- [Orf-96] Orfali, Robert et al. *The Essential Client/Server Survival Guide*. USA: Wiley, 1996.

- [OW-92] G.M.P. O'Hare, M.J. Wooldridge. "A Software Engineering Perspective on Multi-Agent System Design: Experience in the Development of MADE" in Distributed Artificial Intelligence - Theories and Practices, Kluwer Academic, Dordrecht, 1992.
- [Pal-99] Pallmann, D. Programming Bots, Spiders and Agents in MS Visual C++. Microsoft Press, USA 99.
- [Par-96] Parunak, H. Van Dyke, "Go to the Ant: Engineering principles from Natural MultiAgent Systems," Annals of Operations Research (submitted), 1996.
- [Pav-95] Pavlou, George. "Intelligent remote monitoring."
- [Pont-96] Pont M.J, & Moreole, E., "Towards a practical Methodology for Agent Oriented Software Engineering with C++ and Java," Technical Report 96-33, Department of Engineering, Leicester University, Dec. 1996.
- [Procura-99] <http://www.sce.carleton.ca/netmanage/perpetuum.shtml>
- [Smi-80] Smith, R. G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers C29 (12), pp. 1104-1113, 1980.
- [Smi-96] Smith, R., "Software Agent Technology," Proceedings of The First International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology, London, UK, pp. 557-571, 1996.
- [Til-96] Tilley, K.J., "Chapter 9 - Machining Task Allocation in Discrete Manufacturing Systems," Market-based Control, Scott H. Clearwater (ed.), World Scientific, pp. 224-251, 1996.
- [Vas-95] Vassila, Anastasia. "Introducing active managed objects for effective and autonomous distributed management."
- [Vis-92] Viswanadhan, N. & Narahari, Y. (eds.), "Chapter 5 - Petri Net Models," Performance Modeling of Automated Manufacturing Systems, Prentice Hall, pp. 473-580, 1992.
- [Voyager-99] Voyager, ObjectSpace, "<http://www.objectspace.com/voyager>".
- [Way-95] Wayner, Peter. "Free agents." Byte March 1995.
- [Woe-95] Woelk, Darrell. "Uncovering the next generation of active objects." Object Magazine August 95: 33- 40.
- [Whi-96] White, James. "Mobile agents." Web Techniques May 96: 37-47.
- [Wol-96] Wolfe, Alexander. "Secret agents." Electronic engineering times Feb 12, 1996: 51.
- [w3-96] www.w3.org. "Mobile Code" Survey of Agent oriented languages.
- [Yam-96] Yamada, Ken. "General Magic pulls new plan from hat." Computer reseller news May 6, 1996: 79-81.

Mobile Agents

- [BHS-93] B. Burmeister, A. Haddadi, K. Sundermeyer,: "Generic Configurable Cooperation Protocols for Multi-Agent Systems" in: C. Castelfranchi, J.-P. Müller (eds.): From Reaction to Cognition. Proc. MAAMAW'93, LNAI 957, Springer, Berlin, etc., published 1995.
- [CON-92] ``Methodology Report (D13)"', British Aerospace, Durham University, Cambridge Consultants, 1992.
- [Col- 94] D. Coleman, P. Arnold, S. Bodooff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes: Object-Oriented Development. The Fusion Method, Prentice Hall, Englewood Cliffs, 1994.
- [Dor-93] J. Doran: "Using DAI Software Testbeds" in: Proceedings CKBS'93, DAKE Centre, Keele, 1993.
- [FWW-94] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, C. Beck: ``Specification of the KQML Agent-Communication Language" (Draft), The DARPA Knowledge Sharing Initiative, External Interfaces Working Group, 1994.
- [Gar-98] Garijo, F. et al. Development of a Multi-Agent System for cooperative work with network negotiation capabilities, IATA 98, Paris.
- [Had-96] A. Haddadi: Communication and Cooperation in Agent Systems: A Pragmatic Theory, LNAI 1056, Springer, Berlin, etc., 1996.
- [Jac-92] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard: Object-Oriented Software Engineering. A Use Case Driven Approach, ACM Press/Addison-Wesley, 1992.
- [Kin-93] D. Kinny: ``The Distributed Multi-Agent Reasoning System Architecture and Language Specification", Australian Artificial Intelligence Institute, Melbourne, 1993.
- [KGR-96] D. Kinny, M. Georgeff, A. Rao: ``A Methodology and Modelling Technique for Systems of BDI-Agents" in: W. van der Velde, J. Perram (eds.): Agents Breaking Away. Proc. MAAMAW'96, LNAI 1038, Springer, Berlin, etc., 1996.
- [Lang-98] Lange, D. Programming and Deploying Java Mobile Agents with Aglets. Addison Wisley, New York, 98.
- [Msa-99] Microsoft Agent, www.microsoft.com.
- [Mey-88] B. Meyer: Object-oriented Software Construction, Prentice Hall, Englewood Cliffs, 1988.
- [OG-92] A. Ovalle, C. Garbay: ``Towards a Methodology for Multi-Agent System Design" in: Proceedings Expert Systems 92, Cambridge, 1992.
- [OW-92] G.M.P. O'Hare, M.J. Wooldridge: ``A Software Engineering Perspective on MultiAgent System Design: Experience in the Development of MADE" in: N. Avouris, L. Gasser (eds.): Distributed Artificial Intelligence - Theory and Praxis, Kluwer Academic, Dordrecht, etc., 1992.
- [Rom-91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorenson: Object-Oriented Modelling and Design, Prentice Hall, Englewood Cliffs, 1991.

-
- [RG-92] A. S. Rao, M. P. Georgeff: "An Abstract Architecture for Rational Agents" in B. Nebel, C. Rich, W. Swartout (eds.): Proc. International Conference on Principles of Knowledge Representation and Reasoning (KR-92), Morgan Kaufmann, San Mateo, 1992.
- [RZ-94] J. S. Rosenschein, G. Zlotkin: Rules of Encounter. Designing Conventions for Automated Negotiation among Computers, MIT Press, Cambridge MA, London, 1994.
- [Sho-90] Y. Shoham: "Agent Oriented Programming", Stanford University Technical Report STAN-CS-90-1335, Stanford, 1990.
- [Zeid-97] Zeid, A. Towards Software Engineering for Mobile Agents, European Conference on Object Oriented Programming (ECOOP), Jyvaskyla, Finland.

Intelligent Networks and TINA

- [Amb-89] Amborsh, W., et al. The intelligent network: a joint study by Bell Atlantic, IBM and Siemens. Germany: Springer Verlag, 1989.
- [App-94] Appleby, S. "Mobile software agents for control in telecommunications networks." BT Tech Journal April 94:104-113.
- [Bay-92] Bayliss, P. Intelligent networks: the path to global networking. Netherlands: IOS Press, 1992.
- [Bha-96] Bhattacharjee, B. "An Architecture for Active Networking" 1996. Technical Report GIT-CC-96-20, College of Computing, Georgia Institute of Technology.
- [Bru-96] Bruegge, Brend. "Applications of mobile computing and communications." IEEE personal communications Fe9 1996: 64-68.
- [Erc-96] Ericson WWW. "Network services- intelligent networks."
- [For-94] Forman, George. "The challenges of mobile computing." IEEE Computer: April 94: 38-47.
- [Gar-95] Garlan, David. "Research directions in software architecture." ACM Computing Surveys June 95:256-260.
- [Har-96] Hartman, J. "Liquid Software: A New Paradigm for Networked Systems." 1996, Technical Report 96-11, Department of Computer Science, University of Arizona.
- [Hey-95] Heymann, Leonard. "Mobility: a managerial dilemma." LAN Times Oct 1995: 57.
- [Lai-95] Laitinen, Mikko. "Integration of intelligent network services into future GSM networks." IEEE Communications Magazine June 1995:76-86.
- [LAN-95] Successful remote implementation, LAN Time, Oct 95.
- [New-96] "Advanced intelligent networks to be \$6 Bil market." NewsBytes July 24th 96.
- [Rei-94] Reinhardt, Andy. "The networks with smarts." Byte October 94:51-64.
- [Rob-96] Robertson, Bruce. "Driving applications on the network." Network Computing Jan 96: 68-76.
- [Ten-97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, pp80-86. January 1997.
- [TINA-97] Tina service architecture, TINA-C consortium, Feb 97.
- [TINA-99] www.tinac.com
- [Tho-94] Thorner, Jan. Intelligent Networks. MA: Artech House, 1994.
- [Yan-96] Yan, James. "Dimensioning network resources for IN services." Computer Networks and ISDN Systems 28, 1996: 627-633.
- [Yas-95] Yasin, Rutrel. "New agents distribute management." CommunicationsWeek Nov 13 1995.
- [Yem-96] Yemini, Y. "Towards Programmable Networks." 1996 in IEEE International Workshop on Distributed Systems. Oct 1996.
- [Zeid-97a] Zeid, A. Towards Plug-and-Play Networks. Proceedings of The 22nd International Conference on Computer & Industrial Engineering (ICC&IE97).
- [Zeid-97b] Zeid, A. Towards Plug-and-Play Networks: an Agent-Oriented Approach for TINA Services, OOPSLA 97, Doctoral Symposium, Atlanta, Georgia.
- [Zeid-98] Zeid, A. A Multi-Agent Testbed and an Agent Launch Tool for Diverse Seamless Personal Information Networking applications, (ICMAS'98), Paris, France.

Object-Oriented Analysis/Design

- [Aks-96] Aksit, Mehmet. "Load balanced mapping of distributed objects to minimize network communications." Journal of Parallel and Distributed Computing June 96:117-136.
- [Alhi-98] Alhir, Siman. UML in a Nutshell. USA: O'Reilly. 1998.
- [Boehm-81] Boehm. Software Engineering Economics. Englewood Cliffs, NJ Prentice Hall, 81.
- [Gam-95] Gamma, Erich et al. Design Patterns. USA: Addison-Wesley, 1995.
- [Gol-95] Goldberg, Adele, et al. Succeeding with Objects. New York: Addison Wesley, 1995.
- [Gra-95] Graham, Ian. Migrating to Object Technology. USA: Addison-Wesley, 1995.
- [Hen-94] Henderson-Sellers, Book two of OO knowledge. Australia: Prentice Hall, 94.
- [Hen-95] Henderson-Sellers, Comparison of object-oriented and traditional systems development issues in distributed environments, Information and Management, 1995.
- [Hen-96a] Henderson-Sellers, B. et al. "Incorporation of distributed Computing Concerns into Object-Oriented Methodologies." Journal of Object Oriented Programming June 96: 12-20.
- [Hen-96b] Henderson-Sellers, B. et al. "Expanding the Moses Methodology to Distributed Systems." Journal of object-oriented programming July 96: 39-46.
- [Hut-94] Hut, Andrew. Object Analysis and Design. USA: OMG, 1994.
- [Jac-92] Jacobson, Ivar et al. Object Oriented Software Engineering. USA: Addison-Wesley, 1992.
- [Orf-96b] Orfali, Robert et al. The Essential Distributed Objects Survival Guide. USA: Wiley, 1996.
- [Pur-91] Purtilo, James. "Dynamic reconfiguration of distributed problems." IEEE 1991: 560-570.
- [Rational-99] <http://www.rational.com/uml/resources/documentation/summary/index.jtmpl>
- [Selic-98] Selic, B, et al. Using UML for Modeling Complex Real-Time Systems. Technical Report, The objectTime Limited.
- [Svn-90] Svensson, Andres. "History, an intelligent load sharing filter." IEEE 1990: 546-553.
- [Two-95] Two-tier versus three-tier apps, Information week, Nov 1995.
- [Usz-96] Uszok, Andrzej. "A tool for monitoring software-heterogeneous distributed object applications."
- [Your-92] Yourdon. The Decline and the Fall of the American Programmer. Englewood Cliffs, NJ: Prentice Hall, 92.
- [WW-90] R. Wirfs-Brock, B. Wilkerson, L. Wiener: Designing Object-Oriented Software, Prentice-Hall, Englewood Cliffs, 1990.
- [Zeid-94] Zeid, A. Object-Oriented Methodologies: A Matter of Perception! Technical Report, Carleton University, Ottawa, Canada.

Appendix A

Generated Code Samples

Samples of Java Script generated files (Chapter 6)

JAVASCRIPT CODE GENERATED BY AGENTINA

The following is the code generated by AgentTINA demo for the example discussed in Chapter 6. The code is generated in Java script.

CODE SAMPLE

```
<Object ID="AgentControl" Width=0 Height=0
  ClassID="CLSID:D45FD31B-5C6E-11D1-9EC1-00C04FD7081F"
  CodeBase="#VERSION=2,0,0,0">
</Object>

<Object ID="L&HTruVoice" Width=0 Height=0
  ClassID="CLSID:B8F2846E-CE36-11D0-AC83-00C04FD97575"
  CodeBase="#VERSION=6,0,0,0">
</Object>

<Script Language="JavaScript">
<!--//
/*
'This template was generated by AgentTINA demo
'Author: Amir Zeid
*/

// * Character Objects
var Genie;

// * Variables
var UsedChars;
var GenieID;
var GenieACS;
```

```
var GenieLoaded;
var LoadReq;
var HideReq;

// * Initialize
UsedChars = "Genie";

// * Genie
GenieID = "Genie";
GenieACS = "Genie.acs";
GenieLoaded = false;

Window_OnLoad();

function Window_OnLoad() {
    // Purpose: Runs automatically when page is loaded

    // * INSERT ANY NON-AGENT RELATED SCRIPTING HERE

    AgentControl.Connected = true;

    GenieLoaded = LoadLocalAgent(GenieID, GenieACS);

    if (GenieLoaded) {
        SetCharObj();
    }
    CheckLoadStatus();
}

function LoadLocalAgent(CharID, CharACS) {
    // Purpose: Attempts to load the specified character
    // Returns: true if successful, false if not

    AgentControl.RaiseRequestErrors = false;
    LoadReq = AgentControl.Characters.Load(CharID, CharACS);
    AgentControl.RaiseRequestErrors = true;

    if (LoadReq.Status != 1) {
        return(true);
    }
    return(false);
}

function SetCharObj() {
    // Purpose: Sets the character reference and TTS Language
    ID

    Genie = AgentControl.Characters.Character(GenieID);
    Genie.LanguageID = 0x409;
}
```



```
function CheckLoadStatus() {
    // Purpose: Determines if required characters have been
    loaded.
    //           If not, issue request to load next character
    //           else run the AgentIntro routine

    if (!GenieLoaded) {
        LoadError();
        return(false);
    }

    window.status = "";
    AgentIntro();
    return(true);
}

function LoadError() {
    var strMsg;
    window.status = "";
    strMsg = "Error Loading Character: " + GenieID + "\n";
    strMsg = strMsg + "This Microsoft Agent Script requires
the character(s):\n";
    strMsg = strMsg + UsedChars;
    alert(strMsg);
}

function InitAgentCommands() {
    // Purpose: Initialize the Commands menu

    Genie.Commands.RemoveAll();
    Genie.Commands.Caption = "My Menu Name";
    Genie.Commands.Add("ACO", "Advanced Character Options",
"Advanced Character Options");
}

function AgentIntro() {
    InitAgentCommands();

    // *** BEGIN USER SCRIPT ***
    Genie.Show();
    Genie.MoveTo (55,100);
    Genie.Play ("Congratulate");
    Genie.MoveTo (155,200);
    Genie.Play ("Blink");
    Genie.Speak("Hello World!");

    // *** END USER SCRIPT ***
}
//-->
</Script>
```

```
<Script Language="JavaScript" For="AgentControl"
Event="RequestComplete(RequestObject)">
<!--//
    // Purpose: Take action on completion or failure of
requests

//-->
</Script>

<Script Language="JavaScript" For="AgentControl"
Event="Command(UserInput)">
<!--//
    // Purpose: Determine Command that was selected either by
menu or voice
    //           and run the applicable Command Script
    var BadConfidence;
    BadConfidence = 10;

    if (UserInput.Confidence <= -40)
    {
        // Bad Recognition
    }
    else if ((UserInput.Alt1Name != "") &&
(Math.abs(Math.abs(UserInput.Alt1Confidence) -
Math.abs(UserInput.Confidence)) < BadConfidence))
    {
        // Bad Confidence - too close to another command
    }
    else if ((UserInput.Alt2Name != "") &&
(Math.abs(Math.abs(UserInput.Alt2Confidence) -
Math.abs(UserInput.Confidence)) < BadConfidence))
    {
        // Bad Confidence - too close to another command
    }
    else
    {
        // High Confidence
        // *** BEGIN USER COMMANDS ***
        switch(UserInput.Name) {
            case "ACO" :
                AgentControl.PropertySheet.Visible = true;
                break;
        }
        // *** END USER COMMANDS ***
    }
//-->
</Script>
```

[illegible]

Appendix B

Petri-Net Analysis

Analysis for Chapter 7

APPENDIX B

PETRI-NET ANALYSIS FOR CHAPTER 7

In this appendix we present the petri-net analysis for the examples presented in chapter 7.

1- DISPATCHING A MOBILE AGENT

-----Petri Net General Analysis-----

Results generated on: Thu Mar 16 13:17:10 EST 2000

1. This is a safe Petri net(1-bounded net)
2. Possible different markings: 64
3. This Petri net structure contains:
 - 6 places,
 - 6 transitions,
 - 8 input/inhibitor arcs,
 - 7 output arcs,
 - 2 tokens.

-----Petri Net Marking Analysis-----

1. This is a marked Petri Net
2. This is not a marked graph
3. The following places have reached their capacities:
 - 1, 5,

4. Properties of places

Place ID	Marking	Capacity	Text
1	1	1	
2	0	1	
3	0	1	
4	0	1	
5	1	1	
6	0	1	

-----Petri Net Inefficiency Checking-----

1. There are no places with more tokens than their capacity allows.

2. The following places have no input or output arcs:
--

3. The following transitions have no input or output arcs:
--

-----Petri Net Incidence Matrix-----

	P1	P2	P3	P4	P5	P6
T1	-1	0	1	0	0	0
T2	-1	0	0	1	0	0
T3	1	-1	0	0	0	0
T4	0	0	-1	0	-1	1
T5	0	0	0	-1	-1	1
T6	0	1	0	0	1	-1
T7	0	0	0	0	0	0

2- NEGOTIATION BETWEEN PCA AND SPA

-----Petri Net General Analysis-----

Results generated on: Mon Mar 20 11:45:05 EST 2000

1. This is a safe Petri net(1-bounded net)

2. Possible different markings: 4096

3. This Petri net structure contains:
 12 places,
 11 transitions,
 17 input/inhibitor arcs,
 15 output arcs,
 2 tokens.

-----Petri Net Marking Analysis-----

1. This is a marked Petri Net
2. This is not a marked graph
3. The following places have reached their capacities:
-- 1, 9,

4. Properties of places

Place ID	Marking	Capacity	Text
1	1	1	
2	0	1	
3	0	1	
4	0	1	
5	0	1	
6	0	1	
7	0	1	
8	0	1	
9	1	1	
10	0	1	
11	0	1	
12	0	1	

-----Petri Net Inefficiency Checking-----

1. There are no places with more tokens than their capacity allows.
2. The following places have no input or output arcs:
--
3. The following transitions have no input or output arcs:
--

-----Petri Net Incidence Matrix-----

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
T1	-1	1	1	0	0	0	0	0	0	0	0	0
T2	0	0	-1	1	0	0	0	0	-1	0	0	0
T3	0	0	-1	0	1	0	0	0	-1	0	0	0
T4	0	-1	0	-1	0	0	0	0	0	0	0	1
T5	0	-1	0	0	-1	0	0	1	0	1	0	0
T6	0	0	0	0	0	1	0	0	0	-1	0	0
T7	0	0	0	0	0	0	1	0	0	-1	0	0
T8	0	0	0	0	0	-1	0	-1	0	0	0	1
T9	0	0	0	0	0	0	-1	-1	0	0	1	0
T10	1	0	0	0	0	0	0	0	1	0	0	-1
T11	1	0	0	0	0	0	0	0	1	0	-1	0

3- ESTABLISHING A CONNECTION FOR A MOBILE AGENT

-----Petri Net General Analysis-----

Results generated on: Mon Mar 20 21:06:57 EST 2000

1. This is a safe Petri net (1-bounded net)
2. Possible different markings: 512
3. This Petri net structure contains:
 - 9 places,
 - 8 transitions,
 - 12 input/inhibitor arcs,
 - 12 output arcs,
 - 2 tokens.

-----Petri Net Marking Analysis-----

1. This is a marked Petri Net
2. This is not a marked graph
3. The following places have reached their capacities:
 - 6, 9,

4. Properties of places

Place	ID	Marking	Capacity	Text
1		0	1	
2		0	1	
3		0	1	
4		0	1	
5		0	1	
6		1	1	
7		0	1	
8		0	1	
9		1	1	

-----Petri Net Inefficiency Checking-----

1. There are no places with more tokens than their capacity allows.
2. The following places have no input or output arcs:
 -
3. The following transitions have no input or output arcs:
 -

-----Petri Net Incidence Matrix-----

	P1	P2	P3	P4	P5	P6	P7	P8	P9
T1	-1	1	0	0	0	0	0	0	0
T2	1	0	1	0	-1	0	0	-1	0
T3	1	0	0	1	-1	0	-1	0	0
T4	0	-1	0	-1	0	1	1	0	0
T5	0	-1	-1	0	0	1	0	1	0
T6	0	0	0	0	1	-1	0	0	0
T7	0	0	0	0	0	0	1	0	-1
T8	0	0	0	0	0	0	0	1	-1

4- MONITORING EMAIL

-----Petri Net General Analysis-----

Results generated on: Tue Mar 21 14:36:03 EST 2000

1. This is a safe Petri net (1-bounded net)
2. Possible different markings: 128
3. This Petri net structure contains:
 - 7 places,
 - 6 transitions,
 - 9 input/inhibitor arcs,
 - 9 output arcs,
 - 3 tokens.

-----Petri Net Marking Analysis-----

1. This is a marked Petri Net
2. This is not a marked graph
3. The following places have reached their capacities:
 - 1, 2, 6,

4. Properties of places

Place ID	Marking	Capacity	Text
1	1	1	
2	1	1	
3	0	1	
4	0	1	
5	0	1	
6	1	1	
7	0	1	

-----Petri Net Inefficiency Checking-----

1. There are no places with more tokens than their capacity allows.

2. The following places have no input or output arcs:
--

3. The following transitions have no input or output arcs:
--

-----Petri Net Incidence Matrix-----

	P1	P2	P3	P4	P5	P6	P7
T1	-1	0	0	0	1	0	0
T2	-1	0	1	0	0	0	0
T3	1	1	-1	-1	0	0	0
T4	0	-1	0	1	0	0	0
T5	0	0	0	-1	-1	0	1
T6	1	1	0	0	0	0	-1

Abbreviations

<i>Term</i>	<i>Meaning</i>
ACL	Agent Communication Language
AOP	Agent Oriented Programming
AOT	Agent Oriented Technology
ATM	Asynchronous Transfer Mode
B-ISDN	Broadband Integrated Services Digital Network
BCP	Basic Call Processing
BDI	Belief Desire Intention
BIN	Broadband Intelligent Networks
DPE	Distributed Processing Environment
FE	Functional Entities
FIPA	Foundation for Intelligent Physical Agent
Genie	Microsoft Agent
GSL	Global Service Logic
GSM	Global System For Mobile Telecommunication
IA	Intelligent Agent
IN	Intelligent Networks
INCM	Intelligent Network Conceptual Model
ITU-T	International Telecommunications Union
MAF	Mobile Agent Facility
NCCE	Native Computing and Communication Environment
NMAS	Natural Multi-Agent Systems
NPA	Network Provider Agent
OAA	Open Agent Architecture
ODP	Open Distributed Processing
OMA	Object Management Architecture
OMG	Object Management Group
OMT	Object Modeling Technique
OOP	Object Oriented Programming

<i>Term</i>	<i>Meaning</i>
ORB	Object Request Broker
PCA	Personal Communication Agent
PCAF	PCA Factory
PN	Petri-Net
POI	Point Of Initiation
POR	Point Of Return
QoS	Quality Of Service
RDD	Responsibility Driven Design
RP	Remote Programming
SCP	Service Control Point
SDA	Service Detection Agent
SF	Service Feature
SIB	Service Independent Building
SOMA	Semantic Object Modeling Approach
SPA	Service Provider Agent
SQL	Structured Query Language
SUA	Service Upgrading Agent
TINA	Telecommunications Information Networking Architecture
TMN	Telecommunications Management Network
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
VOD	Video On Demand
W3C	Worldwide Web Consortium