



**Using Intelligent Agents concepts in Analysis and  
Design of Library Management System**

**By:**

***Naiema Musu Amrayid***

**Supervisor: *Dr. Abdulhamed Mohamed Abdulkafi***

**This thesis is submitted to the Department of Computer Science in partial  
fulfillment of the requirements for the degree of  
Master in Computer Science**

***Al-Tahaddi University, Faculty of Science  
Department of Computer Science  
Sirte, G.S.P.L.A.J.***

**Academic year 2006/2007**

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

(( سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا إِنَّكَ أَنْتَ  
الْعَلِيمُ الْحَكِيمُ ))

آية ( 32 ) من سورة البقرة



التاريخ : .....  
الوقت : 25 .....  
الرقم الاشاري : 1/449/1/1

**Faculty of science**  
**Department of computer science**

*Title of Thesis*  
**Using Intelligent Agents Concepts in Analysis and  
Design of Library Management System**

By  
**Naiema Musa Amrayid**

Approved by:

Dr. Abdulhamed Mohamed Abdulkafi .....  
(Supervisor)

Dr. Mohamed Aboulgasem Arteimi .....  
(External examiner)

Dr. Ali Ahmed Abdulaziz .....  
(Internal examiner)

Countersigned by:  
**Dr. Ahmed Farag Mhgoup**  
(Dean of Faculty of Science)



## **Acknowledgements**

*I would like to thank my supervisor Dr.Abdulhamed.M.bdukkafi for their cooperation with me and for all their support and advice and encouragement towards the research.*

*My special thanks are extended to Dr.Hassan M.Amrieiz whose information, and correcting the language of this thesis, and criticisms really enriched my work. I do sincerely thank Dr.Zakaria.S.Zubi for his help.*

*My grateful thanks go to my husband, for his love, thoughtfulness, support, and sacrifice in so many ways in achieving this. To my mother, Father for their love, support in all ways over the years. To my brothers, specially khaled, and my sisters for their love and support, without them, the completion of this thesis would not to be possible, special thanks to my friends who shared this experience with me.*

*Above all, I thank God for making all this work possible.*

*Naema*

## **Abstract**

"Agent" and "Agent software" have become popular words in computer software. The reason why this area gains the popularity is because it is based upon AI (Artificial Intelligence) but works only in its specific field, like a narrowly focused AI program. Agent software has been developed for many different uses in a variety of areas because of its extraordinary ability of adapting to the specific field of interest" [10].

This thesis first presents an overview of the software agent, which starts from the introduction of the definition of the software agent, its terminology, and its other basics. Afterwards, the concepts and specifications of 'Intelligent' agent are brought into context to describe how the software agent works behind the scene. This includes action, operation, autonomous behavior and communication of agents. The way that a software agent generates its goals of achievement and to evaluation of its progress is also very important in keeping agent working in the right path. Next, the thesis presents a methodology for agent-oriented analysis and design and the key aspects of agent-based software development, focusing on one of the most known methodologies as the *Prometheus* methodology for developing intelligent agent systems. And then we discuss of the phases of analysis and design of agent system according to a selected methodology.

In this thesis one selected function of the library system is programmed. This function is related to the library email agent. This is because Electronic mail has become one of the main communication tools between people around the world regardless of their physical distance. The library email agent project is a very typical software agent application. The source code of the Intelligent Library Email Agent (ILEA) program is included in the Appendix A.

## Table of contents

<i>Acknowledgements</i>	ii
<i>Abstract</i>	iii
<i>List of Figures</i>	viii
<i>List of Tables</i>	ix
<b>Chapter 1: Introduction</b>	<b>1</b>
1.1 <i>Review of literature</i>	3
1.2 <i>Objective's</i>	5
1.3 <i>Thesis organization</i>	5
<b>Chapter 2: Fundamental Concepts for Software Agent</b>	<b>7</b>
2.1 <i>Introduction for Software Agent</i>	7
2.2 <i>Definition of Software Agent</i>	8
2.3 <i>A Typology of Agents</i>	9
2.4 <i>Agent Variants</i>	13
2.4.1 <i>Mobile Agent</i>	13
2.4.2 <i>Distributed Agent</i>	14
2.4.3 <i>Multiple Agents</i>	14
2.4.4 <i>Collaborative Agents</i>	14
2.4.5 <i>Social Agents</i>	15
2.5 <i>Why are Agents Needed</i>	16
2.6 <i>Agent Architectures</i>	17
2.6.1 <i>Deliberative Model</i>	17
2.6.2 <i>Reactive Model</i>	19
2.6.3 <i>Hybrid Model</i>	20
2.7 <i>Benefits of Agents</i>	20
2.8 <i>Security Issues in Agent System</i>	22
2.9 <i>Multi- Agent Systems</i>	24
<b>Chapter 3: Intelligent Agent</b>	<b>25</b>
3.1 <i>Definition of Intelligent Software Agent</i>	25
3.2 <i>Intelligent Agents and Artificial Intelligence</i>	26
3.3 <i>Intelligent Agent Model</i>	26
3.3.1 <i>Task Level Skills</i>	27

3.3.2 Knowledge Module	28
3.3.3 Communication Skill	28
3.4 Taxonomy of Intelligent Agents	29
3.5 Concepts for Intelligent Agents	30
3.6 Structure of Intelligent Agents	39
3.7 Applications of Intelligent Agents	42
3.8 Intelligent Information Management	47
3.9 Email Filtering	49
<b>Chapter 4: Prometheus as an Agent-Oriented Analysis and</b>	<b>86</b>
<b>Design Methodology</b>	
4.1 Introduction	50
4.2 Differences between AO and OO methodologies	52
4.3 Prometheus	55
4.3.1 Agent Concepts in Prometheus Methodology	57
4.3.2 Overview of Prometheus Methodology	59
4.3.2.1 System Specification	60
4.3.2.2 Architectural Design	63
4.3.2.3 Detailed Design	67
4.3.3 Tool Support	70
<b>Chapter 5: Design and Implementations of the Case Study</b>	<b>73</b>
5.1 System Specification	73
5.1.1 Goals of the System	74
5.1.2 Sub-goals of the System	74
5.1.3 Scenarios of the System	75
5.2 Architectural Design	79
5.3 Detailed Design	81
5.3.1 Checkout Agent	82
5.3.2 Reservation Agent	83
5.3.3 Overdue Agent	84
5.4.4 Library Email Agent	84
5.4.4.1 Features of (LEA)	84
5.4.4.2 Tools Required	88
5.4.4.3 Graphical user Interface	89

5.4.4.4 <i>Storing the Rules In a Control File</i>	93
5.4.4.5 <i>Prototype Algorithm</i>	95
<b>Chapter 6: Conclusion</b>	99
6.1 <i>Summary</i>	99
6.2 <i>Recommendation and Future Work</i>	101
<b>References</b>	102
<b>Appendix A: Source code Discussions</b>	108



## List of Figures

<i>Figure</i>		<i>Page</i>
2-1	Agent interacts with environments through sensors and effectors	7
2-2	Classification of Agents	9
2-3	A part view of an Agent Typology	11
2-4	Agent taxonomy	16
2-5	Simple Deliberative Agent Model	18
2-6	BDI Agent Model	18
2-7	Reactive Agent Model	20
3-1	Intelligent Agent Model	27
3-2	Agents are situated	31
3-3	Proactive Agents have Goals, Reactive Agent have Events	33
3-4	Adding Plans and Beliefs	35
3-5	Agent Execution Cycle	37
4-1	Genealogy of Agent-Oriented Methodologies	54
4-2	Agent-Oriented Methodologies	54
4-3	The phases of the Prometheus methodology	60
4-4	Notation used in System Overview Diagram	67
4-5	Notation used in Agent Overview Diagrams	67
4-6	The Prometheus Design Tool (PDT)	72
5-1	Goal Overview Diagram	76
5-2	Scenarios Diagram	76
5-3	System Roles Diagram	79
5-4	Data Coupling Diagram	80
5-5	Agent Role Coupling Diagram	80
5-6	Agent Acquaintance Diagram	81
5-7	System Overview Diagram	81
5-8	Checkout Capability	82
5-9	Return Capability	82
5-10	1.Get Return Date Capability	82
5-11	2.Get Return Date Capability	83

5-12	Reservation Capability	83
5-13	Arrival Notification Capability	83
5-14	Overdue Agent	84
5-15	Library Email Agent	84
5-16	Features of intelligent agent	86
5-17	Block diagram of the intelligent agent for email information processing	87
5-18	Data extraction from sorting process	87
5-19	Library E-mail Agent Main Form	89
5-20	Create Rule Form	90
5-21	The Setup Form	91
5-22	Prototype Algorithm	96
5-23	sorting algorithm with identification of forward and replied	97
5-24	sorting algorithm with sensitivity of intelligent	98
<b>Appendix A</b>		<b>108</b>
A.1	Coding the Support Routines	108
A.1.1	The Initialization Routines	108
A.1.2	The List-Handling Routines	109
A.1.3	The Message Processing Routines	109
A.2	Building (ILEA) Actions, Tests, and Rules	120

## **List of Tables**

<i>Table</i>		<i>Page</i>
<i>2-1</i>	Features, Advantages and Benefits of Agent Technology	<i>21</i>
<i>3-1</i>	Agent Concept	<i>39</i>
<i>3-2</i>	The basic elements for a selection of agent types	<i>41</i>
<i>4-1</i>	The Major Models of Prometheus	<i>58</i>
<i>5-1</i>	Tools required	<i>88</i>
<i>A-1</i>	(ILEA) tests and actions	<i>120</i>
<i>A-2</i>	Adding rules to the (ILEA)	<i>121</i>

## Chapter One

### Introduction

The past decade has experienced a great deal of scientific activity on standard intelligent techniques (fuzzy logic, evolutionary computing, machine learning & neural networks) as well as on adaptive and hybrid intelligent systems. Despite the continuously growing number of publications in this area, it is not clear yet whether companies have realized the potential of intelligent techniques in solving real- world problems and the new opportunities given on a business level. Research and development efforts in computer science have increasingly embraced the concept of software agents and multi-agent systems. One key reason is that the idea of an agent as an autonomous system, capable of interacting with other agents in order to satisfy its design objectives, is a naturally appealing one for software designers. This has led to the growth of interest in agents as a new design-paradigm for software engineering [12].

Intelligent agents are a powerful Artificial Intelligence technology, which shows considerable promise as a new paradigm for mainstream software development. In recent years, agent-based systems have received considerable attention in both academic and industry. An intelligent agent is one, which is able to make rational decisions, i.e., blending proactiveness and reactiveness, showing rational commitment to decisions made, and exhibiting flexibility in the face of an uncertain and changing environment. Despite their promise, intelligent agents are still scarce in the market place. There is a real technical reason for this, which stems from the fact that developing intelligent agent software currently requires significant training and skills.

The reasons why developing intelligent agent systems are difficult include: [11]

1. Immature tool support: There is a lack of good debugging tools. many tools are research prototypes and lack efficiency, portability, documentation, and/or support.
2. The need for processes and methodologies: Programmers are familiar with designing object-oriented systems. However, the design of agent-oriented systems differs in a number of ways e.g. identifying rules, goals, and interaction patterns.
3. Design guidelines and examples: Designing a collection of plans to achieve a goal is different to designing a single procedure to perform a function. This difference is fundamental, the fact that developing intelligent agent, is a different programming paradigm and needs to be learnt and taught as such.
4. Complex concepts such as intentions are difficult to explain and this is not helped by a lack of agreement on concepts and inconsistent terminology.
5. Lack of a suitable textbook: much of the work on intelligent agents is scattered across many research papers (sometimes collected into volumes).

The development of agent-based software and multi-agent systems is increasing significantly. Thus, agent-based software and multi-agent system designers need methodologies and tools to help them. The main purpose of Agent-Oriented Software Engineering (AOSE) is to create methodologies and tools that enable inexpensive development and maintenance of agent-based software. In addition, the software should be flexible, easy-to-use, scalable, and of high quality. In other words, quite similar to the research issues of other branches of software engineering, for example, object-oriented software engineering. Agent-oriented programming (AOP) can be seen as an extension of object-oriented programming (OOP), on the other hand, can be seen as a successor of

structured programming. Recent years have seen an incredible development of agent-oriented software engineering methodologies trying to cover all the range of features that agents and agent-based systems encompass. These methodologies are based either on object-oriented methodologies or on specific to agent theory [16].

## 1.1 Review of literature

There is a lot of interesting work showing a methodological approach for agent system development. These approaches must be analyzed to identify possible enhancements. One example can be seen in [35], which shows a feature-based evaluation of several AOSE methodologies. Their criteria include software engineering related criteria and criteria relating to agent concepts. Another paper [34] used the same techniques in addition to a small experimental evaluation to perform an evaluation of their own Agent Oriented Modelling Techniques (AOMT), this work suffers from subjectivity in that the criteria they identified are those that they see as important, and naturally, AOMT focuses on addressing these criteria. A framework to carry out an evaluation of agent-oriented analysis and design modeling methods has been proposed by other researchers [18]. The proposal makes use of feature based evaluation techniques but metric and quantitative evaluations are also introduced. The significance of the framework is the construction of an attribute tree, where each node of the tree represents a software engineering criterion or a characteristic of agent-based system. Each attribute is assigned with a score and the score of attributes on the node is calculated based on those of their children. They have applied that framework to evaluate and compare two

AOSE methodologies: the Agent Modeling Techniques for Systems of BDI (Belief, Desire and Intention) Agents and MAS-CommonKADS.

Authors in [2] propose a number of criteria for evaluating methodologies with a view to allowing organizations to decide whether to adopt AOSE methodologies or use existing Object Oriented methodologies. Although they performed a survey to validate their criteria, they do not provide detailed guidelines or methods for assessing methodologies against their criteria. Their example comparison (between MaSE and Booch) gives ratings against the criteria without justifying them. Their work is useful in that it provides a systematic method of taking a set of criteria, weightings for these criteria (determined on a case by case basis), and an assessment of a number of methodologies and determining an overall ranking and an indication of which criteria are critical to the result.

A large number of agent-oriented methodologies have been proposed in recent years [15]. Compared with other methodologies, Prometheus' distinguishing features are that it is aimed at industrial practitioners (as well as students), that it aims to be detailed and complete, that it emphasizes the importance of tool support and automated consistency checking, and that it supports the detailed design of plan-based agents. The structured nature of the design artifacts allow for development of support structures for debugging as demonstrated in this thesis. This work suffers from subjectivity in that the criteria they identified are those that they see as important and, naturally, AOMT focuses on addressing these criteria. Comparisons of Prometheus with other agent-oriented methodologies can be found in [4, 14, 15, 17, and 18].

Another experiment shown in [6] analyzed four agent-oriented methodologies to evaluate them in common criteria. The experiment comprehended two trials in which two agent-oriented systems were developed using each methodology. They presented some contradictions between the two trials that are credited to the different level of experience in agent-oriented development among the participants of each trial.

Other researchers [38] presented five agent oriented software engineering methodologies and discussed their conceptual foundations like interaction, cooperation, collaboration, organizational design etc wherever applicable.

## **1.2 Objectives**

The main objectives of this research are:

1. Illustrate the main reasons why people need software agents.
2. Discussion of the phases of analysis and design of agent system according to a selected methodology.
3. Applying the previous phases to a library system.
4. Coding and programming a part of this library system to act as intelligent agent within the library system.

## **1.3 Thesis Organization**

We intend to organize the research into different Chapters. Chapter two illustrates Fundamental concepts for software agent. While Chapter three, presents a theoretical background for intelligent agents. In Chapter four, the research discusses one of the well-known methodology procedures. Chapter five presents a case study and applies the



methodology procedures to the selected case study. The conclusions of the research are presented in Chapter six. One appendix is used to include the source code of the Intelligent Library Email Agent (ILEA) program.

## Chapter Two

### Fundamental concepts for Software Agent

#### 2.1 Introduction for Software Agent

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors. A software agent has encoded bit strings as its percepts and actions. A generic agent is shown in Figure (2-1) [39].

In this section we will discuss some of the general principles used in the design of agents throughout the thesis, among which is the principle that agents should know things. In the end, we will show how we might go about building one.

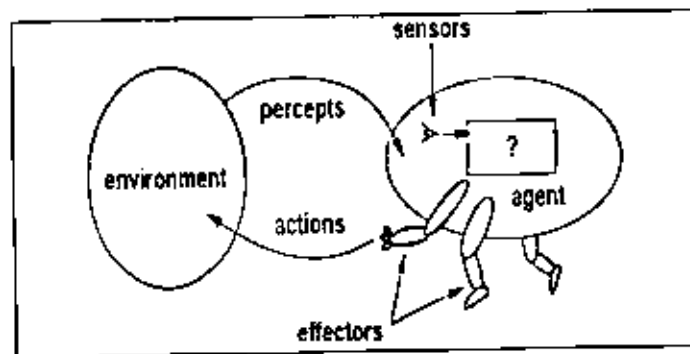


Figure (2-1) - Agent interacts with environments through sensors and effectors.<sup>1</sup>

<sup>1</sup> S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice-Hall, 1995.

## 2.2 Definition of Software Agent:

There are at least two reasons why it is so difficult to define precisely what agents are. **Firstly**, agent researchers do not own this term in the same way as fuzzy logicians/AI researchers, for example, the term fuzzy logic - it is one that is used widely in everyday parlance as in travel agents, estate agents, etc. **Secondly**, even within the software fraternity, the word agent is really an umbrella term for a heterogeneous body of research and development. The response of some agent researchers to this lack of definition has been to invent yet some more synonyms, and it is arguable if these solve anything or just further add to the confusion.

An agent is very generally defined as "a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives". Various definitions have been proposed. For instance, in restricted situations of Game Theory one can adopt this definition : "An agent is an entity which can receive information about a state of the environment, take actions which may alter that state, and express preferences among the various possible states[13].

Instead of the *formal* definition, a list of general characteristics of agents will be given. Together these characteristics give a global impression of what an agent is[28].

## 2.3 A Typology of Agents

This section attempts to place *existing* agents into different agent classes, i.e. its goal is to investigate a typology of agents.

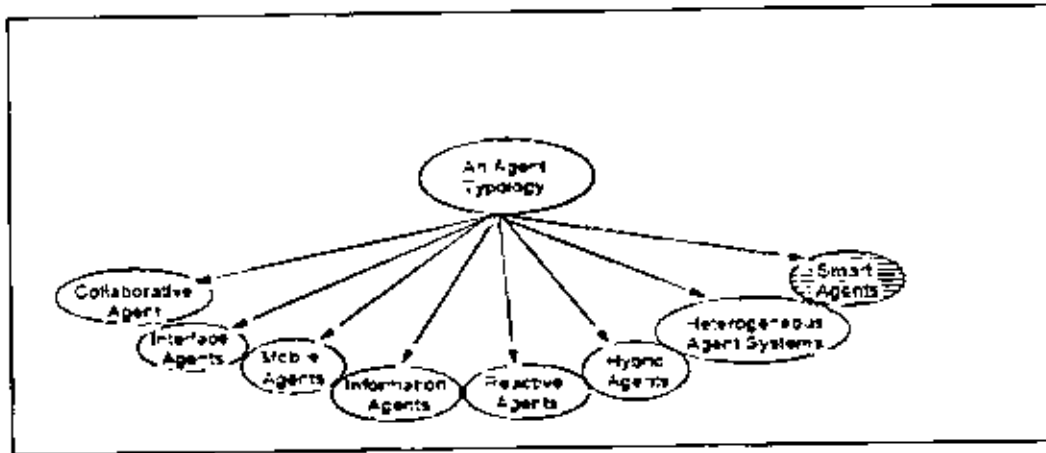


Figure (2-2) - Classification of Agents from <sup>1</sup>

A typology refers to the study of types of entities. There are several dimensions to classify existing software agents.

Firstly, agents may be classified by their mobility, i.e. by their ability to move around some network. This yields the classes of *static* or *mobile* agents.

Secondly, they may be classed as either *deliberative* or *reactive*. Deliberative agents derive from the deliberative thinking paradigm: the agents possess an internal symbolic reasoning model and they engage in planning and negotiation in order to achieve coordination with other agents. These agents on the contrary do not have any internal, symbolic models of their environment, and they act using a stimulus/response type of behavior by responding to the present state of the environment in which they are embedded.

<sup>1</sup> G. Pang, "Implementation of an Agent-Based Business Process", Research paper, Chengdu, Sichuan, China, University Zürich, 2000.

Thirdly, agents may be classified along several ideal and primary attributes, which agents *should* exhibit. At BT Labs in [11] identified a minimal list of three: autonomy, learning and cooperation. The research appreciate that any such list is contentious, but it is no more or no less so than any other proposal. *Autonomy* refers to the principle that agents can operate on their own without the need for human guidance, even though this would sometimes be invaluable. Hence, agents have individual internal states and goals, and they act in such a manner as to meet its goals on behalf of its user.

A key element of their autonomy is their proactiveness, i.e. their ability to take the initiative rather than acting simply in response to their environment. *Cooperation* with other agents is paramount: it is the *reason for being* for having multiple agents in the first place in contrast to having just one. In order to cooperate, agents need to possess a social ability, i.e. the ability to interact with other agents and possibly humans via some communication language. Having said this, it is possible for agents to coordinate their actions without cooperation.

Lastly, for agent systems to be truly smart, they would have to *learn* as they react and/or interact with their external environment. In our view, agents are (or should be) disembodied bits of intelligence. However, we will not attempt to define what intelligence is, we maintain that a key attribute of any intelligent being is its ability to learn.

The learning may also take the form of increased performance over time. We use the three minimal characteristics in Figure (2-3) to derive four types of agents to be included in our typology: *collaborative agents*, *collaborative learning agents*, *interface agents* and truly *smart agents*.

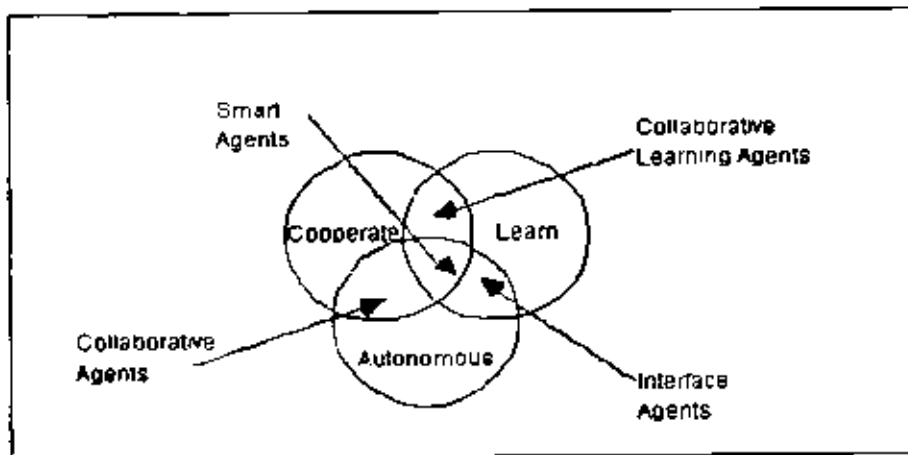


Figure (2-3)- A part view of an Agent Typology<sup>1</sup>

We emphasize that these distinctions are *not* definitive. For example, with collaborative agents, there is more emphasis on cooperation and autonomy than on learning; hence, we do not imply that collaborative agents never learn. Likewise, for interface agents, there is more emphasis on autonomy and learning than on cooperation. We do *not* consider anything else which lie outside the intersecting areas to be agents. For example, most expert systems are largely autonomous but, typically, they do not cooperate or learn. Ideally, in our view, agents should do all three equally well, but this is the *aspiration* rather than the reality.

In principle, by combining the two constructs so far (i.e. static/mobile and reactive/deliberative) in conjunction with the agent types identified (i.e. collaborative

---

<sup>1</sup> H. Xu and S. M. Shatz. "A Framework for Model-Based Design of Agent-Oriented Software", Research paper, The University of Illinois, Chicago, 1999.

Available at: [www.cis.umassd.edu/~hXu/Papers/UIC/TSE.pdf](http://www.cis.umassd.edu/~hXu/Papers/UIC/TSE.pdf) . 2006-7-20.

agents, interface agents, etc.), we could have *static deliberative collaborative agents*, *mobile reactive collaborative agents*, *static deliberative interface agents*, *mobile reactive interface agents*, etc. However, these categories, though quite a mouthful, may also be necessary to further classify existing agents.

Fourthly, agents may sometimes be classified by their rules (preferably, if the rules are major ones), e.g. World Wide Web (WWW) information agents. This category of agents usually exploits internet search engines such as WebCrawlers, Lycos and Spiders. Essentially, they help manage the vast amount of information in wide area networks like the internet. We refer to these classes of agents in this research as *information or internet agents*. Again, information agents may be static, mobile or deliberative. Clearly, it is also pointless making classes of other minor rules as in report agents, presentation agents, analysis and design agents, testing agents, packaging agents and help agents - or else, the list of classes will be large.

Fifthly, we have also included the category of *hybrid* agents, which is combined of two or more agent philosophies in a single agent. There are other attributes of agents, which we consider *secondary* to those already mentioned. For example, is an agent versatile (i.e. does it have many goals or does it engage in a variety of tasks)? Does an agent lie knowingly or is it always truthful (this attribute is termed veracity)? Can you trust the agent enough to (risk) delegate tasks to it? Is it temporally continuous? Does it degrade gracefully in contrast to failing drastically at the boundaries? Perhaps unbelievably, some researchers are also attributing emotional attitudes to agents - are they fed up being asked to do the same thing repeatedly? What role does emotion have in constructing believable agents? Some agents are also imbued with *mentalistic* attitudes or

notions such as beliefs, desires and intentions - referred to typically as BDI agents. Such attributes as these provide for a stronger definition of agenthood.

In essence, *agents exist in a truly multi-dimensional space*, which is why we have not used a two or three-dimensional matrix to classify them - this would be incomplete and inaccurate. However, for the sake of clarity of understanding, we have collapsed this multi-dimensional space into a single list.

In order to carry out such an audacious move, we have made use of our knowledge of the agents we know are currently out there and what we wish to aspire to. Therefore, the ensuing list is to some degree arbitrary, but we believe these types cover most of the agent types being investigated currently. We have left out collaborative learning agents, see Figure (2-3), because we do not know of the existence out there of any such agents, which collaborate and learn, but are not autonomous [12].

## **2. 4 Agent Variants**

There are several variants to agents: [12]

### **2. 4.1 Mobile Agents:**

Also known as traveling agents, these programs will shuttle their being, code and state, among resources. This often improves performance by moving the agents to where the data reside instead of moving the data to where the agents reside.

The alternative typical operation involves a client-server model. In this case, the agent, in the role of the client, requests that the server transmit volumes of data back to the agent to be analyzed. In many situations, the agent must return the data to



the server in a processed form. Significant bandwidth performance improvements can be achieved by running the agents within the same chassis as the data.

Mobile agent frameworks are currently rare due to the high level of trust required to accept a foreign agent onto one's data server. With advances in technologies for accountability and immunity, mobile agent systems are expected to become more very popular.

#### **2.4.2 Distributed Agents:**

Load balancing can be achieved by distributing agents over a finite number of computational resources. Some mobile agents are self-distributing, seeking and moving to agent platforms that can offer the higher computational resources at lower costs.

#### **2.4.3 Multiple Agents:**

Some tasks can be broken into sub-tasks in order to be performed independently by specialized agents. Such agents are unaware of the existence of the others but nonetheless rely upon the successful operations of all.

#### **2.4.4 Collaborative Agents:**

Collaborative agents interact with each other to share information or barter for specialized services to affect a deliberate synergism. While each agent may uniquely speak the protocol of a particular operating environment, they generally share a common

interface language, which enables them to request specialized services from their brethren as required.

#### 2.4.5 Social Agents:

Some researchers see anthropomorphism as a key requirement to successful collaboration between humans and agents. To this end, agents are being developed, which can both present themselves as human-like creations as well as interpret human-generated communications such as continuous speech, gestures, and facial expressions. There are some applications, which combine agents from two or more of these categories, and these are referred to as *heterogeneous agent systems*. Such applications already exist even though they are relatively few. Another issue of note is that agents need not be benevolent to one another. It is quite possible that agents will be in competition with one another, or perhaps quite antagonistic towards each other [12].

In the middle of the nineties when the model first appeared it was relatively clear that there is not a widely shared conceptual framework and this proposal, among others, reflected a clear effort to show the origins of the different meanings and offer some alternatives of complementary conceptual frameworks. For instance the conceptual review of taxonomy of autonomous agents, reproduce it in the Figure (2-4). All this kind of classifications are oriented to understand the scientific production and the related software products. [5].

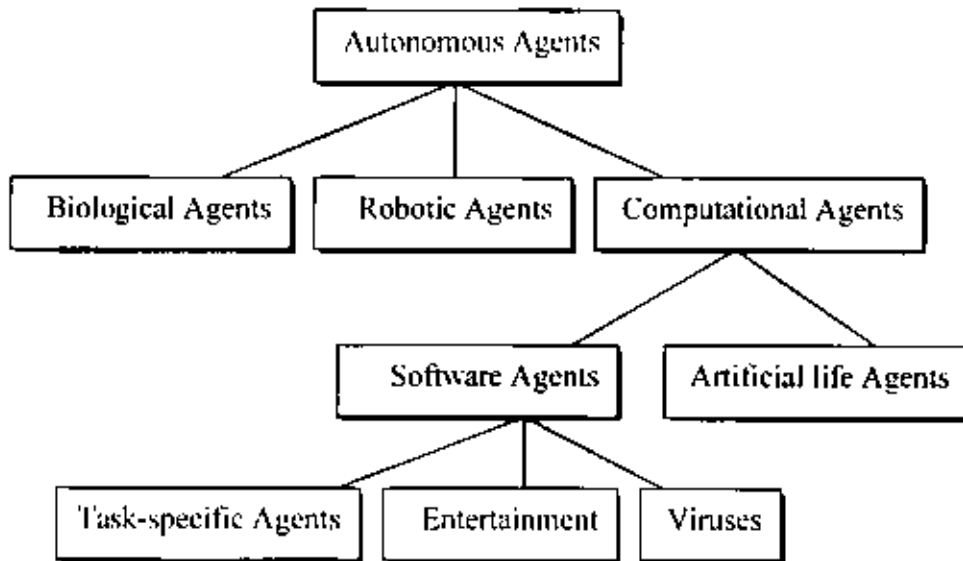


Figure (2-4). Agent taxonomy

## 2.5 Why Are Agents Needed

Today increasing usage of any kind of networking makes the working environment of computer software becoming dynamic and open. Software agents are designed for this kind of environment.

There are four main reasons why people need software agents.

1. A personal requirement.
2. They are changes in the business environment that are driving agent development.
3. There is a huge increasing growth of personal computers.
4. The emergence of the Internet.

It is obvious that both the tasks performed on the computers and the amount of time, that people spend on computers are increasing and will continue to do so. In addition, the tasks performed by the computers are becoming much more complicated as people try to make computers do their jobs.

Software agents are designed for working autonomously without user intervention. The use of software agents can not only provide help and assistance to the users, but can also result in cost effectiveness [10].

## **2.6 Agent Architectures**

The choice of a specific architecture is quite a crucial one in the construction of an agent, and will have influence on characteristics of the agent. There are three basic agent architectures: deliberative, reactive and hybrid models [10].

### **2.6.1 Deliberative Model**

A deliberative agent is one that contains an explicitly represented, symbolic model of the world, and in which decisions are made through logical reasoning based on pattern matching and symbolic manipulation. There are two problems, which arise in the implementation of a deliberative agent.

**Firstly**, the transduction problem: "How can the real world be translated into a symbolic description that the agent can understand?"

**Secondly**, the representation/reasoning problem: "How can a complex world be represented symbolically", and "how can agents use the information to reason about a problem and make the results to be useful?"

Deliberative agents are sometimes called BDI (belief, desire, intention) agents.

This picture gives an overview of the structure of the mental state.

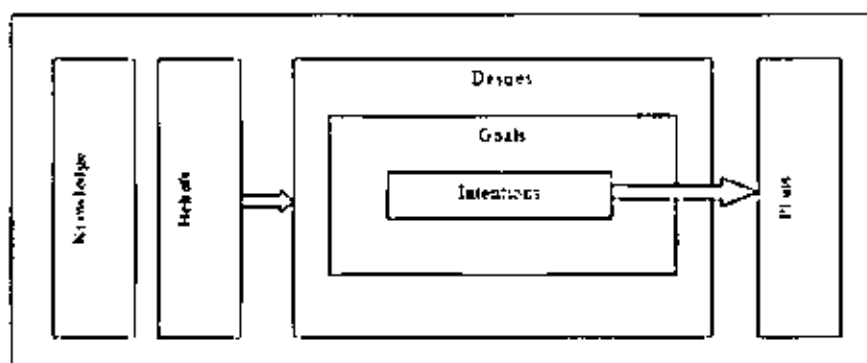


Figure (2-5). Simple Deliberative Agent Model<sup>1</sup>

From Figure (2-5), the desires are what first are formed, based on the agent's beliefs. Desires are the most unstructured of the agent's hope for future situations and are allowed to be conflicting and unrealistic. A subset of the desires is the goals. These are basically desires that do not conflict with each other and that are realistic. Intentions are goals that the agent has decided to follow. The plans consist of the single actions necessary to do so.

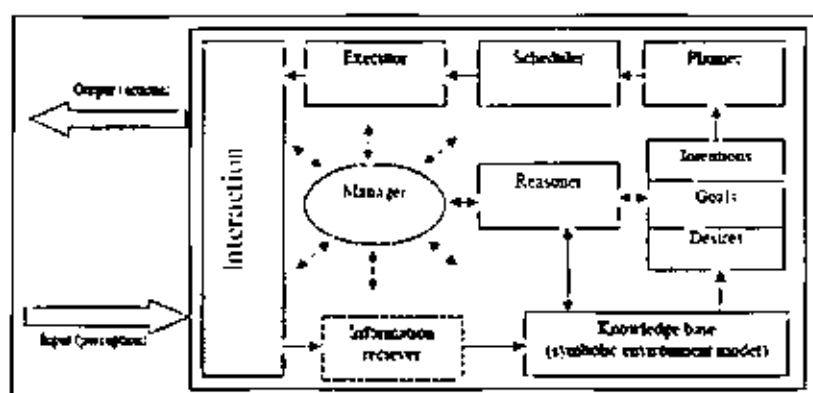


Figure (2-6). BDI Agent Model<sup>1</sup>

<sup>1</sup> Hsi-Kuo Li, "software agent". Thesis, Skokie, Illinois, U.S.A December 2002.

Figure (2-6) shows that the information receiver has a very small role to play in the architecture. This means that the internal model of the agent can only be modified to a limited extent. The rigid structure of the plan-based system and the minimal updating of the internal model make the agent unsuitable for a dynamically changing environment. Another problem is the time necessary for the agent to reach a decision. As stated earlier, a BDI agent uses logic to form its intentions from its beliefs. The BDI model is not constructed with time efficiency in mind. The algorithms that produce the plans for the agents are designed for perfect results, and this makes them quite [10].

### **2.6.2 Reactive Model**

Figure (2-7) shows Reactive Agent Model, a major advantage of the reactive model is its fault tolerance. Since the competence modules work in parallel, it is not fatal if one of them should fail. The agent can probably carry out its task anyway.

Reactive agents do not have an explicit internal model of their environment, as deliberative agents do. They cannot use logic to reason with their knowledge and reach decisions. That a reactive agent cannot solve a problem for which it has no competence module. BDI agents are designed in a more generic way, not being restricted to a specific class of tasks, but are limited by the amount of information in their knowledge base [10].

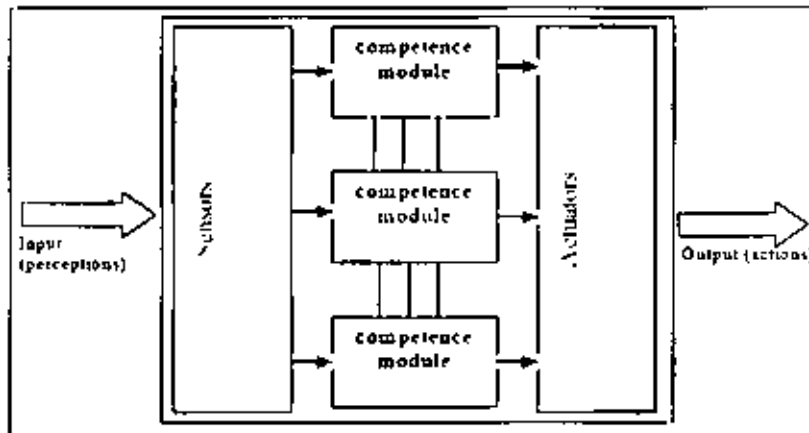


Figure (2-7). Reactive Agent Model<sup>1</sup>

### 2.6.3 Hybrid Model

An agent of the hybrid model has both a reactive and a deliberative part. The deliberative part maintains the knowledge base and does the reasoning, planning and decision-making, and the reactive module is concerned with interaction with the environment [10].

## 2.7 Benefits of Agents

The benefit of an agent is based on the skills that the agent has. Table (2-1) shows Features, Advantages and Benefits of Agent Technology [32].

---

<sup>1</sup> Hsi-Kuo Li. "software agent". Thesis, Skokie, Illinois, U.S.A December 2002.

<b>Feature</b>	<b>Advantage</b>	<b>Benefit</b>
Automation	Perform repetitive tasks	Increased productivity
Customization	Customize information interaction	Reduced overload
Notification	Notify user of events of significance	Reduced workload
Learning	Learn user(s) behavior	Proactive assistance
Tutoring	Coach user in context	Reduced training
Messaging	Perform tasks remotely	Off-line work

Table (2-1). Features, Advantages and Benefits of Agent Technology<sup>1</sup>

The repetitive behavior can be either (time-based) or (event-based), for example clearing up the recycle bin before turning off the computer everyday. Unfortunately, different users have different preference that will repeat very often. Because of this dissimilarity, it is very hard to have a design that will be suitable for every user. Agent-based automation is very helpful because the agent can remember the user preference and perform the repetitive behavior automatically [32].

Agents present their customization benefits by presenting only the information that matches the preference of the user. Therefore, the workload can be reduced by eliminating the action for finding and delivering the unnecessary information. Agents can also provide notification benefits. Agents can notify their users of occurrences of events in which users are interested. This will free users to their personal significance and reduce their workload. An example of this is the Windows update notification. Agents

---

<sup>1</sup> N. R. Jennings, M. Wooldridge, "Applications of Intelligent Agents", Research paper, Queen Mary & Westfield College, University of London. Available at:  
<http://www.cs.umbc.edu/agents/introduction/jennings98.pdf>. 2006-5-14



with learning abilities can learn both user behavior and preference. Once they have learned this, they can work proactively. However, these agents need some time for them to learn all the user behaviors and preferences. Agents with tutoring abilities can coach users to correct their mistakes. This benefit can reduce the training requirement. A messaging agent can allow users to accomplish their task off-line at a remote site. Mobile agents are one kind of messaging agents. These agents can travel around the sites themselves to interact with other agents and perform tasks on behalf of users [32].

## **2.8 Security Issues in Agent System**

An agent is a computer program that represents a human user. This may mean that the program makes decisions, or gives advice, that would otherwise have been made or given only by a human. It may mean that the program travels over a network to carry out operations on remote systems on behalf of a less mobile human user. It may mean both. Agents that travel from system to system are called mobile or itinerant. A sample itinerant agent is a program that is dispatched into a network to find some information for a user. It moves from system to system, finding pieces of the desired information. A sample intelligent agent is a sophisticated mail-sorting program that processes incoming mail for the user, making decisions about what mail to discard unread, what to mark as urgent, what to send a reply to, and so on. An example of an agent that falls into both categories is a shopping agent that users dispatch into the network to find goods for sale and to decide what goods to purchase on behalf of the users. These three samples are three different kinds of agents that people often use. So what is different about security in agent systems? Here is a list of issues of agent security: [32]

- 1- Delegation: users are delegating some of their authorities to their agents in order to allow agents to perform the tasks. However, according to the basic design of agents, users cannot always see what agents are doing when agents are performing the given tasks. Agents can access wherever the users permit them to access.
- 2- Mobility: the itinerant agents travel from system to system to perform their tasks, the user has no way to know where their agents have been.
- 3- Viruses: agents and viruses are very much alike. Therefore, if the environment is created for agents, this may also mean that virus can breed in this environment.
- 4- Trust: agents can communicate with one another. They can communicate with any agent they want in order to perform their task. However, agents cannot predict the reliabilities of other agents. Unlike human beings, agents cannot assign different reliabilities to other agents.

Since agents work independently with all the authorities delegated to them, there is usually no way that users can detect if, for example, an e-mail agent contains a virus, or prevent agents from executing virus programs. Agents can access anywhere that the users have authorized and this increases the chance that agents may spread the viruses with which they have been infected. These are security issues in agent systems that are different from conventional systems [32].

## 2.9 Multi-Agent Systems

The interest, in the Software Engineering Approach, is focused in Multi Agent Systems (MAS). A MAS is a set of autonomous agents, which work cooperatively to achieve their goals. Each agent may interact with its environment or with other agents, using high-level communication languages and protocols, in order to coordinate its activities and to obtain services and/or information [36].

Defining MAS is not straightforward. However, almost all the definitions given in the literature conceive a MAS as a system composed of cooperative or competitive agents that interact with one another in order to achieve individual or common goals. From the software engineering point of view, one of the most important characteristics of a MAS is that the final set of agents is generally not given at design time (only the initial set is specified), but rather at run time. This basically means that, in practice, MASs are based on open architectures that allow new agents to dynamically join and leave the system. The major difference with the Object Oriented approach, where objects can also be given at run time, join, and leave the system dynamically, is that agents can do these autonomously showing proactive behaviors not completely predictable a priori [37].

Agents (adaptive or intelligent agents and multi- agent systems) constitute one of the most prominent and attractive technologies in computer science at the beginning of this new century. Agent and multi-agent system technologies, methods, and theories are currently contributing to many diverse domains. These include information retrieval, user interface design, robotics, electronic commerce, computer mediated collaboration, computer games, education and training, smart environments, ubiquitous computers, and social simulation [40].

## Chapter Three

### Intelligent Agent

The term "Intelligent Agent" is used in this chapter, instead to emphasize on the ability of autonomy. The intelligent agents do have some other attributes that can make them be thought of as intelligent when used in the real world. Here are these attributes: [10]

- 1- Delegation: the agent performs a set of tasks on behalf of users that are approved by them.
- 2- Communication skills: the agent is able to communicate with users in order to receive the instruction, display task status and complete the task. It also needs to be able to communicate with other agents in order to achieve goals.
- 3- Monitoring: the agent should be able to monitor the environment in order to perform its task autonomously and correctly with respect to the goal.
- 4- Actuation: the agent should be able to affect the environment through a mechanism for autonomous operation.
- 5- Intelligence: the agent needs to understand the monitored information in order to conduct the suitable autonomous operation.

#### **3.1 Definition of intelligent Software Agent**

Intelligent software agents are defined as being a software program that can perform specific tasks for a user and possesses a degree of intelligence that permits it to perform parts of its tasks autonomously and to interact with its environment in a useful

manner [41]. In addition, there is another definition, which may be better where intelligent agents are defined as a set of independent software tools linked with other applications and database running within one or several computer environments [1].

The intelligence that an intelligent agent has is called computational intelligence. It is obtained from several study fields including intentional systems, production systems, reasoning theory, and neural networks.

### **3.2 Intelligent Agents and Artificial Intelligence**

The discipline of intelligent agents has emerged largely from research in artificial Intelligence (AI). In fact, one way of defining AI is as the problem of building an intelligent agent [39]. However, it is important to distinguish between the broad intelligence that is the ultimate goal of the AI community, and the intelligence we seek in agents. The only intelligence requirement we generally make of our agents is that they can make an acceptable decision about what action to perform next in their environment, in time for this decision to be useful. Other requirements for intelligence will be determined by the domain in which the agent is applied: not all agents will need to be capable of learning. Agents are simply software components that must be designed and implemented in much the same way that other software components are. However, AI techniques *are* often the most appropriate way of building agents [30].

### **3.3 Intelligent Agent Model**

In Figure (3-1), Intelligent Agent Model, which describes only the high-level attributes of an intelligent model, it shows that both the task skill module and

communication skill module access the knowledge module. These accesses represent that intelligent agents, will work knowledgably. The following sections describe each component in detail [10].

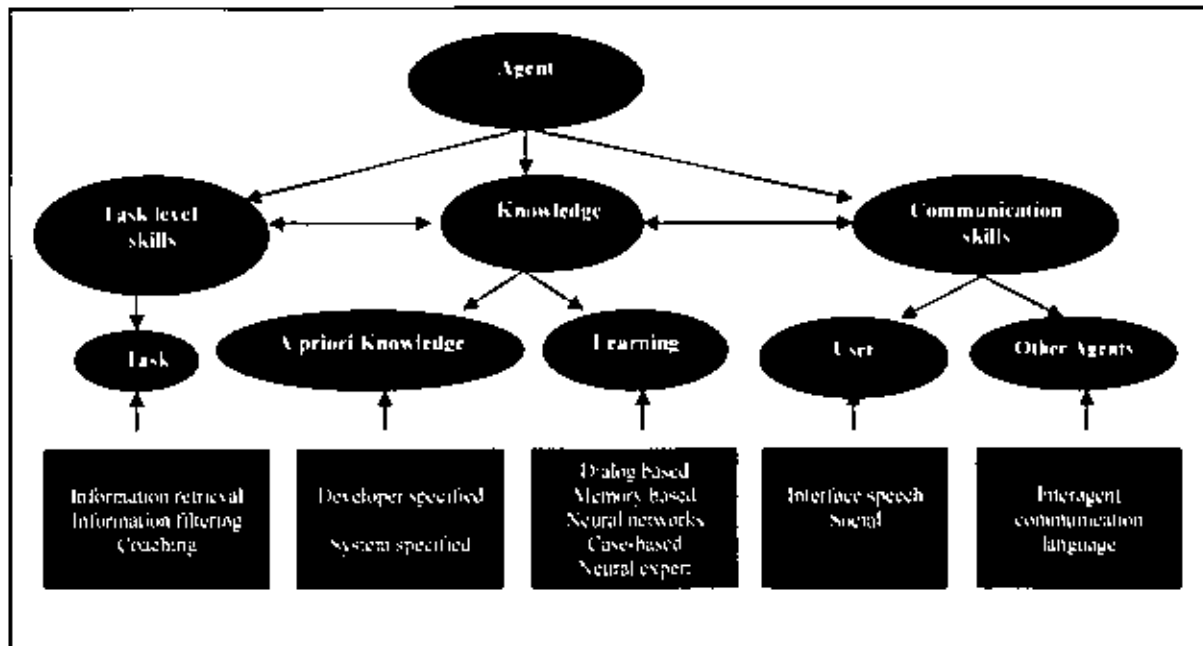


Figure (3-1). Intelligent Agent Model

### 3.3.1 Task Level Skills

The task level skills module is consist of skills needed for the agent to accomplish its goal. This module also tells what functionalities the agent has. For example, the skill of retrieving information, filtering information, database querying and coaching. These skills require an agent to have the capability of perceiving its environment through its sensors and working upon the environment to perform the tasks [10].

### **3.3.2 Knowledge Module**

The knowledge module consists of rules that the agent should follow when it tries to complete its task. Intelligent agents are autonomous and they are able to complete the given task without user intervention. This autonomy is achieved by having the knowledge of the environment built-in by designers in their knowledge module. Knowledge acquirement for intelligent agents can be done by using the following techniques: [10]

- 1- **Developer specified:** according to the application domain and the potential users, developers specify the influence mechanism in the knowledge base of the agent. This knowledge base is formatted in rules and/or frames. The major disadvantage of using knowledge base in intelligent agents is that it will lack the customizability after these agents are deployed.
- 2- **User specified:** this approach uses a rule base. The rule base can be programmed by users. This approach is good for satisfying the real need of the users. However, the disadvantage of the rule-based agents is that users have to be responsible for partial programming of the agents.
- 3- **Derived from other knowledge sources:** the agent communication language allows agents to acquire knowledge from agent communities.
- 4- **Learned by the system:** this approach allows agents to acquire knowledge from the users and the environment.

### **3.3.3 Communication Skill**

This module of the intelligent agent includes the communication with users and communication among agents. In user communication, it is usual that the user interface is

used. This interface can be of many different forms ranging from e-mail messages to dialog boxes. These interfaces can acquire user information for agents or can display finding or status of agents for users. For intra-agent communication, a communication language (ACL) is often necessary [10].

### **3.4 Taxonomy of Intelligent Agent**

There is a systematic way that agents can be divided into categories based on their environment and then further subdivided each category based on their task [].

#### **1. Desktop Agents:**

- i. Operating system agents: Interface agents, which provide help for users to deal with desktop operating systems.
- ii. Application agents: Interface agents, which provide help for users to deal with particular applications.
- iii. Application suite agents: Interface agents, which provide help for users to deal with a suite of applications.

#### **2. Internet Agent:**

- i. Web search agents: Internet agents that provide search service for users.
- ii. Web server agents: Internet agents that reside on a web server to provide services.



- iii. Information filtering agents: Internet agents that can filter out electronic information according to the users' preferences.
- iv. Information retrieving agents: Internet agents that deliver a set of information according to the users' preferences.
- v. Notification agents: Internet agents that notify the users when the user-interested events occur.
- vi. Service agents: Internet agents that provide user specialized services.
- vii. Mobile agents: Internet agents that travel around the network to execute user specified task.

### **3. Intranet Agent:**

- i. Collaborative customization agents: Intranet agents that allow for automatic workflow process in a business unit.
- ii. Process automation agents: Intranet agents that make the workflow process automatic.
- iii. Database agents: Intranet agents that provide service to database users.
- iv. Resource brokering agents: Agents that perform resource allocation for users in client/server architecture.

### **3.5 Concepts for Intelligent Agents**

This research described the set of concepts, which use in developing intelligent agent application, we believe that these are necessary and sufficient for building the sort

of applications appropriately approached using BDI agents. Property of agents: they are situated (see Figure (3-2)).[]

Thus, we have actions and percepts. Internally, the agent is making a decision: from the set of possible actions as it is selecting an action (or actions) to perform (a 2 as).

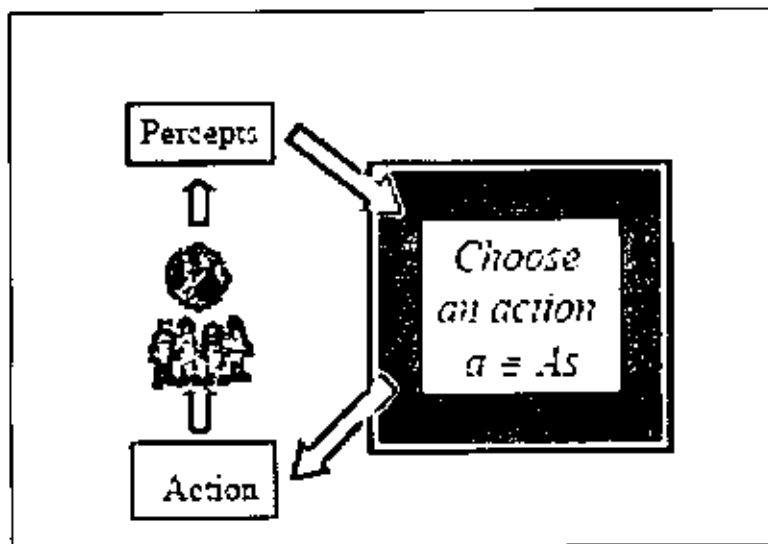


Figure (3-2). Agents are situated<sup>1</sup>

Loosely speaking, where the description of the agent's internal working contain a statement of the form "[select] X 2 Y" then we have a decision being made, thus the type of decisions being made depend on the *internal agent architecture*.

An **action** is something, which an agent does, such as *move north* or *squirt*. Agents are situated, and an action is basically an agent's ability to effect its environment. In their simplest form actions are automatic and instantaneous and either fail or succeed. In the more general case actions can be durational (encompassing behaviors over time)

<sup>1</sup> M. Winikoff, L. Padgham, J. Harland. "Simplifying the Development of Intelligent Agents ", Research paper, RMIT University, Melbourne, Australia.

and can produce partial effects; for example a failed *move to* action may well have changed the agent's location. In addition to actions, which directly affect the agent's environment, we also want to consider "internal actions". These correspond to an ability, which the agent has which is not structured in terms of plans and goals [29].

Typically, the ability is a piece of code which either already exists or would not benefit from being written using agent concepts, for example image processing in a vision sub-system.

A **percept** is an input from the environment, such as the location of a fire and an indication of its intensity. The agent may also obtain information about the environment through sensing actions.

A **decision**: The essence of intelligent agents is rational decision-making. There are a number of generic, non-application-specific questions, which intelligent agents must answer, such as: *Which* action shall we perform now? *Which* goal do we work on now? *How* shall we attempt to realize this goal? *Where* shall we go now (for mobile agents)? And *who* shall we interact with (for social agents)? Mechanisms to answer these kinds of questions are core intelligent agent processes.

They result in decisions, which must fulfill rationality conditions, in that we expect that decisions be persistent and only be revisited when there is a good reason for doing so. It is also important that the answer to the questions not be trivial: if an agent only has a single goal at a time and a single means of realizing this goal, then we have reduced the agent to the special case of a conventional program and there is no scope for decision-making or for flexible, intelligent behavior.

Note that although the concept of a decision is fundamental to intelligent agents, it is not always necessary to represent the decisions explicitly. For example, the decision regarding choice of goal could be represented using a "current goal" variable, which is updated when a decision is made [29].

We now consider the internal workings of the agent (see Figure (3-3)). We want our intelligent agents to be both *proactive* and *reactive*.

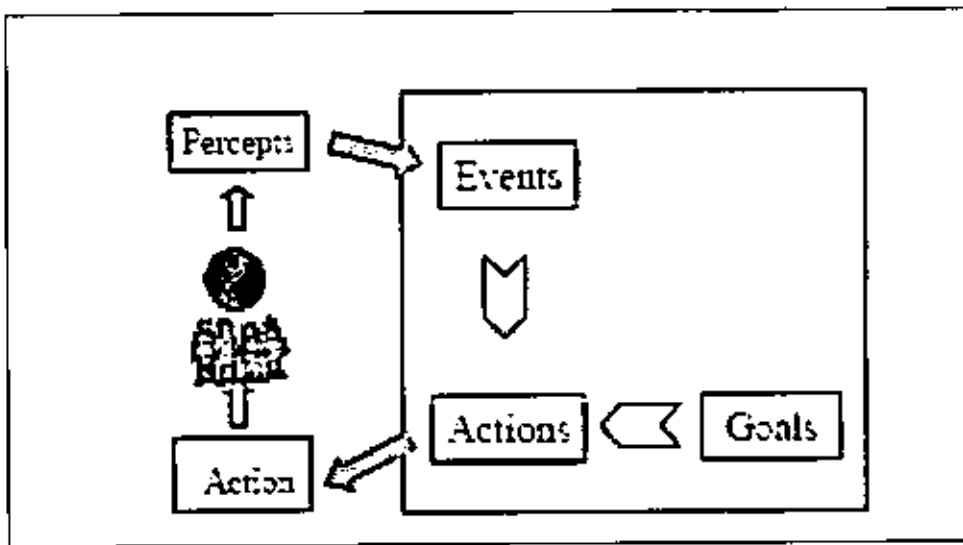


Figure (3-3). Proactive Agents have Goals, Reactive Agent have Events<sup>1</sup>

A proactive agent is one, which pursues an agenda over time. The agent's proactiveness implies the use of goals and modifies the agent's internal execution cycle: rather than select an action one at a time, we select a goal, which is persistent and constrains our selection of actions.

<sup>1</sup> M. Winikoff, L. Padgham, J. Harland, "Simplifying the Development of Intelligent Agents", Research paper, RMIT University, Melbourne, Australia.

A *reactive* agent is one, which will change its behavior in response to changes in the environment. An important aspect in decision-making is balancing proactive and reactive aspects. On the one hand, we want the agent to stick with its goals by default; and we want it to take changes in the environment into account the key to reconciling these aspects, thus making agents suitably reactive, is to identify *significant* changes in the environment. These are *events*. We distinguish between percepts and events: an event is an interpreted percept, which has significance to the agent. For example, seeing a fire is a percept. This percept could give rise to a *new fire* event or a *fire under control* event depending on history and possibly other factors.

A *goal* (variously called "task", "objective", "aim", or "desire"), is something the agent is working on or towards, for example *extinguish fire*, or *rescue civilian*. Often goals are defined as states of the world, which the agent wants to bring about; however, this definition rules out maintenance goals (e.g. "maintain cruising altitude") and avoidance goals, or safety constraints (e.g. "never move the table while the robot is drilling"). Goals give the agent its autonomy and proactiveness.

An important aspect of proactiveness is the persistence of goals: if a plan for achieving a goal fails then the agent will consider alternative plans for achieving the goal in question. We have found that goals require greater emphasis than is typically found in existing systems. It is important for the developer to identify the top-level goals of the agent as well as subsidiary goals, which are used in achieving main goals.

we differentiate between top-level goals and subsidiary goals in those subsidiary goals are not important in their own right and may therefore be treated differently in the reasoning process than top-level goals.

An event is a significant occurrence. Events are often extracted from percepts, although they may be generated internally by the agent, for example on the basis of a clock. An event can trigger new goals, cause changes in information about the environment, and/or cause actions to be performed immediately [29].

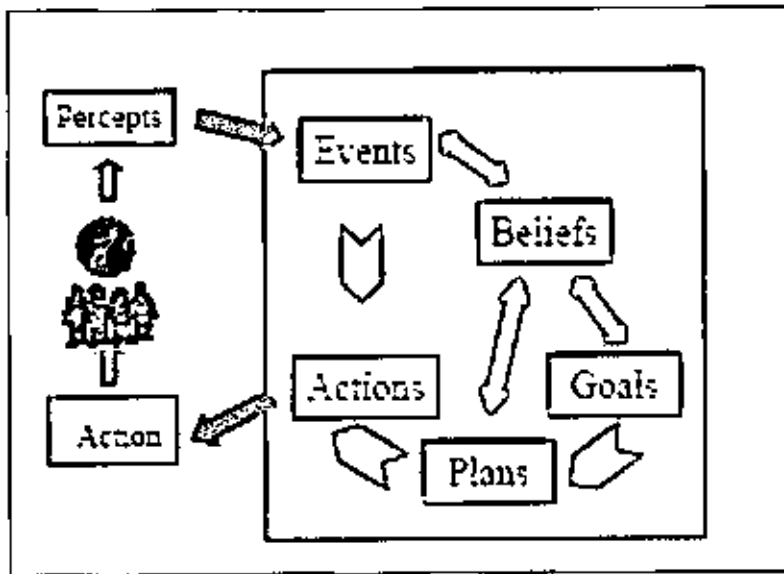


Figure (3-4). Adding Plans and Beliefs<sup>1</sup>

Actions generated directly by events correspond to "reflexive" actions, executed without deliberation. Events are important in creating reactive agents in that they identify important changes, which the agent needs to react to.

Agents in realistic applications usually have limited computational resources and limited ability to sense their environment. Thus, the auxiliary concepts of **plan** and **belief** are needed. Beliefs are effectively a cache for perceived information about the environment, and plans are effectively a cache for ways of pursuing goals (see Figure (3-

<sup>1</sup> M. Winikoff, L. Padgham, J. Harland. "Simplifying the Development of Intelligent Agents". Research paper, RMIT University, Melbourne, Australia.

4)). Although both of these concepts are “merely” aids in efficiency, they are not optional. Beliefs are essential since an agent has limited sensory ability and also it needs to build up its knowledge of the world over time. Plans are essential for two reasons.

The first is pure computational efficiency: although planning technology and computational speed are improving, planning from action descriptions is still incompatible with real-time decision-making.

The second reason is that by providing a library of plans we avoid the need to specify each action’s preconditions and effects: all we need to provide for an action is the means to perform it. This is significant in that representing the effects of continuous actions operating over-time and space in an uncertain world in sufficient detail for first principles planning is unrealistic for large applications.

A **plan** is a way of realizing a goal. for example, a plan for achieving the goal *extinguishes fire* might specify the three steps: plan a route to the fire, follow the route to the fire, and squirt the fire until it has been put out. Although the concept of a plan is common, there is no agreement on the details. From our point of view, it is not necessary to adopt a specific notion of a plan; rather we can specify abstractly that a plan for achieving a goal provides a function, which returns the next action to be performed. This function takes into account the current state of the world (beliefs), what actions have already been performed, and might involve sub-goals and further plans. For computational reasons it is desirable for this to at least include a “library of recipes” approach, rather than requiring construction of plans at runtime from action descriptions.

A **belief** is some aspect of the agent's knowledge or information about the environment, self or other agents. For example an agent might believe that there is a fire at X because it saw it recently, even it cannot see it now.

These concepts (actions, percepts, decisions, goals, events, plans, and beliefs) are related to each other via the bold **execution cycle** of the agent.

An agent's execution cycle follows a sense think- act cycle, since the agent is situated.

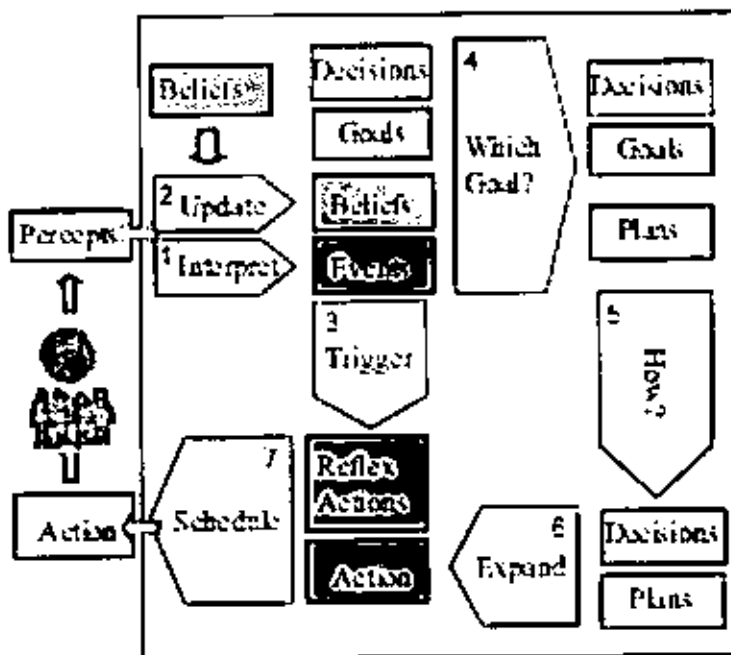


Figure (3-5). Agent Execution Cycle<sup>1</sup>

The think part of the cycle involves rational decision-making, consisting of the following steps :( depicted in Figure (3-5)) [29]

<sup>1</sup> M. Winikoff, L. Padgham, J. Harland. "Simplifying the Development of Intelligent Agents ". Research paper, RMIT University, Melbourne, Australia.



- 1- Percepts are interpreted (using beliefs) to give events
- 2- Beliefs are updated with new information from percepts
- 3- Events yield reflexive actions and/or new goals
- 4- Goals are updated, including current, new and completed goals.
- 5- If there is no selected plan for the current goal, or if the plan has failed, or if reconsideration of the plan is required (due to an event) then a plan is chosen.
- 6- The chosen plan is expanded to yield an action
- 7- Action(s) are scheduled and performed.

By comparison, the BDI abstract execution cycle consists of the following steps:

- i. Use events to trigger matching plans (options).
- ii. Select a subset of the options
- iii. Update the intentions and execute them.
- iv. Get external events.
- v. Drop successful and impossible attitudes.

The execution cycle presented here differs from the BDI execution cycle in a number of ways including the use of reflexive actions, the derivation of events by interpreting percepts, the process of going from goals to plans to actions, and increased reasoning about goals [29]. Table (3-1) offer a summary of agent concepts [10].

Agency property	Definition
<b>Agent hood properties</b>	
Interaction	An agent communicates with the environment and other agents by means of sensors and effectors.
Adaptation	An agent adapts/modifies its mental state according to messages received from the environment.
Autonomy	An agent is capable of acting without direct external intervention; it has its own control thread and can accept or refuse a request message.
<b>Additional properties</b>	
Learning	An agent can learn based on previous experience while reacting and interacting with its environment.
Mobility	An agent is able to transport it self from one environment in a network to another.
Collaboration	An agent can cooperate with other agents in order to achieve its goals and the systems goals.

Table (3-1). Agent Concept.

### 3.6 Structure of Intelligent Agents

So far, we have talked about agents by describing their *behavior* and the action that is performed after any a given sequence of percepts. Now, the job of AI is to design the **agent program**: a function that implements the agent mapping from percepts to actions. We assume this program will run on some sort of computing device, which we will call the **architecture**. Obviously, the program we choose has to be one that the architecture will accept and run. The architecture might be a plain computer, or it might include special-purpose hardware tasks, such as processing camera images or filtering audio input. It might also include software that provides a degree of insulation between

the raw computer and the agent program, so that we can program at a higher level. In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program's action choices to the effectors as they are generated. The relationship among agents, architectures, and programs can be summed up as follows:

$$\textit{Agent} = \textit{Architecture} + \textit{Program}$$

Before we design an agent program, we must have a good idea of the possible percepts and actions, what goals or performance measures the agent is supposed to achieve, and what sort of environment it will operate. These come in a wide variety. Table (3-2) shows the basic elements for a selection of agent types.

We also include in this list of agent types some programs that seem to operate in the entirely artificial environment defined by keyboard input and character output on a screen. "Surely," one might say, "This is not a real environment, is it?" In fact, what matters is not the distinction between "real" and "artificial" environments, among the behavior of the agent, the percept sequence generated by the environment, and the goals that the agent is supposed to achieve.

Some "real" environments are actually quite simple. For example, a robot designed to inspect parts as they come by on a conveyer belt can make use of a number of simplifying assumptions: that the lighting is always just so, that the only thing on the conveyer belt will be parts of a certain kind, and that there are only two actions which are: accept the part or mark it as a reject.

Agent type	percepts	actions	Goals	Environment
Medical diagnosis system	Symptoms findings, patients answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize students score on test	Set of students

Table (3-2). The basic elements for a selection of agent types.

In contrast, some software agents (or software robots or softbots) exist in rich, unlimited domains. Imagine a softbot designed to fly a flight simulator for a 747. The simulator is a very detailed, complex environment, and the software agent must choose from a wide variety of actions in real time. Alternatively, imagine a softbot designed to scan online news sources and show the interesting items to its customers. To do well, it will need some natural language processing abilities, it will need to learn what each

customer is interested in, and it will need to dynamically change its plans when, for example, the connection for one news source crashes or a new one comes online.

Some environments blur the distinction between "real" and "artificial." In the ALIVE environment, software agents are given as percepts a digitized camera image of a room where a human walks about. The agent processes the camera image and chooses an action. The environment also displays the camera image on a large display screen that the human can watch, and superimposes on the image a computer graphics rendering of the software agent. One such image is a cartoon dog, which has been programmed to move toward the human (unless he points to send the dog away) and to shake hands or jump up eagerly when the human makes certain gestures [39].

The most famous artificial environment is the Turing Test environment, in which the whole point is that real and artificial agents are on equal footing, but the environment is challenging enough that it is very difficult for a software agent to do as well as a human.

### **3.7 Applications of Intelligent Agents**

Intelligent agents are a new paradigm for developing software applications. Currently, agents are the focus of intense interest on the part of many sub-fields of computer science and artificial intelligence.

Agents are being used in an increasingly wide variety of applications, ranging from comparatively small systems such as email filters to large, open, complex, mission critical systems such as air traffic control [32].

The current applications of agents are of a rather experimental and ad hoc nature. Besides universities and research centres, a considerable number of companies, like *IBM* and *Microsoft*, are doing research in the area of agents.

To make sure their research projects will receive further financing, many researchers & developers of such companies (but this is also applicable to other parties, even non-commercial ones) are nowadays focusing on rather basic agent applications, as these lead to demonstrable results within a definite time. Examples of this kind of agent applications are:[32]

1. Agents who partially or fully handle someone's e-mail;
2. Agents who filter and/or search through (Usenet) news articles looking for information that may be interesting for a user;
3. Agents that arrange for gatherings such as a meeting, for instance by means of lists provided by the persons attending or based on the information (appointments) in the electronic agenda of every single participant.

The current trend in agent developments is to develop modest, low-level applications. Yet, more advanced and complicated applications are more and more being developed as well. At this moment, research is being done into separate agents, such as mail agents, newsagents and search agents. These are the first step towards more integrated applications, where these single, basic agents are used as the building blocks. Expectations are that this will become the trend in the next two or three years to come. (Note that this does not mean that there will be no or little interesting developments and opportunities in the area of smaller, more low-level agent applications.).

Eight application areas are identified where now (or in the near future) agent technology is (or will be) used [32]. These areas are:

### **1. Systems and Network Management:**

Systems and network management is one of the earliest application areas to be enhanced using intelligent agent technology. The movement to client/server computing has intensified the complexity of systems being managed, especially in the area of LANs, and as network centric computing becomes more prevalent, this complexity escalates further. Users in this area (primarily operators and system administrators) need greatly simplified management, in the face of rising complexity.

Agent architectures have existed in the systems and network management area for some time, but these agents are generally "fixed function" rather than intelligent agents. However, intelligent agents can be used to enhance systems management software. For example, they can help filter and take automatic actions at a higher level of abstraction, and can even be used to detect and react to patterns in system behaviour. Further, they can be used to manage large configurations dynamically [32].

### **2. Mobile Access / Management:**

As computing becomes more pervasive and network centric, computing shifts the focus from the desktop to the network. users want to be more mobile. Not only do they want to access network resources from any location, they want to access those resources despite bandwidth limitations of mobile technology such as wireless communication, and despite network volatility. Intelligent agents, who (in this case) reside in the network

rather than on the users' personal computers, can address these needs by persistently carrying out user requests despite network disturbances. In addition, agents can process data at its source and ship only compressed answers to the user, rather than overwhelming the network with large amounts of unprocessed data [32].

### **3. Adaptive User Interfaces:**

Although the user interface was transformed by the advent of graphical user interfaces (GUIs), for many, computers remain difficult to learn and use. As capabilities and applications of computers improve, the user interface needs to accommodate the increase in complexity. As user populations grow and diversify, computer interfaces need to learn user habits and preferences and adapt to individuals.

Intelligent agents (called *interface agents*) can help with both these problems. Intelligent agent technology allows systems to monitor the user's actions, develop models of user abilities, and automatically help out when problems arise.

When combined with speech technology, intelligent agents enable computer interfaces to become more human or more "social" when interacting with human users [32].

### **4. Information Access and Management:**

Information access and management is an area of great activity, given the rise in popularity of the Internet and the explosion of data available to users. Here, intelligent agents are helping users not only with search and filtering, but also with categorisation, prioritisation, selective dissemination, annotation, and (collaborative) sharing of information and documents:[32]



## **5. Collaboration:**

Collaboration is a fast-growing area in which users work together on shared documents, using personal video-conferencing, or sharing additional resources through the network. One common denominator is shared resources; another is teamwork. Both of these are driven and supported by the move to network centric computing.

Not only do users in this area need an infrastructure that will allow robust, scalable sharing of data and computing resources, they also need other functions to help them actually build and manage collaborative teams of people, and manage their work products. One of the most popular and most heard-of examples of such an application is the *groupware* packet called *Lotus Notes* [32].

## **6. Workflow and Administrative Management:**

Administrative management includes both workflow management and areas such as computer/telephony integration, where processes are defined and then automated.

In these areas, users need not only to make processes more efficient, but also to reduce the cost of human agents. Much as in the messaging area, intelligent agents can be used to ascertain, then automate user wishes or business processes [32].

## **7. Electronic Commerce:**

Electronic commerce is a growing area fuelled by the popularity of the Internet. Buyers need to find sellers of products and services, they need to find product information (including technical specifications, viable configurations, etc.) that solve their problem, and they need to obtain expert advice both prior to the purchase and for

service and support afterward. Sellers need to find buyers and they need to provide expert advice about their product or service as well as customer service and support. Both buyers and sellers need to automate handling of their "electronic financial affairs".

Intelligent agents can assist in electronic commerce in a number of ways. Agents can "go shopping" for a user, taking specifications and returning with recommendations of purchases, which meet those specifications. They can act as "salespeople" for sellers by providing product or service sales advice, and they can help troubleshoot customer problems [32].

### **8. Mail and Messaging:**

Messaging software (such software for e-mail) has existed for some time, and is also an area where intelligent agent function is currently being used. Users today want the ability to automatically prioritise and organise their e-mail, and in the future, they would like to do even more automatically, such as addressing mail by organisational function rather than by person.

Intelligent agents can facilitate all these functions by allowing mail handling rules to be specified ahead of time, and letting intelligent agents operate on behalf of the user according to those rules. Usually it is also possible (or at least it will be) to have agents deduce these rules by observing a user's behaviour and trying to find patterns in it: [32]

### **3.8 Intelligent Information Management**

The automated management of electronic documents, web pages, newsgroup articles, email, and other electronic information is an active area of research for

several fields including Artificial Intelligence, Machine Learning, and Information Retrieval. Text-based information is challenging for several reasons, including its lack of structure and constraints, its large state space, its varied content, and the importance and difficulty of representing the context in which the information occurs. These challenges have led researchers to carefully structure and limit the information in many AI systems. For existing information types such as web pages and email, it is less likely that the structured approach is applicable. However, in some cases, there is structure; an intelligent system should use it when available. For example, although the body of email is unstructured, the sender, routing, and recipient information are explicitly labeled.

One solution is to design an agent capable of extracting useful statistical features from documents. Once transformed into a feature vector, learning algorithms can be trained to apply the appropriate actions. The use of a machine learning approach allows the email agent to learn from the user. By watching the user interact with documents, an agent can learn the information tasks, the user preferences and interests, and example solution techniques [8].

In some systems, the features are fixed by the programmer; for example, a particular set of words or fields may be pre-specified.

In my system, the agent learns the features automatically based on the task or on user suggestions about the training e-mail's content. Because the features represent the conceptual contents in the email, we call these features "concept" features. The agent looks for these concepts in the data and then learns to perform the desired actions .

### 3.9 Email Filtering

Agent can employ several techniques. Agents are created to act on behalf of its users in carrying out difficult and often time-consuming tasks; most agents today employ some type of artificial intelligence technique to assist the users with their computer-related tasks, such as reading e-mail, filtering information. Some agents can be trained to learn through example in order to improve the performance of the tasks [27].

Email provides an example of a rich information management domain. Email is typically short, less than a hundred lines, and contains a limited amount of structure. The body of an email is usually unstructured text, while the headers provide some tagged information. For example, an agent knows a priori the meaning of a from header: it defines the sender; similarly, the date header provides a known type of information. The Subject header is problematic. It says something about the contents of the email, but not in a fixed or pre-specified way, even for the headers with known content, the utility of their information is limited. Knowing the sender of an email is useful, but often the same sender discusses different topics, some form of content understanding is required, the current generation of email filtering packages, requires user-written rules to interpret and sort email. Rule-based systems can be challenging for users due to the difficulty of not only determining the rule that achieves the desired result, but also writing it with the correct syntax [8].

## Chapter Four

### Prometheus as an agent-Oriented Analysis and Design

#### Methodology

##### 4.1 introductions

Agent-oriented Software Engineering (AOSE) has become an active area of research in recent years. The main purpose of AOSE is to create methodologies and tools that enable inexpensive development and maintenance of agent-based software. In addition, the software should be flexible, easy-to-use, scalable, and of high quality [10].

Agent-Oriented methodologies provide a set of mechanisms and models for developing agent-based systems. Most agent-oriented methodologies follow the approach of extending existing software engineering methodologies to include abstractions related to agents. Agent methodologies capture concepts like conversations, goals, believes, plans or autonomous behavior. Most of them take advantage of software engineering approaches to design MAS, and benefit from UML and/or AUML diagrams to represent these agent abstractions [ 24].

Prometheus, Gaia, MESSAGE , MaSE and Tropos methodologies of agent-oriented methodologies is maturing rapidly and that the time has come to begin drawing together the work of various research groups with the aim of developing the "next generation" of agent-oriented software engineering methodologies [17].

There are many methodologies with different strengths and weakness and different specialized features to support different applications domains. Clearly, there is not a widely used or general-purpose methodology, but we took into account some issues

like the modeling diagrams used, the kind of application domain it is appropriated for, and above all, the level of detail provided at the design phase and the available documentation [24].

The analysis and design methodology supports the processes of eliciting the system requirements, analyzing the organizational environment, and designing the information system. The early requirements and analysis phases deal with specifying the requirements of the system, while the design phase deals with the detailed specification of how the system will accomplish the requirements. In the analysis phase, the application domain is modeled using abstract notions, and the conceptual model is generated. The conceptual model is a model of the organization. In the design phase, the information system itself is modeled using concrete concepts, which relate directly to components of the software system. The design model should provide detailed information of how the system would look like, but it should not provide instruction of *how* to implement the design. The design model is similar to an architect's plan – it describes the exact form of the end product, without specifying the technique and methods that should be used to realize this plan. Although various analysis and design methodologies use various names to describe the models and the phases, there is generally an agreement on the structure of the A&D process. It is seen as a collection of transformations, from the very abstract model (the conceptual model) to the most concrete model (the detailed design model), where each transformation shrinks the space of possible end products, and introduces more and more implementation bias. Ideally, the transformations should be straightforward i.e. leaving no degrees of freedom to the designer, but this is not the case in most methodologies [33].

The importance of a structured A&D process, which can be found in most systems analysis and design books, is twofold [33]:

- 1- During development: a formal analysis and design process ensures that the system is developed according to its requirements. It makes implementation easier, and allows for fewer mistakes. The A&D process turns software engineering into a science and not an art. A&D methodologies provide the system analysts, designers, and programmers a common language to exchange ideas about the system and share common work processes.
- 2- During maintenance: the analysis and design models capture the early requirements that shaped the system, as well as the ideas and constraints that directed the system designers.

When there is a need to change or re-evaluate the information system, or migrate it to a different platform, the documented A&D models allows programmers to locate the exact spot where the modification is needed, without having to digest thousands of code lines.

#### **4-2 Differences between (Agent Oriented) and (Object Oriented) methodologies**

Although A&D methodologies have been used in software systems engineering for a long time and current object-oriented techniques have matured and gained wide acceptance, they are unsuitable for developing agent-oriented information systems. The reason is that agent-oriented systems use different abstractions, and the definition of an object cannot capture the structure and behavior of agents. The important ones are the

agent's flexible, autonomous problem solving behavior (the ability of the agent to reason about its environment and choose an action that draws him closer to achieving his goal) and the richness of agents' interactions. An intuitive way of describing the differences is the slogan "objects do it because they have to, while agents do it because they want to (or choose to)" [5].

Due to the aforementioned differences between objects and agents, object-oriented methodologies cannot be used as is to develop agent-oriented information systems. While some ideas from object-oriented A&D techniques can be borrowed, there is clearly a need for new methodologies, which are rich enough to capture the structure and behavior of the individual agent (e.g., cognition, intelligence, rational, and emotions) and those of the agent society (e.g., trust, fraud, commitment, and social norms) [33].

Figurer (4-1) shows a genealogy that includes many of the methodological proposals. We reproduce it in Figurer (4-1). It presents a significant number of proposals as well as their origin. The recognized genesis is symbolized by OO (object-oriented), RE (requirement engineering) and KE (knowledge engineering). *P* states other unclear sources [5].



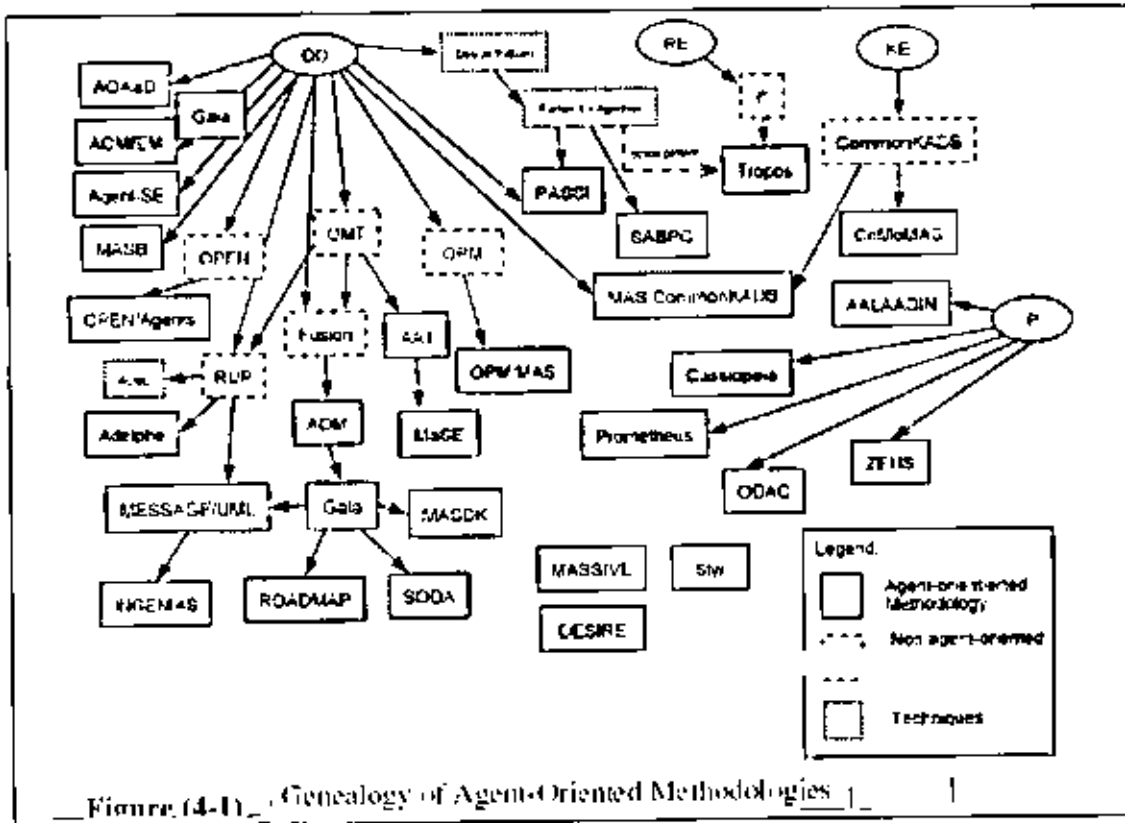
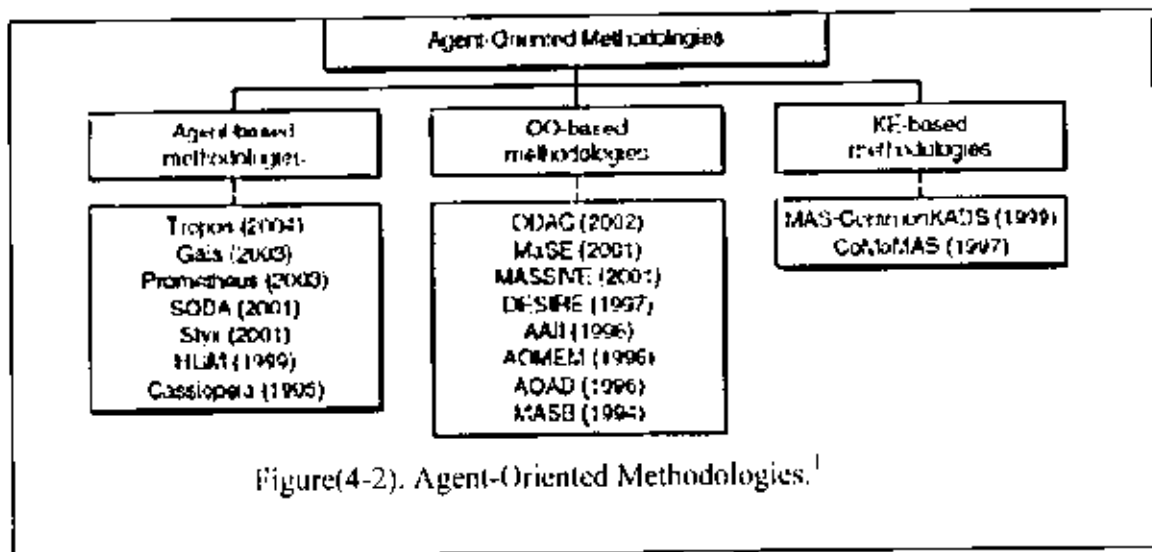


Figure (4-1). Genealogy of Agent-Oriented Methodologies

In the second scheme, agent concepts have been recognized constituting a specific conceptual stream. We show this classification in Figure (4-2).



Figure(4-2). Agent-Oriented Methodologies.

We will use Prometheus methodology as an agent-Oriented Analysis and Design Methodology for illustrating and Discussion of phases of analysis and design of agent system

### 4.3 Prometheus

The Prometheus methodology is a detailed AOSE methodology, which aims to cover all of the major activities required in developing agent systems. The aim of Prometheus is to be usable by expert and non-expert users. The methodology uses an iterative process, which consists of three phases: system specification, architectural design and detailed design. Each of these phases is explained in details below.

Prometheus [19] is an intelligent agent development methodology. A key feature of this methodology is that it covers all phases of development - specification, design, implementation and testing/debugging. Like most modern software engineering methodologies, Prometheus is intended to be applied in an iterative manner.

It is widely accepted in the agent research community that a key issue in the transition of agents from research labs to industrial practice and the need for a mature software engineering methodology for specifying and designing agent systems. In this chapter, we describe the *Prometheus* methodology, which aims to address this need.

Prometheus is intended to be a *practical* methodology. As such, it aims to be complete: providing everything that is needed to specify and design agent systems. Other distinguishing features of the Prometheus methodology are: [19]

- 1- Prometheus is *detailed* – it provides detailed guidance on *how* to perform the various steps that form the process of Prometheus.

- 2- Prometheus supports (though is not limited to) the design of agents that are based on goals and plans. We believe that a significant part of the benefits that can be gained from agent-oriented software engineering comes from the use of goals and plans to realize agents that are flexible and robust.
- 3- Prometheus covers a range of activities from requirements specification through to detailed design.
- 4- The methodology is designed to facilitate tool support, and tool support exists in the form of the *Prometheus Design Tool* (PDT) which is freely available.

Before we present Prometheus, it is important to consider the question "what is a methodology?" If we view a methodology as consisting purely of notations for describing designs or as consisting only of a high-level process then we end up with a very different result.

We adopt a pragmatic stance: rather than debating what should and should not be considered part of a methodology. In particular, the Prometheus methodology as described in [20] includes a description of **concepts** for designing agents, a **process**, a number of **notations** for capturing designs, as well as many "tips" or **techniques** that give advice on how to carry out the steps of Prometheus' process.

Since design is a human activity that is inherently about tradeoffs, rather than about finding the single best design (which often does not exist), it is not possible to provide hard and fast rules. However, it is important to provide detailed techniques and guidelines for carrying out steps. We would like to stress that Prometheus is a general purpose methodology. Although the detailed design phase makes some assumptions about the agent architecture, the rest of the methodology does not make these

assumptions. It is not possible to provide a detailed methodology that proceeds to detailed design and towards implementation without making some assumptions.

#### 4.3.1 Agent Concepts in Prometheus methodology

Before we proceed to present the process, notations, and techniques that are associated with the Prometheus methodology, we begin by discussing *concepts* [29]:

The reason why it is important to consider and discuss concepts is that the concepts are the foundation, which a software engineering methodology builds upon. For instance, object-oriented methodologies assume that the designer is familiar with concepts such as objects, classes and inheritance.

The concepts that are appropriate for designing agents are, not surprisingly, different from those that are used for objects. Whereas the concepts of object-oriented programming are well known, those associated with agent-oriented programming are not, and so we feel that it is useful and important to discuss them. In considering what concepts are appropriate for designing agent systems we take as a starting point, the definition of an intelligent agent as being software that is situated, autonomous, reactive, proactive, flexible, robust and social [20, 31]. Let us begin with a basic property: agents are situated in an environment. As a result, capturing the agent's interface with the environment is important (and is done as part of Prometheus' system specification phase). The agent's interface is expressed in terms of (*percepts*) providing information from the environment and *actions*, which the agent(s) can perform to change the environment.

Agents are also *proactive* and *reactive*. A proactive agent is one that pursues goals, and so a key design (and implementation!) concept that is used to build agents with this property is *goals*. A *reactive* agent is one that responds to significant occurrences (*events*). These events may be percepts from the environment, but may also be messages from another agent, or even internal occurrences. There are two other concepts that we believe are important to designing intelligent agents: beliefs and plans. Beliefs are important because agents need to store state, unless they are to be purely reactive. Plans are important because there is often not enough time to plan from first principles, and having a library of plans to achieve goals or respond to events enables agents to respond more rapidly.

	<i>Dynamic Models</i>	<i>Structural Overview Models</i>	<i>Entity Descriptors</i>
<i>System Specification</i>	Scenarios	Goals	Functionalities actions & percepts
<i>Architectural Design</i>	(interaction diagrams) Interaction Protocols	(coupling diagram) (agent acquaintance) System Overview	Agents Messages
<i>Detailed Design</i>	Process Diagrams	Agent Overview Capability Overview	Capabilities Plans, Data, Events

Table (4-1). The Major Models of Prometheus

Prometheus, as a methodology, is intended to be able to support design of BDI systems, although it is not limited to such: all but the lowest level of design, leading into code, can be used equally well for non-BDI systems. However, the lowest level needs to be modified to accommodate the particular style of implementation platform being targeted. For instance if building JADE agents, the lowest level would specify *behaviors* rather than plans, and there would be some changes in details.

### 4.3.2 Overview of the Prometheus Methodology

We now turn to considering the overall structure of the Prometheus methodology. The sections below go through each of Prometheus' three phases in more detail and will discuss the notations used by the methodology as well as some specific techniques.

The Prometheus methodology consists of three phases: [20]

- 1- System specifications: where the system is specified using goals and scenarios; the system's interface to its environment is described in terms of actions, percepts and external data; and functionalities are defined.
- 2- Architectural design: where agent types are identified; the system's overall structure is captured in a system overview diagram; and scenarios are developed into interaction protocols.
- 3- Detailed design: where the details of each agent's internals are developed and defined in terms of capabilities, data, events and plans; process diagrams are used as a stepping-stone between interaction protocols and plans.

Each of these phases include models that focus on the *dynamics* of the system, (graphical) models that focus on the structure of the system or its components, and textual descriptor forms that provide the details for individual entities. In the following sections, we briefly describe the processes and models associated with each of the three phases. Due to space limitations and the desire to describe all of the methodology, this chapter cannot do justice to Prometheus. In particular, we cannot describe a running example in detail, and the detailed techniques, that are *how* particular steps in the process are performed, are not described. For more information on Prometheus, including a complete description, see [20].

### 4.3.2.1 System Specification

The system specification is the first phase of Prometheus. Its main purpose is building the system's environment model, identifying the goals and functionalities of the system, and describing key use case scenarios.

System specification begins with a rough idea of the system, which may be simply a few paragraphs of rough description, and proceeds to define the requirements of the system in terms of: [20]

- 1- The *goals* of the system
- 2- *Use case scenarios*
- 3- *Functionalities*, and
- 4- The interface of the system to its environment, defined in terms of *actions* and *percepts*.

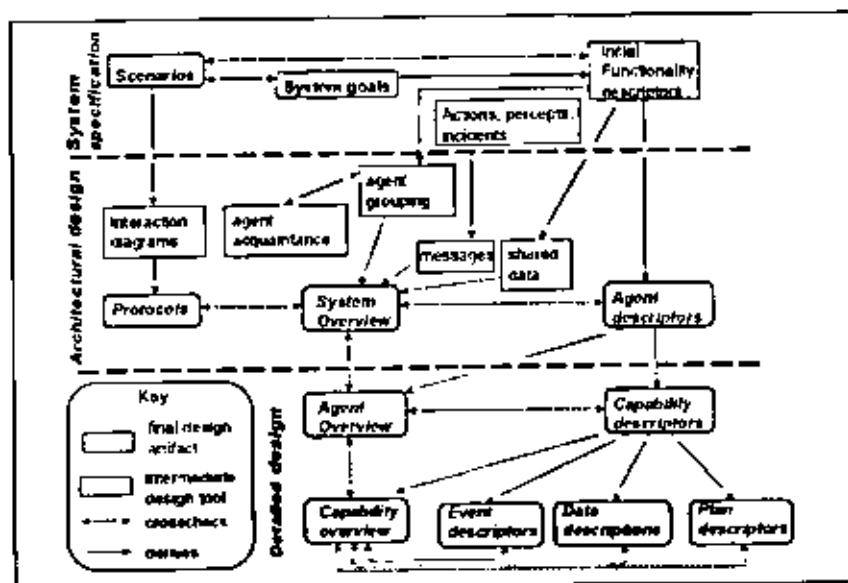


Figure (4-3) - The phases of the Prometheus methodology <sup>1</sup>

<sup>1</sup> K. H. Dam, "Evaluating and Comparing Agent-Oriented Software Engineering Methodologies", Master of Applied Science in Information Technology, RMIT University, Australia, June 27, 2003.

We would like to stress that these are not considered in sequence. Rather, work on one of these will lead to further ideas on another. For example, the goals of the system are a natural starting point for developing use case scenarios. Conversely, developing the details of use case scenarios often suggests additional sub-goals that need to be considered. Thus, system specification is an iterative process. Since agents are proactive, and have goals, it is natural to consider using goals to describe requirements.

The process for capturing the goals of the system begins by capturing an initial set of goals from the high-level system description. For example, from a description such as “*we wish to develop a group scheduling system that allows users to book meetings with other users . . .*” we can extract goals such as scheduling meetings, re-scheduling meetings, and managing users’ calendars. These initial goals are then developed into a more complete set of goals by considering each goal and asking *how* that goal could be achieved, this identifies additional sub-goals. For example, by asking how meetings can be scheduled we may realize that we need to find a common free time. This is a new sub-goal of the top-level goal of scheduling a meeting. In addition to identifying additional goals, the set of goals is also revised as common sub-goals are identified. For example, both scheduling and re-scheduling meetings may have a sub-goal of determining a common free time for the participants [20].

The goals are represented using a *goal diagram*. This depicts goals as ovals and shows the sub-goal relationships with arrows from parent goals to sub-goals. In fact, there are also other reasons for considering goal-oriented requirements.

As we revise the groupings of goals, we are attempting to identify what we term “functionalities” – coherent chunks of behavior, which will be provided by the system.



Functionality encompasses a number of related goals, percepts that are relevant to it, actions that it performs, and data that it uses. Functionalities can be thought of as “abilities” that the system needs to have in order to meet its design objectives, indeed, often functionalities of the system end up as *capabilities* of agents in the system.

An initial set of functionalities is identified by considering groupings of goals. The functionalities are often then revised as a result of considering the agent types (done as part of architectural design).

Functionalities are described using *descriptors*. These are just textual forms that capture necessary information. In addition to a (brief) natural language description, the descriptor form for functionality includes the goals that are related to it, the actions that it may perform, and “triggers” – situations that will trigger some response from the functionality. Triggers may include percepts, but more generally will include events as well. Finally, the descriptor form also includes notes on the information used and produced by the functionality [20].

The third aspect of system specification is *use case scenarios*. Use case scenarios are a detailed description of one particular example sequence of events associated with achieving a particular goal, or with responding to a particular event.

Scenarios are described using a name, description, and a triggering event. However, the core of the scenario is a sequence of steps. Each step consists of the functionality that performs that step, the name of the step, its type (one of ACTION, PERCEPT, GOAL, SCENARIO or OTHER) and, optionally, the information used and produced by that step. In addition, scenarios often briefly indicate variations. For example, when scheduling a meeting a scenario may include a step that selects a preferred time from a list of possible

times. A variation of this scenario might be where there is only a single time when all participants are available. In this case, the selection step is omitted. Finally, the environment within which the agent system will be situated is defined. This is done by describing the percepts available to the system, the actions that it will be able to perform, as well as any external data that is available and any external bodies of code. When specifying percepts we also consider *percept processing*. Often percepts will need to be processed in some way to extract useful information. For example, raw image data indicating that a fire exists at a certain location may not be significant if the agent is already aware of this fire. When agents are situated in physical environments then percept processing can be quite involved. For example, extracting features of interest from camera images. Similarly, actions may also be complex and require design.

#### 4.3.2.2 Architectural Design

In the architectural design phase, the focus is on [20]:

1. Deciding on the *agent types* in the system; where agent types are identified by grouping functionalities based on considerations of coupling; and these are explored using a coupling diagram and an agent acquaintance diagram. Once a grouping is chosen, the resulting agents are described using agent descriptors.
2. Describing the interactions between agents using *interaction diagrams* and *interaction protocols*; where interaction diagrams are derived from use case scenarios; and these are then revised and generalized to produce interaction protocols.

3. Designing the overall system structure: where the overall structure of the agent system is defined and documented using a system overview diagram. This diagram captures the agent types in the system, the boundaries of the system and its interfaces in terms of actions and percepts, but also in terms of data and code that is external to the system. Deciding on the agent types that will exist in the system is perhaps the most important decision that is made in this phase. Ideally, each agent type should be cohesive and coupling between agents should be low.

In Prometheus, an agent type is formed by combining one or more functionalities. Different groupings of functionalities give alternative designs that are evaluated based on the cohesiveness of the agent types, and the degree of coupling between agents.

Some reasons that might need to be considered when grouping agents include: [20]

1. If two functionalities are clearly related then it might make sense to group them together in the same agent type. Conversely, if two functionalities are clearly not related then they should perhaps not be grouped in the same agent type.
2. If two functionalities need the same data then they should perhaps be grouped together.

A useful tool for suggesting groupings of functionalities is the *data coupling diagram*. This depicts each functionality (as a rectangle) and each data repository (as a data symbol) showing where functionalities read and write data. It is often fairly easy to extract some constraints on the design by visually examining a data-coupling diagram.

The process of deriving agent types by grouping functionalities, with the aid of a data-coupling diagram, can often suggest possible changes to the functionalities. For

example, suppose that the design includes two functionalities, which are unrelated and we would like to put them in two different agent types. However, the two functionalities both read a particular data source. We could change one of the functionalities so that rather than read the data source directly, it sends a message to another agent requesting the information.

The result of this process is a number of possible designs, each design consisting of a grouping of functionalities into agent types. We now need to select a design. One technique that is useful in comparing the coupling of different alternatives is the use of agent acquaintance diagrams. An agent acquaintance diagram shows the agent types and the communication pathways between them. Agent acquaintance diagrams provide a convenient visualization of the coupling between the agent types: the higher the link density, the higher the coupling.

Once agent types have been decided upon, they are documented using an agent descriptor. In addition to capturing the interface of the agent, what goals it achieves, what functionalities were combined to form it, and what protocols the agent is involved with; the descriptor prompts the designer to think about *lifecycle* issues: when are instances of this agent type created? When are they destroyed? What needs to be done when agents are created/destroyed? The next step in the architectural design is to work on the interactions between agents. These are developed using interaction diagrams and interaction protocols. Specifically, the notations used are a simplified variant of UML, sequence diagrams for interaction diagrams, and AUML, (the revised version) for interaction protocols. Interaction diagrams are derived from use case scenarios using a mechanical process (although not completely mechanical) [26].

In essence, if step  $N$  is performed by an agent  $A$ , and this is followed by step  $N + 1$  performed by a different agent  $B$ , then a message needs to be sent from  $A$  to  $B$ . Like use case scenarios, interaction diagrams show example interactions rather than all possible interactions. In order to define plans that will handle all necessary interactions we use interaction protocols to capture all possible sequences of messages. Often an interaction protocol will combine a number of interaction diagrams. For example, if there are three interaction diagrams corresponding to different cases of scheduling a meeting then there will be an interaction protocol that covers all cases and, which subsumes the interaction diagrams. When looking at the use case scenarios we also consider the documented variations of these scenarios.

Another useful technique for developing interaction protocols is to consider each point in the interaction sequence and ask, "What else could happen here?" If the interaction diagram shows an example sequence, where a request for possible meeting times is replied to with a number of possible times, then an alternative possibility is that there will not be any meeting times available.

The interaction protocols are accompanied with descriptors for both the protocols and for the messages. These descriptors capture additional information such as the information carried by a message.

Finally, the overall architecture of the system is captured using a system overview diagram. The system overview diagram is one of the most important design artifacts produced in Prometheus and is often a good starting point when trying to understand the structure of a system [26].

The system overview diagram shows agents, percepts, actions, messages, and external data as nodes.

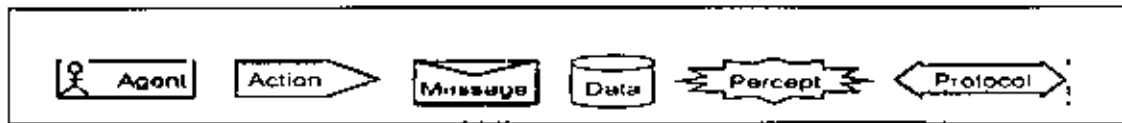


Figure (4-4) - Notation used in System Overview Diagram<sup>1</sup>

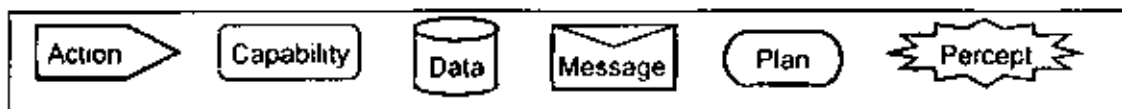


Figure (4-5) - Notation used in Agent Overview Diagrams<sup>1</sup>

Each of these node types has its own distinct visual depiction (see Figure 4-4). Directed arrows between nodes indicate messages being sent and received by agents, actions being performed by agents, percepts being received by agents, and data being read and written by agents.

#### 4.3.2.3 Detailed Design

Detailed design consists of: [20]

1. Developing the internals of agents, in terms of capabilities (and, in some cases directly in terms of events, plans and data). This is done using agent overview diagrams and capability descriptors.
2. Develop process diagrams from interaction protocols.
3. Develop the details of capabilities in terms of other capabilities as well as events, plans and data.

<sup>1</sup> L. Padgham, M. Winikoff, "Developing Intelligent Agent Systems: A Practical Guide", Research paper, RMIT University, Melbourne, Australia, 2004. ISBN 0-470-86120-7.

This is done using capability overview diagrams and various descriptors. A key focus is developing plan sets to achieve goals and ensuring appropriate coverage.

Capabilities are a structuring mechanism similar to modules. A capability can contain plans, data, and events. It can also contain other capabilities allowing for a hierarchical structure. In identifying the capabilities that each agent type contains, one usually starts by considering a capability for each functionality that was grouped in the agent type. This initial detailed design is then refined by merging capabilities that are similar and small, splitting capabilities that are too large and adding capabilities that correspond to common “library” code.

The structure of each agent is depicted by an agent overview diagram. This is similar to the system overview diagram except that it does not contain agent nodes and does not (usually) contain protocol nodes. However, the agent overview diagram does (usually) contain capability nodes and (sometimes) plan nodes. The node types found in agent overview diagrams are shown in Figure (4-5). During the architectural design phase, the system’s dynamics were described using interaction protocols.

These are global in that they depict the interaction between the agents from a “bird’s eye-view”. In the detailed design phase we develop process diagrams based on the interaction protocols. The process diagrams depict *local* views for each agent. Typically, each interaction protocol will have multiple process diagrams corresponding to the viewpoints of different agents. The notation that we use for process diagrams is an extension of UML activity diagrams, for more details see [20, chapter 8]. The design of each agent is, usually, in terms of capabilities. These capabilities are then refined in turn. Eventually the design of how each agent achieves its

goals is expressed in terms of plans, events and data. At this point, the design process needs to make certain assumptions about the implementation platform. Prometheus methodology differs from existing methodologies in that it: [21]

1. Supports the development of *intelligent* agents, which use goals, beliefs, plans, and events. By contrast, many other methodologies treat agents as “simple software processes that interact with each other to meet an overall system goal”.
2. Provides “start-to-end” support (from specification to detailed design and implementation) and a *detailed process*, along with design artifacts constructed and steps for deriving artifacts.
3. Evolved out of practical industrial and teaching experience, and has been used by both industrial practitioners and by undergraduate students. By contrast, many other methodologies have been used only by their creators and often only on small (and unimplemented) examples.
4. Provides hierarchical structuring mechanisms, which allow design to be performed at multiple levels of abstraction. Such mechanisms are crucial to the practicality of the methodology on large designs.
5. Uses an iterative process over software engineering phases rather than a linear “waterfall” model. Although the phases are described in a sequential fashion in this research, the intention is *not* to perform them purely in sequence.
6. Provides (automatable) crosschecking of design artifacts.

Of the properties above, perhaps the most contentious is the first: many existing methodologies intentionally do not support intelligent agents; rather, they aim for



generality and treat agents as black boxes. It is believed that in this case, generality needs to be sacrificed in favor of usefulness. By specifically supporting the development of BDI-like agents, we are able to provide detailed processes and deliverables, which are useful to developers. Of course, this makes Prometheus less useful to those developing non-BDI like agents. However, it must be noted that the initial stages of the methodology *are* appropriate for the design of any kind of multi-agent system. Although none of these properties is unique in isolation, their combination is unique. These properties are all essential for a practical methodology that is usable by non-experts and accordingly the design of Prometheus was guided by these properties. Although Prometheus' contribution is the combination of these properties, this combination was achieved through careful design of the methodology. It is not possible to easily construct a methodology, which has the above properties by combining methodologies that have some of them. For example, given a methodology that provides automated support but does not support intelligent agents and another methodology that supports intelligent agents but not provide automated cross-checking; it is not at all obvious how a hybrid methodology could be created that supports both feature [22].

### **4.3.3 Tool Support**

One consequence of the iterative nature is that the design is often modified. As the design becomes larger, it becomes more difficult to ensure that the consequences of each change are propagated and that the design remains consistent.

Perhaps the simplest example of introduced inconsistency is renaming an entity and failing to rename it everywhere it is mentioned. Other forms of inconsistency that can be

easily introduced when making changes to a design include adding a message to an agent in the system overview diagram, but failing to ensure that the message appears in the agent overview diagram of that agent type.

The Prometheus Design Tool (see Figure (4-6)) allows users to create and modify Prometheus designs. It ensures that certain inconsistencies cannot be introduced and provides cross checking that detects other forms of inconsistency. The tool can also export individual design diagrams as well as generate a report that contains the complete design.

For more details on tool support for Prometheus see [23]. Another tool that supports the Prometheus methodology is the JACK Development Environment (JDE), which provides a design tool that allows Prometheus-style overview diagrams to be drawn. The JDE can then generate skeleton code from these diagrams. This facility has proven quite useful.

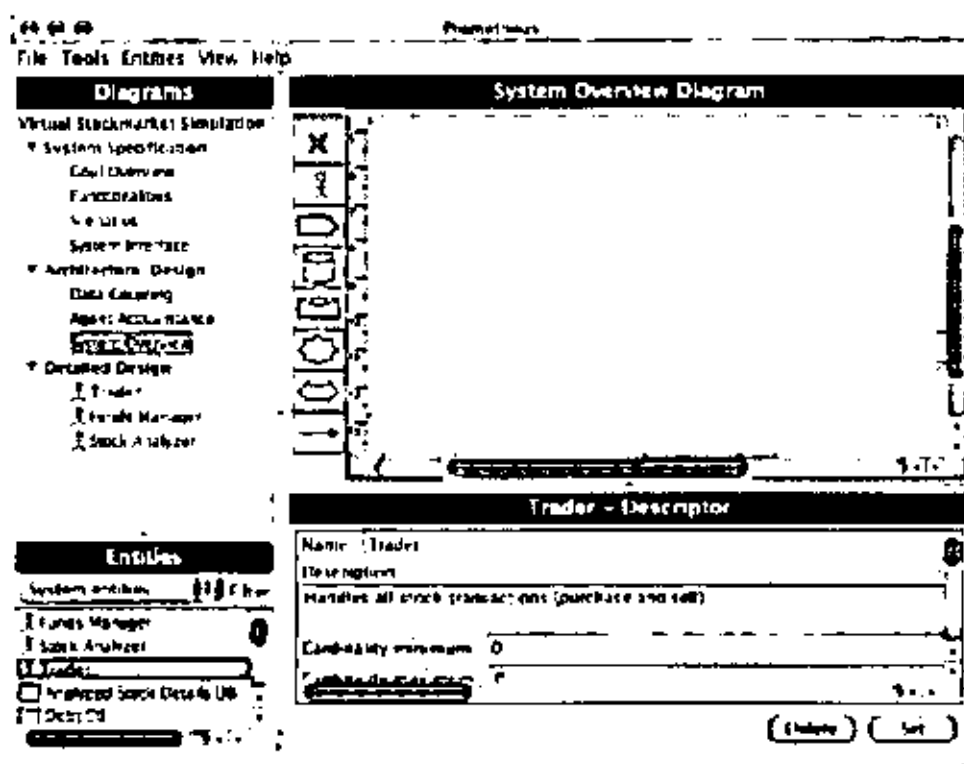


Figure (4-6) - The Prometheus Design Tool (PDI)<sup>1</sup>

<sup>1</sup> L. Padgham, M. Winikoff. "Prometheus: A pragmatic methodology for engineering intelligent agents". Research paper, pp 97-108, Seattle, 2002.

## Chapter Five

### Case study

We would present the previous methodology procedures in a small library system performing the following functions:

#### 1. Book functions

- i. Allow for checkout of books, providing a return date to the customer.
- ii. Allow for return of books.
- iii. Allow for reservation of unavailable books.

#### 2. Email functions

- i. Allow for notification of new emails.
- ii. Allow Timed scanning of a library inbox.
- iii. Allow Notification of message arrivals based on level of importance.
- iv. Allow automatic routing of incoming messages to other e-mail users.
- v. Allow Automatic replies to messages based on subject or sender.
- vi. Allow Automatic sorting of unread messages into separate folders based on subject, sender, or level of importance.

### 5.1 System Specification

In order to design such system, goals, sub-goals, appropriate scenarios, and rules are defined as following:

### **5.1.1 Goals of the System**

The goals are identified from the specification based on the functionality required by the system. Here the requirements are converted into goals of the system:-

- 1, Checkout books
2. Provide return date
- 3, Return books
- 4, Reserve unavailable books
5. Give notification of new emails
- 6, Give notification for arrival of subjects
- 7, Give notification for overdue books
- 8, Give notification of arrival of reserved books

Now we have to ask the question 'how?' for each goal and find out sub-goals.

### **5.1.2 Sub-goals of the System**

#### **1. Checkout books**

- i. Record book code to the user ID checked out list.
- ii. Provide return date.

#### **2, Return books**

- i. Remove book code from the user ID.

#### **3. Reserve unavailable books**

- i. Record book code as reserved for user ID.
- ii. Show the current due date for the book.

**4. Give Notification of new emails**

- i. Timed scanning of a library inbox.
- ii. Notification of message arrivals based on level of importance.
- iii. Automatic routing of incoming messages to other e-mail users.
- iv. Automatic replies to messages based on subject or sender.
- v. Automatic sorting of unread messages into separate folders based on subject, sender, or level of importance.

**5. Give notification for overdue books**

- i. Access book record at the start of the day.
- ii. Send email for overdue books

**6. Give notification of arrival of reserved books**

- i. Access the reserved list for user.
- ii. Send email notification.

Now we will create the goal overview diagram. Select the Goal overview from system specification and add the goal to the diagram. Add the sub-goals and connect to the main goals using the edge. Draw a diagram based on the goals and sub-goals we have already identified. The resultant goal overview diagram will be as shown in Figure (5-1).

**5.1.3 Scenarios of the System**

Depending on the system, we can identify five different scenarios.

1. When the user comes to checkout the book.
2. When the user returns the book.

3. When a book becomes overdue.
4. When the user asks to reserve a book.
5. When the reserved book arrives.
6. When the Agent handle each new message.

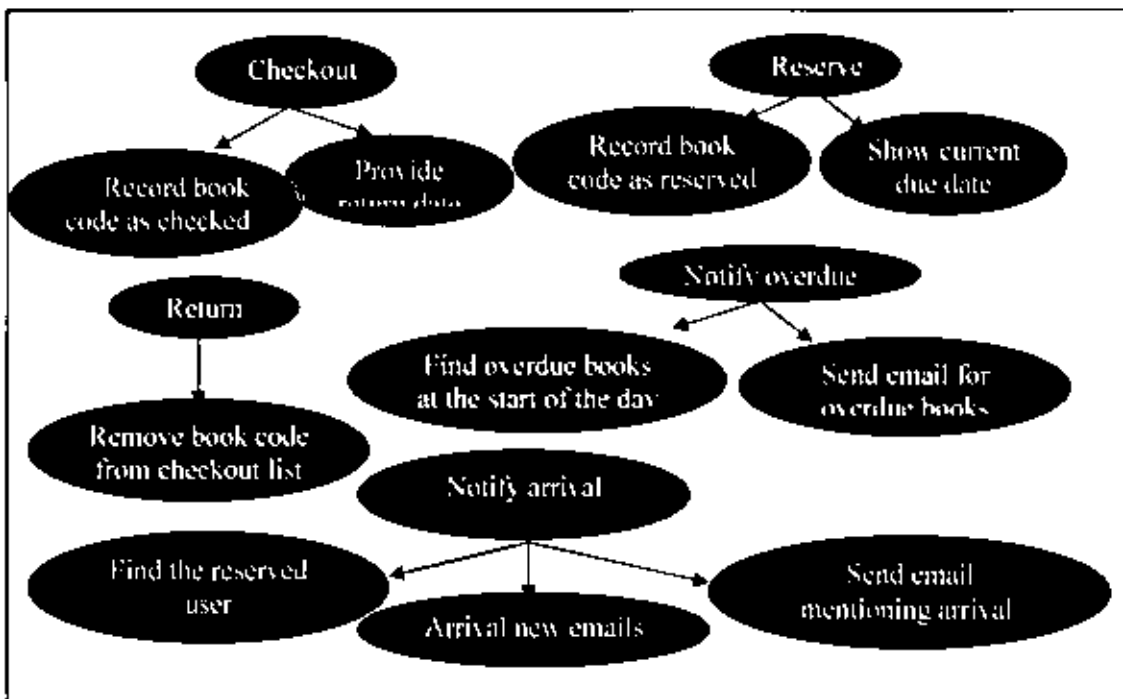


Figure (5-1) - Goal Overview Diagram

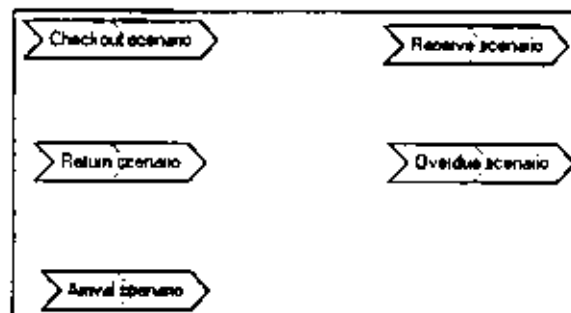


Figure (5-2) - Scenarios Diagram.

We can identify the following steps for each of the following scenarios:

### **1. Checkout Scenario**

- i. Request for checkout
- ii. Provide return date.
- iii. Record book code as checked out.
- iv. Provide book

### **2. Return Scenario**

- i. Book returned.
- ii. Remove book code from checkout list

### **3. Reserve Scenario**

- i. Request for reservation.
- ii. Record book code as reserved.
- iii. Show current due date.
- iv. Provide current due date

### **4. Arrival Scenario**

- i. Reserved book arrives.
- ii. Find the reserved user.
- iii. Send arrival email.
- iv. Arrival new emails.

### **5. Overdue Scenario**

- i. Start of the day.
- ii. Find overdue books at the start of the day.
- iii. Send overdue email

Now based on the different functionalities of the system we group it into different roles.



1. Checkout books
2. Return books
3. Overdue books
4. Reserve books
5. Send Arrival Notification
6. Reviews and processes the messages for library.

Typically, e-mail agents process messages in the user's inbox. Users can set up rules that tell the agent how to handle each new message. These rules can tell the e-mail agent to check various parts of the incoming message and then take a specific action. Now link the percepts, goals and actions to the roles as shown in the diagram *Figure (5-3)*.

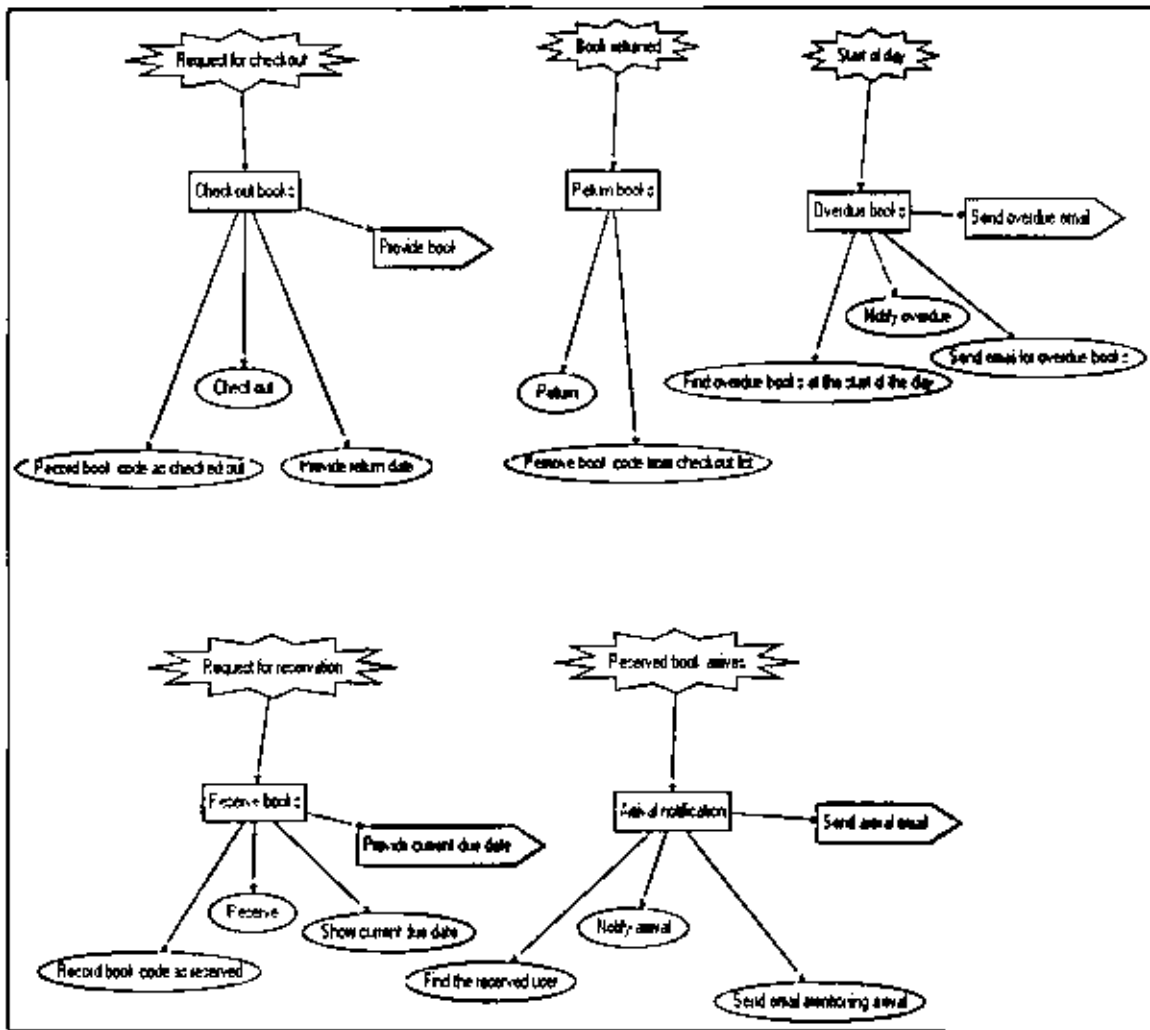


Figure (5-3) - System Roles Diagram

## 5.2 Architectural Design

Now we need to identify what type of data need to be stored in the system as beliefs. We can see that we need to keep the information of all books that has been checked out and also the books that has been reserved. The checkout books role and return books role update the checkout belief and the overdue books role uses this to send overdue email. The reserve books role updates the reserve belief and the arrival notification uses it.

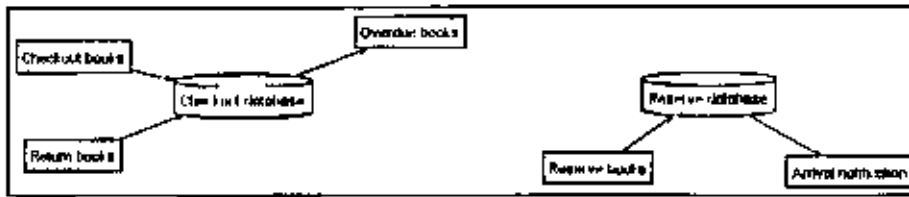


Figure (5-4) - Data Coupling Diagram

Now we group the roles and identify 4 agents to carry out these roles in the system. The agents identified are:

1. Checkout Agent.
2. Reservation Agent.
3. Overdue Agent.
4. Intelligent Email Agent.

The agents interact with each other. This is shown in the agent acquaintance diagram. The links between the agents are automatically established when a message is sent from one agent to another as shown in the system overview diagram.

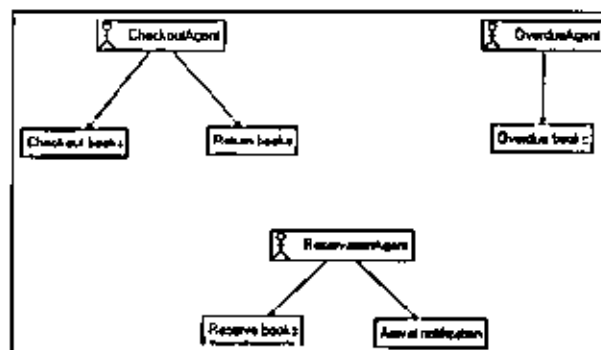


Figure (5-5). Agent Role Coupling Diagram

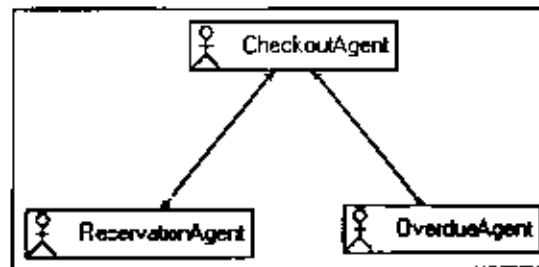


Figure (5-6). Agent Acquaintance Diagram

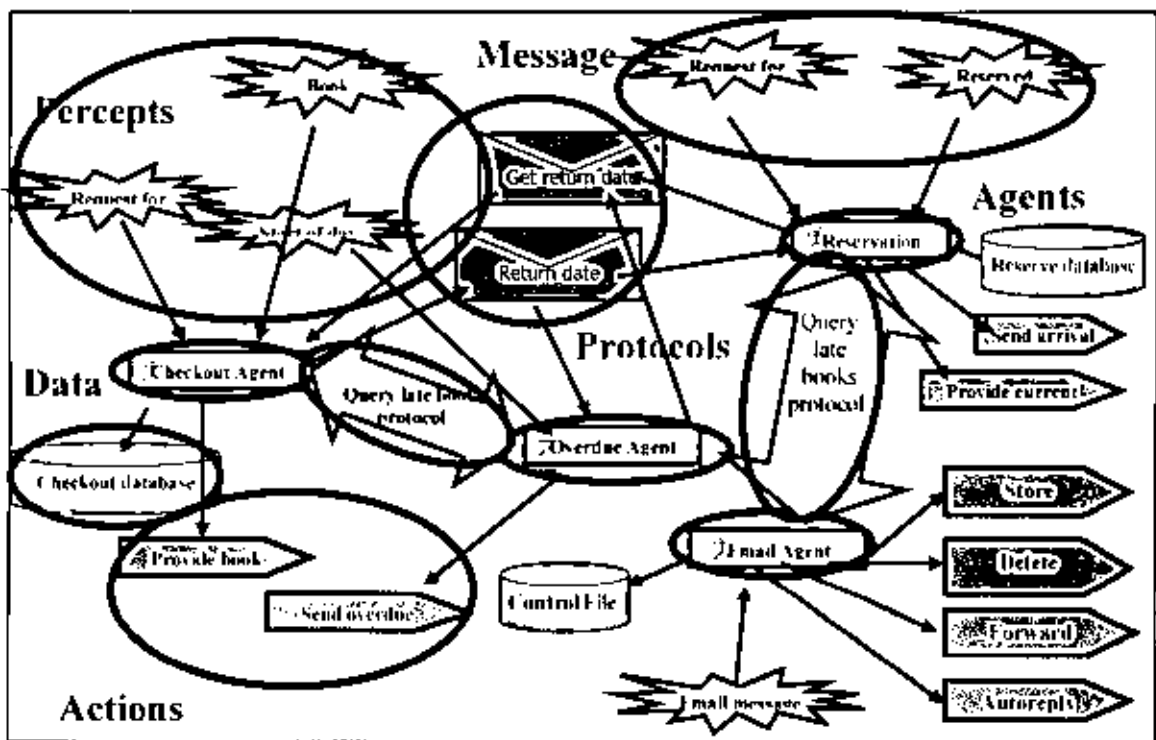


Figure (5-7). System Overview Diagram

### 5.3 Detailed Design

We will go into details of agents and their capabilities.

### 5.3.1 Checkout Agent

For the checkout agent we can identify three capabilities:

1. Checkout Capability
2. Return Capability
3. Get Return Date Capability

The agent overview diagram of the checkout agent is as shown in the Figure.

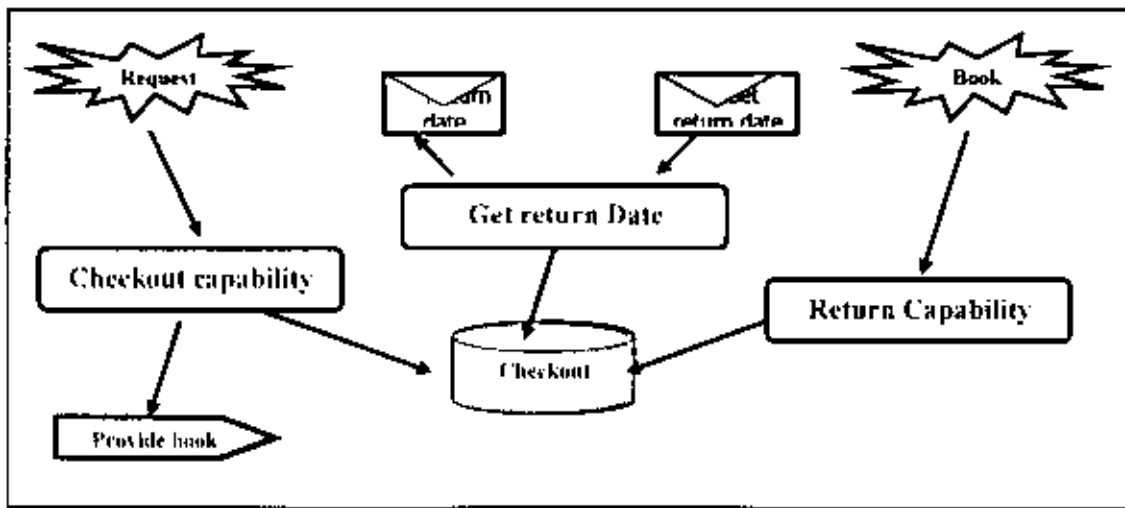


Figure (5-8). Checkout Capability

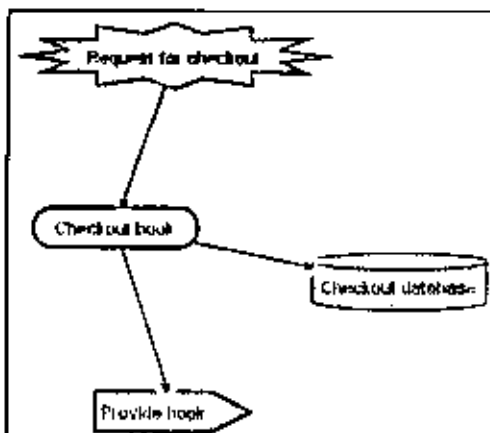


Figure (5-9). Return Capability

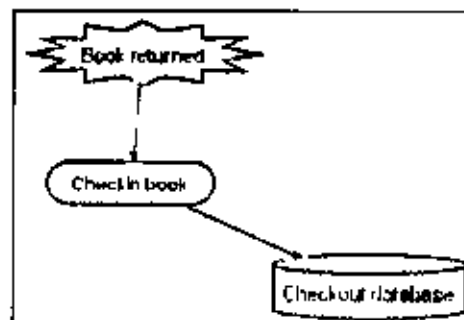


Figure (5-10).1 Get Return Date Capability

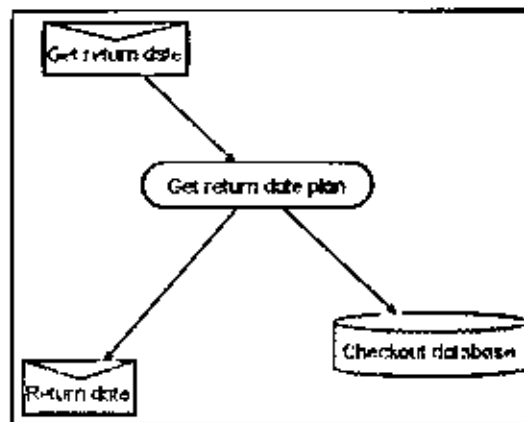


Figure (5-11).2 Get Return Date Capability

### 5.3.2 Reservation Agent

For the reservation agent, we can identify two capabilities:

1. Reservation Capability
2. Arrival Notification Capability

The agent overview diagram of the reservation agent is as shown in the Figure.

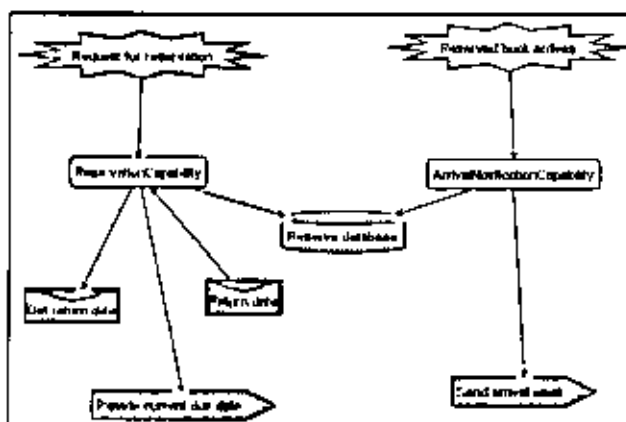


Figure (5-12). Reservation Capability

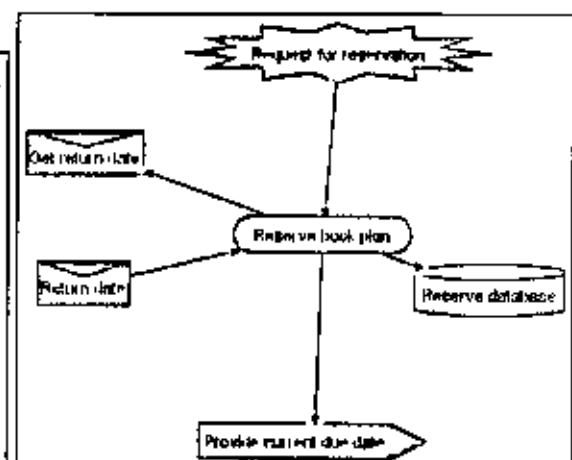


Figure (5-13). Arrival Notification Capability

The agent overview diagram of the overdue agent is as shown in the Figure (5-14).

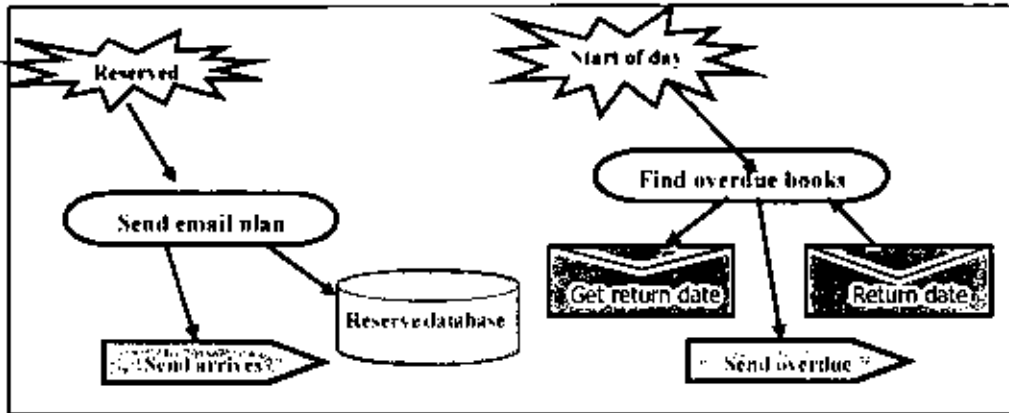


Figure (5-14). Overdue Agent

### 5.3.4 Library Email Agent

In this section one selected function of the library system is programmed. This function is related to the library email agent. This is because Electronic mail has become one of the main communication tools between people around the world regardless of their physical distance.

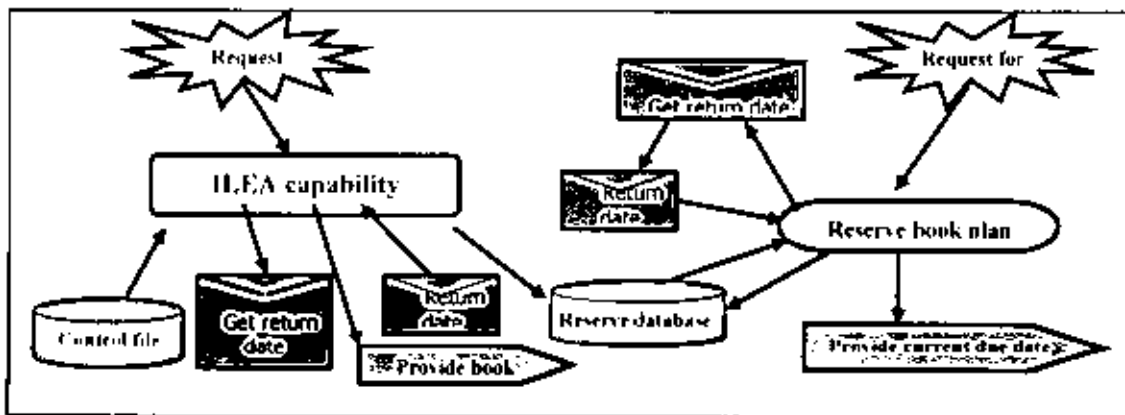


Figure (5-15) Library Email Agent

#### 5.3.4.1 Features of (ILEA)

Before going further into the discussions on the final designs and algorithms of the (ILEA), this section will first illustrate the final list of features, with a brief description, that have been designed. The intelligent agent will be capable of:

1. **Timed scanning of library inbox**-Users can start the program and allow it to run unattended. It scans the inbox for new messages every  $N$  minutes.

Users can also set configuration values that start the program as an icon on the task bar or as a dialog box.

2. **Allowing the user to add, edit and delete their preferred settings.**

The agent will be implemented with graphical user interfaces to allow the users to input their preferred settings, as add, delete actions. These setting include the expected keywords within the subject and contents of the emails and also the expected senders email address.

3. **Sorting the incoming emails into their respective folders according to the users preferred settings with the user of scoring system.**

The intelligent agent will be implemented with a scoring system to effectively compare the emails with the user's settings before distributing the emails into their respective folder.

4. **Identifying the emails level of impotence or urgency according the users choice of sensitivity level for the agent while assessing an email.**

Emails, which are re-directed into particular folder, will be further differentiation into different categories by changing their level of importance. This feature is to allow the user to immediately identify which are the emails that are more important and, hence, being able to look into that email quickly.

5. **Message notification**-Users can establish a rule that causes a dialog box to pop up each time a specific message is received. This notification can be based on sender ID, subject content, or level of importance.



6. Automatic forwarding-Users can create rules that automatically forward messages to other addresses.

Users can also determine whether the original should be kept or discarded.

7. Automatic replies-Users can create rules that generate automated replies to senders based on sender ID, subject content, or level of importance.

8. Automatic message copying or moving-Users can create rules that copy or move incoming messages to other folders in the user's message store. This feature can be used to sort the incoming message by sender ID, subject content, or level of importance.

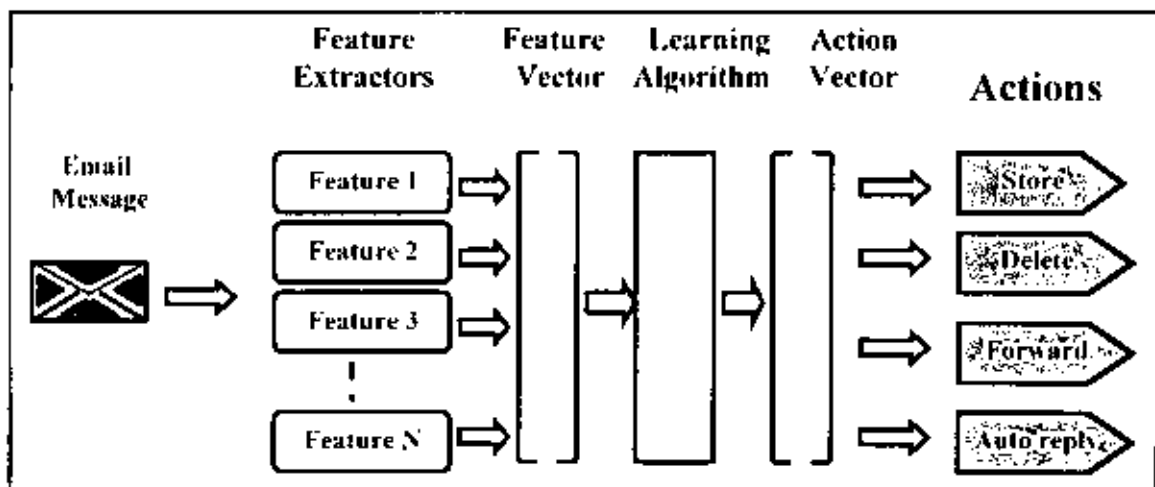


Figure (5-16). Features of intelligent agent

The following block diagram illustrates the overall structure of the email agent system.

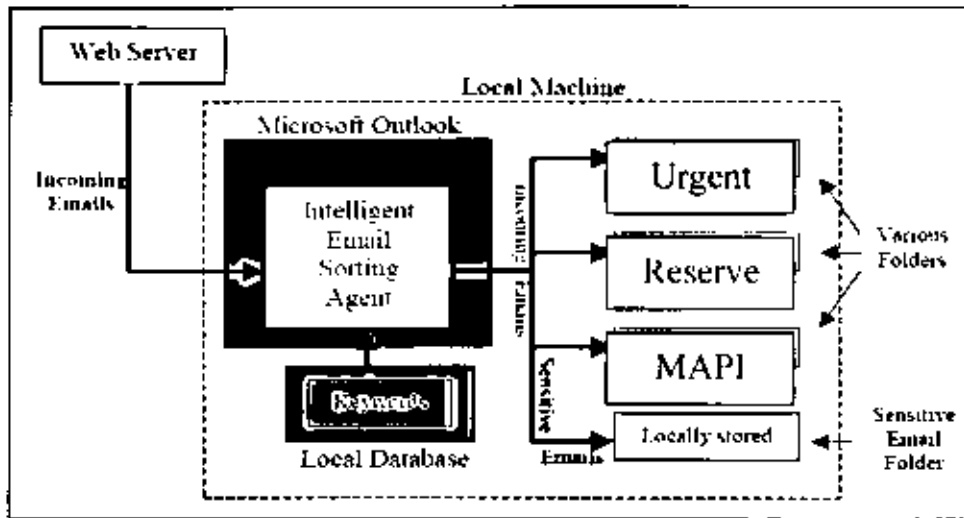


Figure (5-17). Block diagram of the intelligent agent for email information processing.

The Figure (5-17) illustrates a brief model on the various type of information that will be extracted during or after the sorting process.

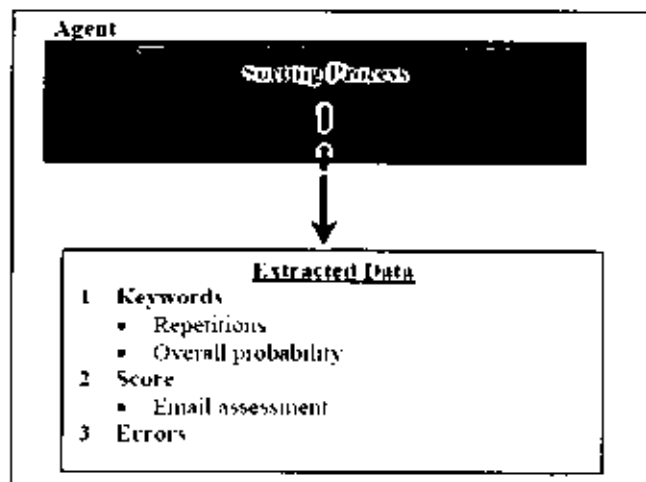


Figure (5-18). Data extraction from sorting process

#### 4.3.4.2 Tools required

The preferred language to be used for this project will be Visual Basic 6.0. Besides this, the tools required are the OLE Messaging library to build intelligent email agent [25] and outlook forms. Visual basic application (VBA) is one of the richest available development environments. VBA allows writing of code that handles many events that take place when working with the outlook information and also the ability to design dialogue boxes that gets information from the user and windows that stay on the screen to provide information to the user. Macros can also be created by VBA, to be added to the outlook tool bar.

The following are object models that will be commonly used when programming with Microsoft outlook.

Object model	Description
Active directory services interface( ADSI)	Interact with the exchange and windows directory.
Collaboration data object (CDO)	Access MAPI properties of outlook items as well as many exchange server properties.
Outlook object model	Create and manipulate outlook items as well as react to application level events.
Other office application object models	Access objects from within outlook
Redemption	Access feature that secure versions of outlook may block and use MAPI features for which the outlook object models has no equivalent.

Table (5-1). Tools required

### 5.4.4.3 Graphical user interface

This section of the chapter will be used to illustrate the designs of the graphical user interfaces.

#### 1. Library E-mail Agent Main Form

The Main form has a set of command button control arrays to handle the user selections (see Figure (5-18)). The first control array covers the top row of buttons.

These buttons handle the main processing steps:

1. **Start Timer** starts the timer to count down to the next message scan.
2. **End Timer** disables the timer.
3. **Setup** calls the configuration dialogue box.
4. **View Log** displays the contents of the Library Email Agent log file.
5. **Refresh** refreshes the list boxes from the control file.
6. **Exit** ends the program.

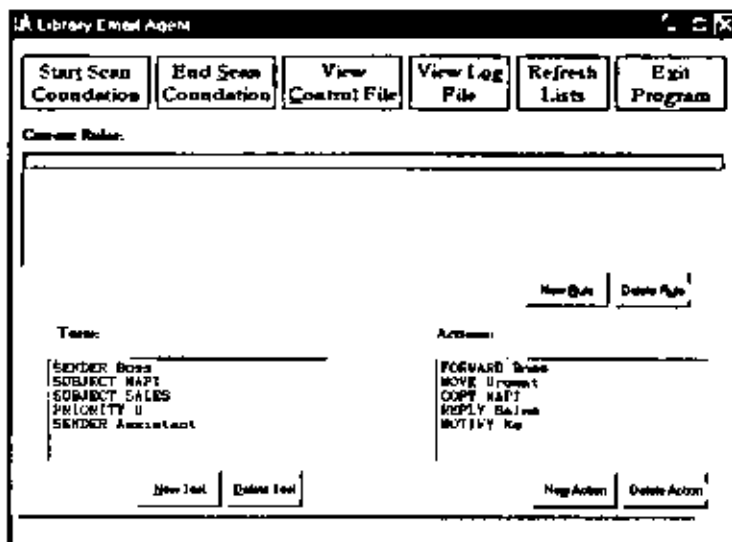


Figure (5-19) Library E-mail Agent Main Form

The second command button control array handles the adding and deleting of tests, actions, and rules. To keep things simple for this project, the system is capable only of adding or deleting rules. Existing rules cannot be edited and saved again. In addition, this program performs only basic input editing. In a production environment, this program should be enhanced to add an improved user interface with additional input checking and recovery.

## 2. The Add Rule Form

The rule form is used to compose new rules for the Library Email Agent (see Figure (5-19)). This form is actually quite simple. It has three list boxes that allow the user to select a test, a compare value, and an action. By combining these three items, the user creates a valid MAPI e-mail agent rule. Once the rule is given a name, it can be saved. All saved rules are acted upon each time Library Email Agent scans the incoming messages.

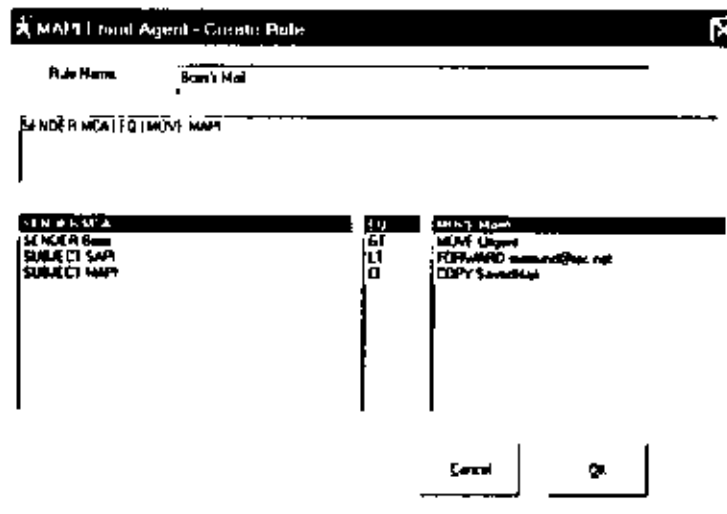


Figure (5-20) Create Rule Form

### 3. The Setup Form

The last form you need to add to the project is the Intelligent Library Email Agent (ILEA) Setup form (see Figure (5-20)). This form allows users to modify the default configuration settings for the (ILEA).

To accomplish this, the Library Email Agent will keep track of rules created by the user. These rules will have three parts: *tests*, *comparisons*, and *actions*.

The test portion of the rule performs a simple scan of the designated portion of the message, searching for requested content. The (ILEA) described in this chapter is capable of inspecting three message parts:

1. **SUBJECT**-Checks the Subject property of the message
2. **SENDER**-Checks the Sender .Name property of the message
3. **PRIORITY**-Checks the Importance property of the message

Editor	notepad.exe
Scan Interval	15
Log File	C:\Documents and Settings\User\My Documents\1
Profile	MCA
Delete Forwarded Messages:	<input checked="" type="checkbox"/>
Delete Replied Messages:	<input checked="" type="checkbox"/>
Minimize On Startup	<input type="checkbox"/>
Start Scan of Upd?	<input checked="" type="checkbox"/>
Use PopUp Dialog on Notify:	<input checked="" type="checkbox"/>
Log Activity to File	<input type="checkbox"/>
Rule Count:	4
Test Count:	7
Action Count:	18/03/2007 09:58:33
Last Updated:	1

Figure (5-21) - The Setup Form

For example, the test SUBJECT MAPI tells the agent to check the message subject for the word "Library." The phrase SENDER Boss tells the agent to check for messages sent to the user from the e-mail ID "boss."

All tests must use a logical condition as part of the processing. The Library Email Agent uses *comparisons* to do this. The program can check for the following four logical conditions:

1. EQ-Equals (SENDER Ali EQ)
2. GT-Greater Than (PRIORITY 0 GT)
3. LT-Less Than (PRIORITY 1 LT)
4. CI-Is Contained In (SUBJECT VB CI)

We will notice that the last value is able to check the selected message part for the occurrence of a word or phrase. Note that all the comparisons are case-insensitive. It is important to note that the LT and GT can be used with character data, too.

The last of the three portions of a rule is the *action*. This is the part of the rule that tells the agent what action to take if the test criteria have been met. The (IL:EA) can perform the following actions on a message:

1. MOVE-Move the message to another folder (MOVE Urgent).
2. COPY- Copy the message to another folder (COPY Archive).
3. FORWARD-Forward the message to another user  
(FORWARDlibrary@yahoo.net).
4. REPLY-Send reply text to the user (REPLY reply.txt)

The agent allows users to determine whether the forwarded and reply messages are retained or removed once the forward/reply is generated.

#### 5.4.4.4 Storing the Rules in a Control File

The Library Email Agent allows users to build tests and actions, and then use them to create rules. All this information is stored in a text file similar to an INI file. This file also contains general control information, such as the scan interval, whether the agent should create a log file, the default log on profiles, and so on. Next listing shows a sample rule file.

Listing shows Sample rule file for the Intelligent Library Email Agent.

```
; *****
; Library Email Agent Control File
; *****

[General]

Editor=notepad.exe

ScanInterval=2

LogFile=mea.log

LogFlag=1

RuleCount=3

ActionCount=4

TestCount=4

Profile=MCA

DeleteForwardFlag=0

NotifyDialog=1

DeleteReplyFlag=0

MinimizeOnStart=0
```



AutoStart=0

LastUpdated=04/3/2007 9:27:30 PM

**[Actions]**

Action0=MOVE MAPI

Action1=MOVE Urgent

Action2=FORWARD library@yahoo.com

Action3=COPY SavedMail

**[Tests]**

Test0=SENDER MCA

Test1=SENDER Boss

Test2=SUBJECT SAPI

Test3=SUBJECT MAPI

**[Rules]**

RuleName0=Boss's Mail

RuleTest0=SENDER Boss

RuleAction0=Move Urgent

RuleCompare0=EQ

RuleName1=Send To ISP

RuleTest1=SENDER MCA

RuleAction1=FORWARD Library@yahoo.com

RuleCompare1=EQ

RuleName2=MAPI Mail

RuleTest2=SUBJECT MAPI

RuleAction2=MOVE MAP1

RuleCompare2=C1

#### 5.4.4.5 Prototype Algorithm

Prototype algorithm is designed with the intention of creating a basic algorithm to process each email by comparing the emails properties with the user specified settings.

Users can start the program and allow it to run unattended. It scans the inbox for new messages every  $N$  minutes. Users can also set configuration values that start the program as an icon on the task bar or as a dialog box.

In addition, can establish a rule that causes a dialog box to pop up each time a specific message is received. This notification can be based on sender ID, subject content, or level of importance.

Users can create rules that automatically forward messages to other addresses. Users can also determine whether the original should be kept or discarded, and can create rules that generate automated replies to senders based on sender ID, subject content, or level of importance, also can create rules that copy or move incoming messages to other folders in the user's message store.

This feature can be used to sort incoming message by sender ID, subject content, or level of importance.

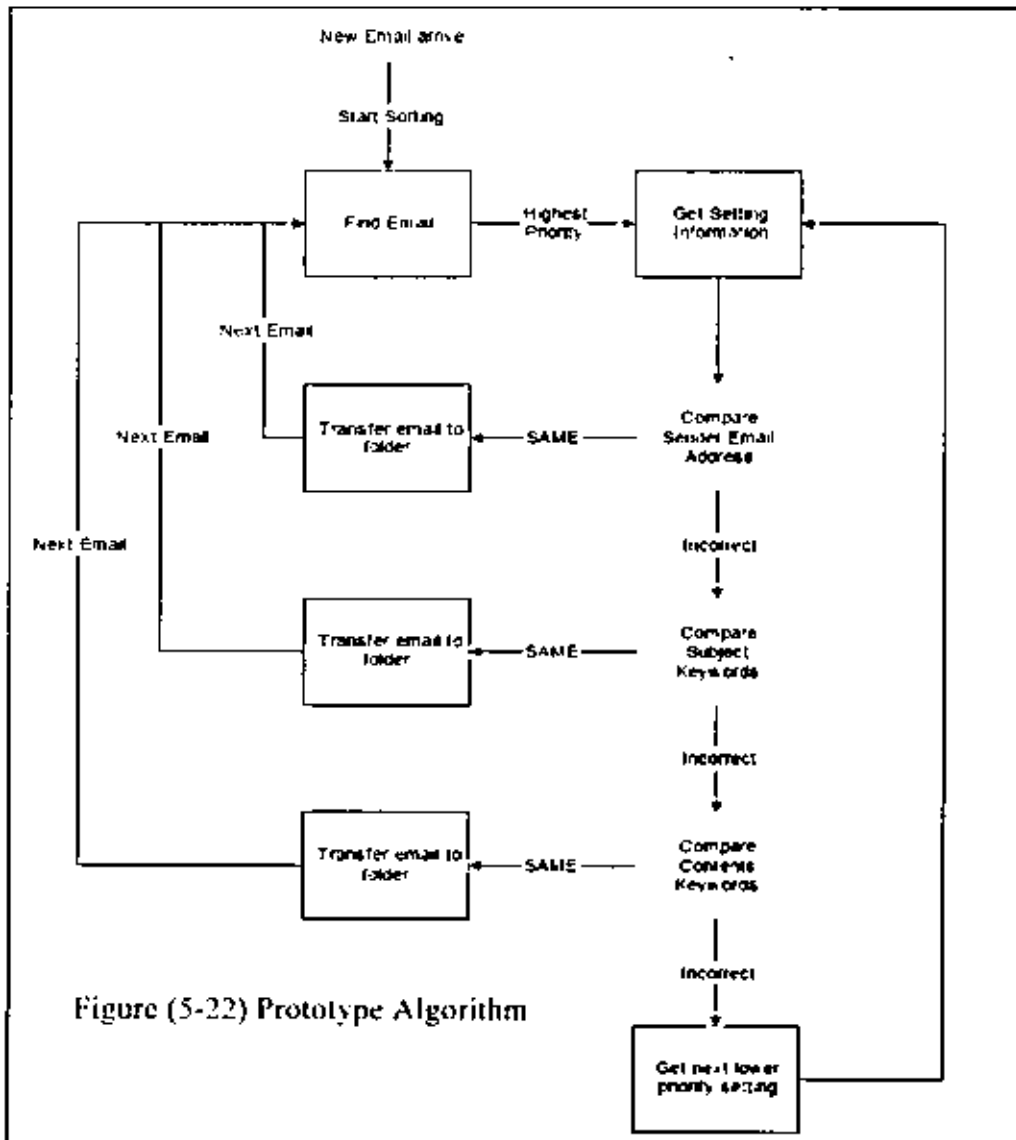


Figure (5-22) Prototype Algorithm

Figure (5-23) sorting  
algorithm with identification  
of forward and replied emails

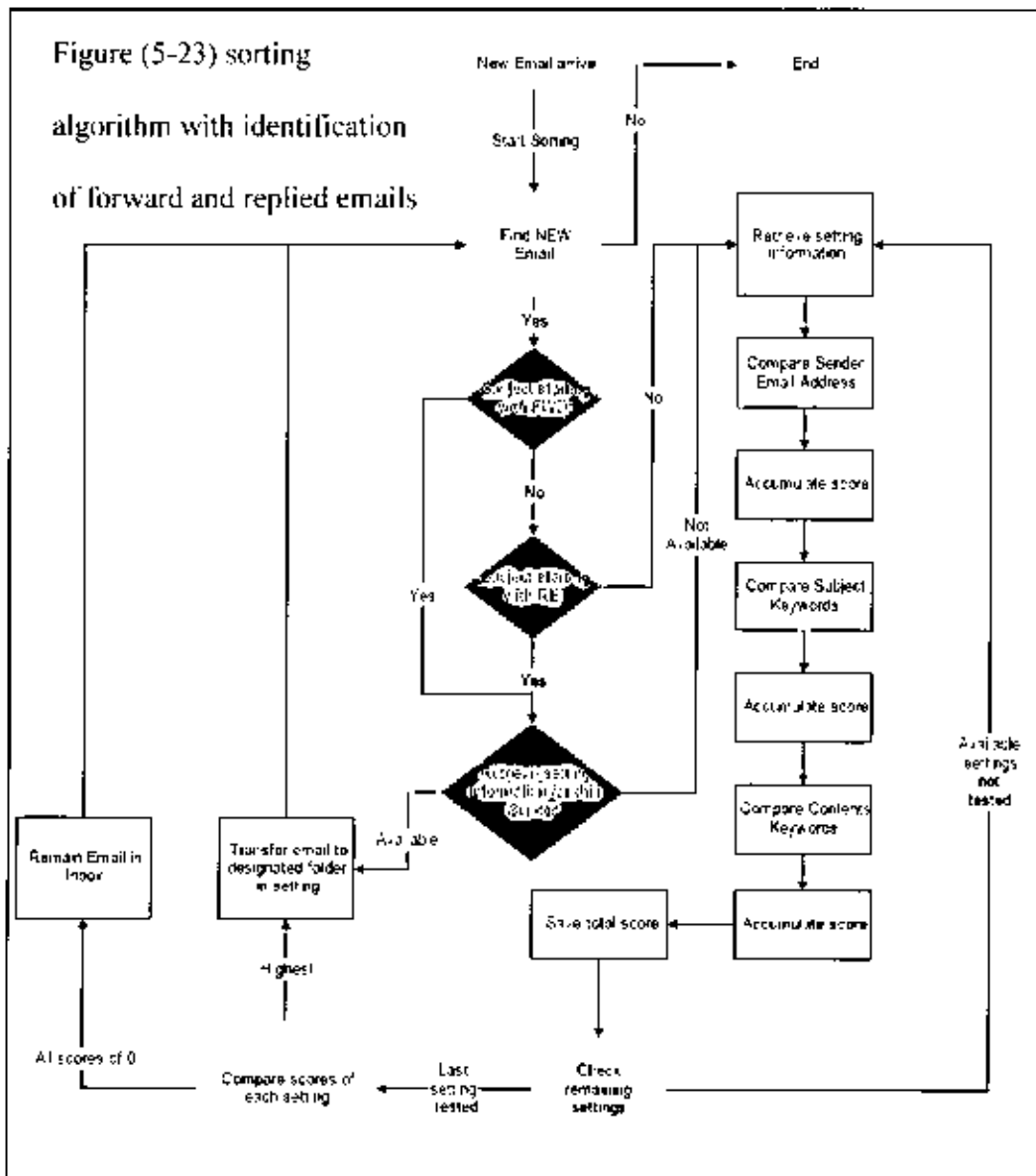
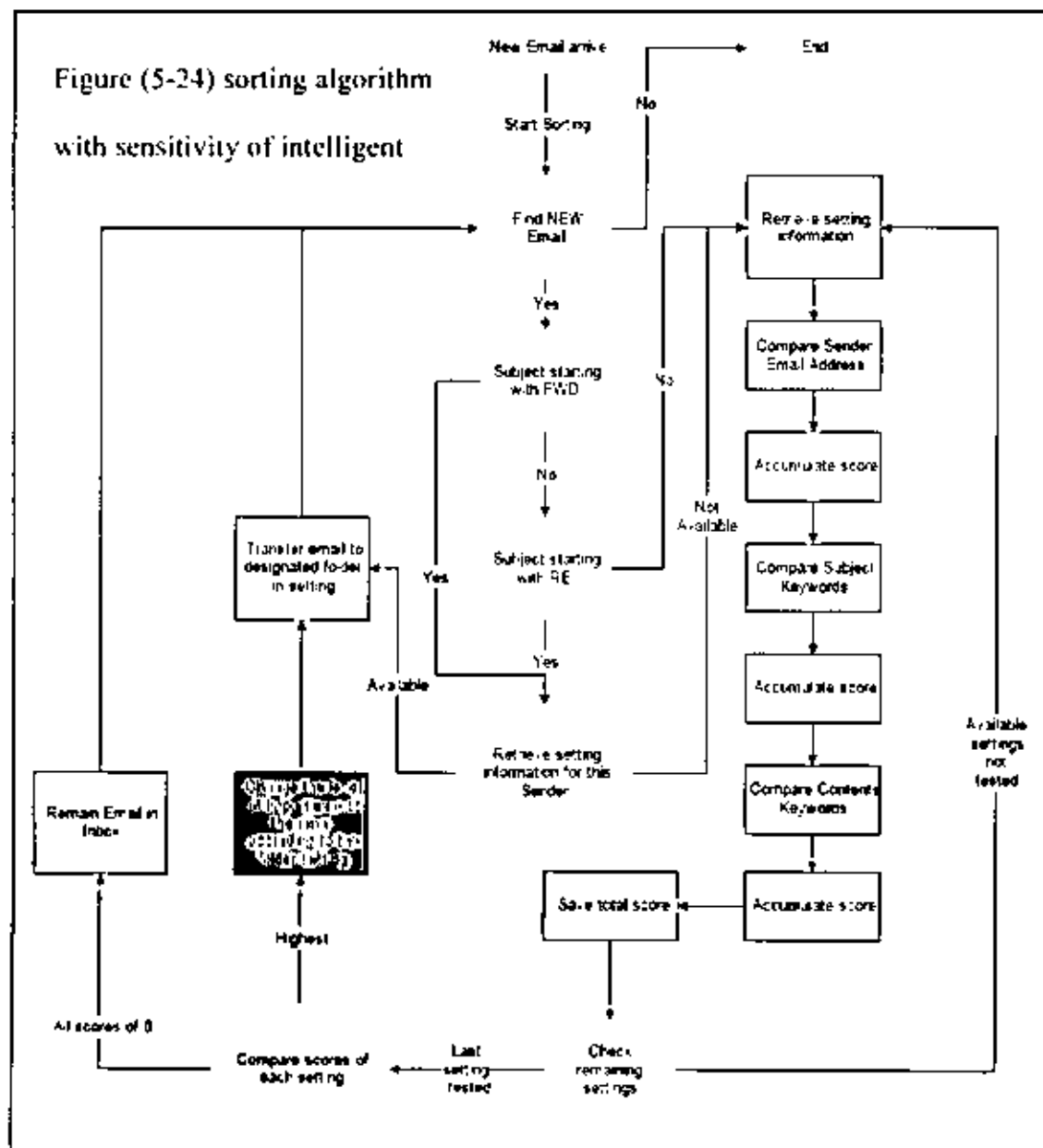


Figure (5-24) sorting algorithm  
with sensitivity of intelligent



## Chapter Six

### Conclusion and Recommendations

#### Summary

Agents are a powerful technology with many significant applications. A key issue in getting the technology into mainstream software development is the development of appropriate methodologies for engineering agent-oriented software [23]. Software agents are used to solve several complex problems. Software engineering processes and tools are vital to support all the development phases of agent-based systems, and guarantee that these systems are built properly. Many of the distributed, complex systems developed in the last decade are based on the intelligent agents technology. Moreover, nowadays, the Internet and web based applications offer new opportunities for building efficient multi-agent systems. Thus, there is a need for a specific software engineering, named agent-oriented, that is more appropriate than other existing software engineering such as the object oriented one. The thesis presents a methodology for agent-oriented analysis and design and the key aspects of agent-based software development, focusing on one of the most known methodologies as the *Prometheus* methodology for developing intelligent agent systems. The methodology has been developed over the last several years in collaboration with Agent Oriented Software.

The first chapter of the thesis started with simple introduction about thesis subject, which deal with agents as a new design-paradigm for software engineering and shows researches into the areas of agent-oriented methodologies. It has illustrated the main objective of this research, which is using methodology from Agent-Oriented

Software Engineering methodologies. It presented appropriate techniques and tools that will enable inexpensive development and maintenance of agent-based software. In addition to this, it showed that the software should be flexible, easy-to-use, scalable, and of high quality. The chapter has also presented the analysis and design intelligent agents that do a good job of acting on their environment.

As for the second chapter of the thesis, it emphasized and illustrated a theoretical background for the research and Fundamental concepts of Agent software. This chapter discussed some of the general principles used in the design of agents throughout the thesis, among which is the principle that agents should know things.

Chapter three of the thesis illustrated a theoretical background for the research and Fundamental concepts of intelligent Agent, which included illustrations of the disciplines of intelligent agents that has emerged largely from research in artificial Intelligence (AI). In fact, one way of defining AI is: "the problem of building an intelligent agent, and illustrate the main reasons why people need software agents" [39]. The chapter has also shown that agent characteristics give a global impression of what an agent "is".

Chapter four of the thesis illustrated and discussed the methodology and plan that has been designed to complete this project. It has also illustrated that Prometheus methodology differs from existing methodologies, and provided distinctive support for issues relating to agent identification. The chapter has shown tools, which are required in the analysis and design process. The next phase of the research was to draw the

final design for the system by using Prometheus methodology, present final features of intelligent agent and illustrate the Graphical user interface for project.

Appendix A of this thesis is completed with the documentation of the source codes and the discussions will touch on the functions in the library Email Agent and most importantly, the sorting process. During the discussions, the source codes for the functions are normally displayed partially as the full source codes are of substantial length.

### **6.1 Recommendations for future work**

As agent-oriented methodologies continue to be developed, research will keep aiming at the direction of determining which agent-oriented methodologies are best suited to support the development of a particular project or system. Hence, there are various future works that can be done in this area as:

1. Extending the model to include various library functions.
2. Applying other methodology.



## Reference

- [1]: A. Jafari, "conceptualizing intelligent agent for teaching and learning", Research paper, University Indianapolis, Indianapolis, Indiana, 2002.
- [2]: A. O'Malley, Scott A. DeLoach," Proceedings of the Second International Workshop On Agent-Oriented", Research paper, Montreal, Canada, Air Force Institute of Technology, May 29th 2001.
- [3]: A. Pakonen, "Information Agent Technology in Process Automation Systems ", Master's Thesis Information and Computer Systems in Automation, helsinki university of technology, November 26, 2004.
- [4]: A. Sturm (joint work with O. Shehory and D. Dori),"Evaluation of Agent-Oriented Methodologies", Research paper, July 2004. Available at: [www.pa.icar.cnr.it/~cossentino/a13f1/docs/evaluation4agentlink.pdf](http://www.pa.icar.cnr.it/~cossentino/a13f1/docs/evaluation4agentlink.pdf), 2006-11-5.
- [5]: C. Cares, "Agent-Oriented Software Engineering: An Introduction" , Research paper, Technical University of Catalonia, Barcelona, Spain, and University of La Frontera, Temuco, Chile.  
Available at: [http://www.agents.com/docs/0608\\_10.pdf](http://www.agents.com/docs/0608_10.pdf). 2006-11- 5.
- [6]: D. Rosa dos Santos, M. Blois Ribeiro, R. Melo Bastos, "A Comparative Study of Multi-Agent Systems, Development Methodologies", Research paper, University of Rio Grande -Porto Alegre Brazil, with resources of Law 8.248/91. Available at: [www.les.inf.puc-rio.br/seas2006/papers/X037.pdf](http://www.les.inf.puc-rio.br/seas2006/papers/X037.pdf) , 2007-01-12.
- [7]: D. Wallace Croft, "Intelligent Software Agents: Definitions and Applications ", Research paper, 1997-10-03.

[www.alumnus.caltech.edu/~croft/research/agent/definition.pdf](http://www.alumnus.caltech.edu/~croft/research/agent/definition.pdf), 2006.10.2.

- [8]: G. Boone, "Re: Agent, &n Intelligent Email Agent" Research paper, Georgia Institute of Technology, 1998  
Available at: [www.cc.gatech.edu/Ngboone](http://www.cc.gatech.edu/Ngboone), 2006-12-7
- [9]: G. Pang, "Implementation of an Agent-Based Business Process", Research paper, Chengdu, Sichuan, China, University Zürich, 2000.
- [10]: Hsi-Kuo Li, "SOFTWARE AGENT", Thesis, Skokie, Illinois, U.S.A December 2002.
- [11]: H. S. Nwana, "Software Agents: An Overview", Research paper. Cambridge University Press, Sept 1996.
- [12]: H. Xu and S. M. Shatz, "A Framework for Model-Based Design of Agent-Oriented Software", Research paper, The University of Illinois, Chicago, 1999.  
Available at: [www.cis.umassd.edu/~hxu/Papers/UIC/TSE.pdf](http://www.cis.umassd.edu/~hxu/Papers/UIC/TSE.pdf) , 2006-7-20.
- [13]: J. Calmet, Pierre Maret and Regine Endsuleit, "Agent-Oriented Abstraction". Research paper, Real Academia de Ciencias, España, 2004.  
Available at: <http://www.rac.es/ficheros/doc/00150.pdf>, 2006-9-14
- [14]: J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf , "Evaluation of Agent-Oriented Software. Methodologies – Examination of the Gap Between Modeling and Platform", Research paper, University of Applied Sciences Hamburg, Berliner Hamburg, Germany, available at [www.informatik.uni-hamburg.de/get\\_doc.php/publications/207/aose04-03.pdf](http://www.informatik.uni-hamburg.de/get_doc.php/publications/207/aose04-03.pdf), 2006-7-20.
- [15]: K. H. Dam, "Evaluating and Comparing Agent-Oriented Software, Engineering Methodologies", Master of Applied Science in Information Technology.

RMIT University, Australia, June 27, 2003.

- [16]: K. Chan, Leon Sterling, Shanika Karunasekera, "Agent-Oriented Software Analysis", Research paper, Software Engineering Conference, Proceedings, University of Melbourne, Australian, pp 20 – 27, 2004.
- [17]: K. H. Dam, Michael Winikoff, "Comparing agent - oriented methodologies ", Research paper, University of Melbourne , Australia, 2003.
- [18]: L. Cernuzzi, G. Rossi, "On the evaluation of agent oriented modeling methods", pp 21–30, Seattle, November 2002.
- [19]: L. Padgham, J. Thangarajah, M. Winikoff, "Tool Support for Agent Development using the Prometheus Methodology", Research paper, RMIT University, Melbourne, Australia, 2005.
- [20]: L. Padgham, M. Winikoff, "Developing Intelligent Agent Systems: A Practical Guide", Research paper, RMIT University, Melbourne, Australia, 2004, ISBN 0-470-86120-7.
- [21]: L. Padgham, M. Winikoff , "The Prometheus Methodology ", Research paper, RMIT University, Melbourne, AUSTRALIA, April 2004.
- [22]: L. Padgham, M. Winikoff, " Prometheus: A Methodology for Developing Intelligent Agents", Research paper, RMIT University, Melbourne, AUSTRALIA available at, <http://www.cs.rmit.edu.au> 2006-11-18.
- [23]: L. Padgham, M. Winikoff. "Prometheus: A pragmatic methodology for engineering intelligent agents". Research paper, pp 97–108, Seattle, 2002.
- [24]: M. Amor, L. Fuentes and A. Vallecillo, "Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA", Research paper,

University de Málaga. Campus de Teatinos, Spain, 2002.

- [25]: M. Amundsen, " SAPI and TAPI Developer's Guide", FIRST EDITION, Book Number: 0-672-30928-9, address Sams Publishing 201 W. 103rd St., Indianapolis, IN 46290, Copyright © 1996.
- [26]: Marc-Philippe Huet, J. Odell, Øystein Haugen, Mariam "Misty" Nodine, Stephen Cranfield, Renato Levy, and Lin Padgham, "Fipa modeling: interaction diagrams", Research paper, under "Working Documents", 2003. FIPA Working Draft, (version 2003-07-02). Available at: [www.auml.org](http://www.auml.org)
- [27]: M. Mohammadian, "Intelligent Agent for data mining and information retrieval", Research paper, University of Canberra, Australia, 2004.
- [28]: M. Wooldridge, J.R Jennings, "Intelligent Agents: Theory and Practice", Research paper, January 1995. Available at: <http://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.html>. 2006-5-14.
- [29]: M. Winikoff, L. Padgham, J. Harland, "Simplifying the Development of Intelligent Agents ". Research paper, RMIT University, Melbourne, Australia.
- [30]: M. Wooldridge, "Agent-Based Software Engineering", Research paper, Mitsubishi Electric Digital Library Group, London, United Kingdom, September 19, 1997.
- [31]: M. Wooldridge, "An Introduction to MultiAgent Systems", Research paper. (Chichester, England), 2002, ISBN 0 47149691X.
- [32]: N. R. Jennings, M. Wooldridge, "Applications of Intelligent Agents", Research paper. Queen Mary & Westfield College, University of London. Available at: <http://www.cs.umbc.edu/agents/introduction/jennings98.pdf>. 2006-5-14

- [33]: O. Arazy and C. Woo, "Analysis and Design of Agent-Oriented Information Systems (AOIS)", Research paper, University of British Columbia, December 19, 1999.
- [34]: O. Shehory, A. Sturm, "Evaluation of Modeling Techniques for Agent -Based Systems", Research paper, the 5th International Conference on Autonomous Agents, Montreal, Canada, May 28-June 01, 2001.
- [35]: O. Shehory, A. Sturm, "Agent-Oriented Software Engineering (AOSE) Methodologies", Research paper, New York, July 2004.
- [36]: P. Cuesta, A. Gómez, J. C. González, and F. J. Rodríguez, "A Framework for Evaluation of Agent Oriented Methodologies?", Research paper, University of Vigo Ourense E-32004 (SPAIN).  
Available at: <http://www.montealegre.ei.uvigo.es/gwai.pdf>
- [37]: P. Giorgini "Agent-Oriented Methodologies: An Introduction", Chapter 1, University of Trento, Italy, Brian Henderson-Sellers University of Technology, Sydney, Australia, 2005.
- [38]: P. Havaldar, S. Deloach, "Compare and Contrast five major Agent Oriented Software Engineering methodologies...", Research paper, available at:  
<http://www.cis.ksu.edu/~padmaja/papers/abstract.doc.2007-2-3>
- [39]: S. Russell and P. Norvig, " Artificial Intelligence: A Modern Approach", Prentice -Hall, 1995.
- [40]: V. Gorodetsky, J. Liu, V. A. Skormin (Eds.), " Autonomous Intelligent Systems: Agents and Data Mining", international workshop, AIS-ADM 2005, St.Petersburg, Russia, June 2005.

- [41]: W. Brenner, R. Zarnekow, H. Wittig, "Intelligent software agents: foundations and applications", Research paper, springer - verlag berlin Heidelberg, 1998.
- [42]: W. Wobcke, "Intelligent Agent's technology review", Research paper. University of New South Wales Australia, October 2004.

## **Appendix A: Source code Discussions**

This chapter of the thesis is used to illustrate the discussions on the source code of the intelligent agent. The discussions will touch on the functions in the library Email Agent and, most importantly, the sorting process. During the discussions, the source codes for the functions are normally displayed partially as the full source codes are of substantial length.

### **A.1 Coding the Support Routines**

The real heart of the library Email Agent program is the support routines. There are three main sets of routines in the program:

1. Initialization routines
2. List-handling routines
3. Message-processing routines

The next three sections of this chapter walk you through the process of building the support routines for the library Email Agent program.

#### **A.1.1 The Initialization Routines**

The initialization routines declare the global variables and set them to their initial values. There are also routines to handle the reading and writing of configuration values and the storing and retrieving of the test, action, and rule records.

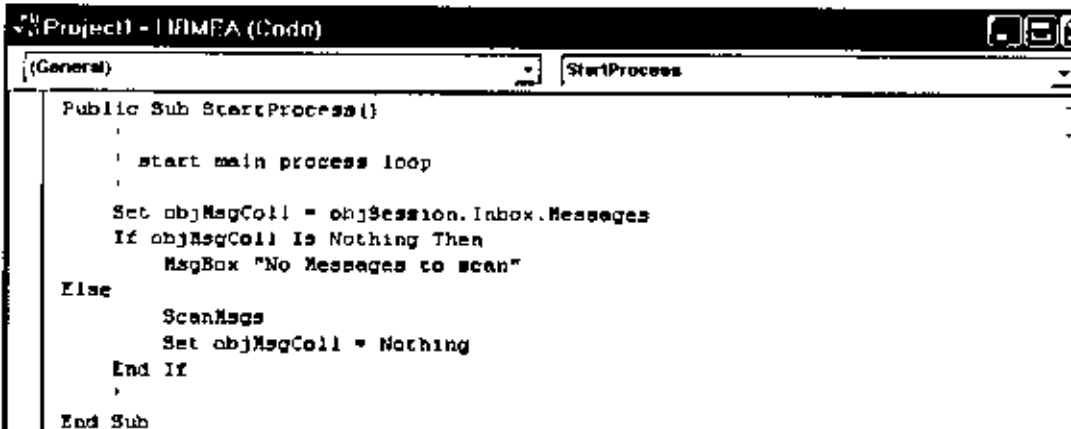
### A.1.2 The List-Handling Routines

The next set of routines handles the addition and deletion of records from the rules, tests, and actions lists. There are also two routines that handle the populating of the list controls on the library Email Agent forms.

### A.1.3 The Message Processing Routines

This last set of routines is where the library services are finally used. The goal of the message processing routines is to inspect each message in the user's inbox and check the messages against the rules that have been established for the library Email Agent. The top-level routines are StartProcess and ScanMsgs. The StartProcess routine is called by the Timer1\_Timer event or by pressing the Start button on the main form. StartProcess checks to see if there are any messages in the user's inbox. If there are, then the ScanMsg routine is called to process each message.

#### 1. Adding the StartProcess code:



```

Project1 - IIMFA (Code)
[General] StartProcess
Public Sub StartProcess()
    ' start main process loop

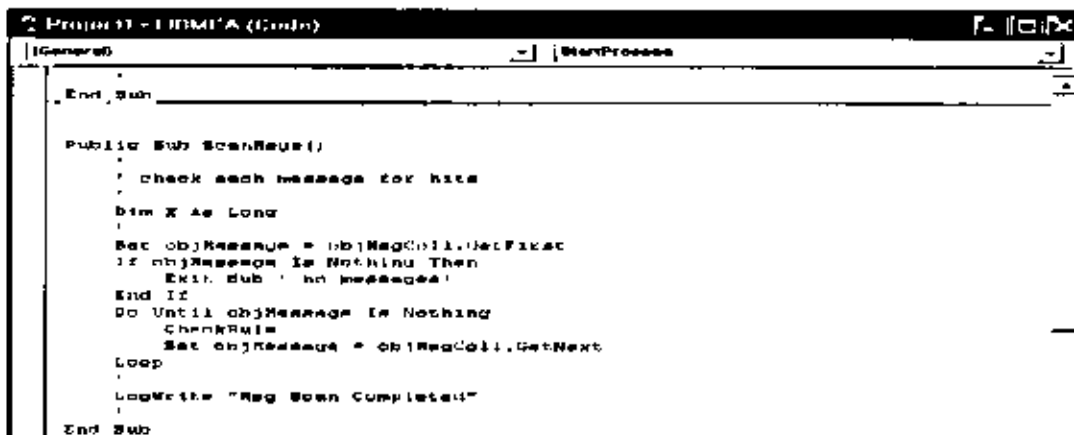
    Set objMsgColl = objSession.Inbox.Messages
    If objMsgColl Is Nothing Then
        MsgBox "No Messages to scan"
    Else
        ScanMsgs
        Set objMsgColl = Nothing
    End If
End Sub

```



The Start Process routine attempts to create a message collection object based on the Session inbox. If that is successful, the Scan Msgs routine can be called.

## 2. Adding the ScanMsgs routine:



```

Project1 - IBM MQA (Code)
|General| StartProcess
|End Sub|
Public Sub ScanMsgs()
    ' check each message for hits
    '
    Dim X As Long
    Set objMessage = objMsgColl.GetFirst
    If objMessage Is Nothing Then
        Exit Sub ' no messages!
    End If
    Do Until objMessage Is Nothing
        CheckRule
        Set objMessage = objMsgColl.GetNext
    Loop
    LogWrite "Msg Scan Completed"
End Sub

```

The ScanMsgs routine selects each message in the collection and submits it to the Check Rule routine for processing.

### 3. Adding the CheckRule routine:

```

(General) CheckRule
Public Sub CheckRule()
    ' check for rule hit
    Dim X As Integer
    Dim cCmd As String
    Dim bAct As Boolean
    Dim objAddrEntry As Object
    '
    On Error GoTo CheckRuleErr
    '
    For X = 0 To iRuleCount - 1
        cCmd = ParseWord(cRuleTest(X))
        '
        Select Case UCCase(cCmd)
            Case UCCase(cTestSender)
                Set objAddrEntry = objMessage.Sender
                If objAddrEntry Is Nothing Then
                    Exit Case
                Else
                    If CheckSender(X, objMessage.Sender.Name) = True Then
                        DoAction X
                    End If
                End If
            Case UCCase(cTestSubject)
                If CheckSubject(X, objMessage.Subject) = True Then
                    DoAction X
                End If
            Case UCCase(cTestPriority)
                If CheckPriority(X, objMessage.Importance) = True Then
                    DoAction X
                End If
            End Select
        Next X
    Exit Sub
CheckRuleErr:
    MsgBox Error0, vbCritical, "CheckRuleErr [" & CStr(X) & "]"
End Sub

```

Several things are going on in this routine. First, the first "word" on the line is removed using the ParseWord() function. This word is then used to determine the type of test to perform on the message. The appropriate check subroutine is called (CheckSender, CheckSubject, CheckPriority) and, if the return is positive, the DoAction routine is called to handle the action portion of the rule.

#### 4. Adding the ParseWord function:

```

(General) ParseWord
Public Function ParseWord(cLine As String) As String
    ' pick a word off line
    Dim nPos As Integer
    nPos = Instr(cLine, " ")
    If nPos <> 0 Then
        ParseWord = Left(cLine, nPos - 1)
    Else
        ParseWord = ""
    End If
End Function

```

The ParseWord() function accepts a string and returns the first full word found in the string. For example ParseWord("SENDER Smith") would return SENDER. This is used to pull the command portion of a test or action record.

The CheckRule routine you entered earlier, uses ParseWord() to get the message portion command of a rule (SENDER, SUBJECT, PRIORITY). This value is then used to call the three message-part-specific check routines (CheckSender, CheckSubject, and CheckPriority).

## 5. Adding the CheckSender function:

```

(General)                                CheckSender
End Function
Public Function CheckSender(nRule As Integer, cName As String)
' check name against sender test
Dim nPos As Integer
Dim cSender As String
nPos = InStr(cRuleTest(nRule), " ")
If nPos <> 0 Then
    cSender = Trim(Mid(cRuleTest(nRule), nPos + 1, 255))
End If
cSender = Trim(UCase(cSender))
cName = Trim(UCase(cName))
Select Case UCase(cRuleCompare(nRule))
Case UCase(cIsEqualTo)
    If cSender = cName Then
        CheckSender = True
    Else
        CheckSender = False
    End If
Case UCase(cIsGreaterThan)
    If cSender > cName Then
        CheckSender = True
    Else
        CheckSender = False
    End If
Case UCase(cIsLessThan)
    If cSender < cName Then
        CheckSender = True
    Else
        CheckSender = False
    End If
Case UCase(cIsContainedIn)
    If InStr(cSender, cName) <> 0 Then
        CheckSender = True
    Else
        CheckSender = False
    End If
End Select
End Function

```

Note that this routine uses a SELECT CASE structure to handle the compare portion of the rule. After locating the correct compare operation, CheckSender tests the Sender portion against the Name in the rule and returns the result (TRUE or FALSE). The CheckSubject and CheckPriority functions work the same way. The only difference is that the CheckPriority function does not test for CI ("is contained in").

## 6. Adding the CheckSubject routine:

```

(Generic) CheckSubject
Public Function CheckSubject(nRule As Integer, cSubjMsg As String) As Boolean
    ' check subject against message text
    Dim nPos As Integer
    Dim cSubjRule As String
    nPos = InStr(cRuleText(nRule), " ")
    If nPos <> 0 Then
        cSubjRule = Trim(Mid(cRuleText(nRule), nPos + 1, 255))
    End If
    '
    cSubjRule = UCase(Trim(cSubjRule))
    cSubjMsg = UCase(Trim(cSubjMsg))
    Select Case UCase(cRuleComp(nRule))
        Case UCase(cIsEqualTo)
            If cSubjRule = cSubjMsg Then
                CheckSubject = True
            Else
                CheckSubject = False
            End If
        Case UCase(cIsLessThan)
            If cSubjRule < cSubjMsg Then
                CheckSubject = True
            Else
                CheckSubject = False
            End If
        Case UCase(cIsGreaterThan)
            If cSubjRule > cSubjMsg Then
                CheckSubject = True
            Else
                CheckSubject = False
            End If
        Case UCase(cIsContainedIn)
            If InStr(cSubjRule, cSubjMsg) <> 0 Then
                CheckSubject = True
            Else
                CheckSubject = False
            End If
    End Select
End Function

```

## 7. Adding the CheckPriority routine:

```

(General) | CheckSubject
End Function
Public Function CheckPriority(nRule As Integer, nImpMsg) As Boolean
    ' check subject against message test
    '
    Dim nPos As Integer
    Dim cImpRule As String
    Dim nImpRule As Integer
    '
    nPos = InStr(cRuleTest(nRule), " ")
    If nPos <> 0 Then
        cImpRule = Trim(Mid(cRuleTest(nRule), nPos + 1, 255))
    End If
    '
    nImpRule = Val(cImpRule)
    '
    Select Case UCase(cRuleCompare(nRule))
        Case UCase(cisEqualTo)
            If nImpRule = nImpMsg Then
                CheckPriority = True
            Else
                CheckPriority = False
            End If
        Case UCase(cisLessThan)
            If nImpRule < nImpMsg Then
                CheckPriority = True
            Else
                CheckPriority = False
            End If
        Case UCase(cisGreaterThan)
            If nImpRule > nImpMsg Then
                CheckPriority = True
            Else
                CheckPriority = False
            End If
    End Select
    '
End Function

```

If the *Checknmn* routine returns TRUE, an action must take place. The *DoAction* routine is used to execute the appropriate e-mail action. *DoAction* accepts the index to the rule as its only parameter. Like the *CheckRule* routine, *DoAction* uses a SELECT CASE structure to act on each command word (NOTIFY, COPY, MOVE, FORWARD, and REPLY).

## 8. Adding the DoAction routine:

```

(General)                                     Check Subject
-----
End Function
Public Sub DoAction(nRule As Integer)
    ' handle valid action
    ' nRule points to rule in array
    ' use current objMessage
    '
    Dim cCmd As String ' action command
    Dim cTarget As String ' action target
    Dim nPos As Integer
    '
    ' get command and target
    cCmd = ParseWord(cRuleAction(nRule))
    nPos = InStr(cRuleAction(nRule), " ")
    If nPos <> 0 Then
        cTarget = Trim(Mid(cRuleAction(nRule), nPos + 1, 255))
    End If
    '
    ' now execute command
    Select Case UCase(cCmd)
        Case UCase(cActionMove)
            MsgMoveCopy "MOVE", cTarget, objMessage
        Case UCase(cActionCopy)
            MsgMoveCopy "COPY", cTarget, objMessage
        Case UCase(cActionForward)
            MsgFwdReply "FORWARD", cTarget, objMessage
        Case UCase(cActionReply)
            MsgFwdReply "REPLY", cTarget, objMessage
        Case UCase(cActionNotify)
            MsgNotify cTarget, objMessage
    End Select
End Sub

```

There are only three routines used to act on all five commands. This is because the FORWARD and REPLY commands act on messages, and the COPY and MOVE commands act on folders. Only one routine is needed for each (with slight behavior changes within each routine).

The NOTIFY option is the easiest to handle. All that is needed is a pop-up dialog box when the message arrives. The routine needed for forwarding and replying to messages involves making a new message (with the contents of the original) and sending it to a new address. Since this is actually a messaging operation, Send function to force the message into the transport for delivery.

## 9. Adding the MsgNotify routine:

```

(General) DoAction
Public Sub MsgNotify(cNotify As String, objMsg As Object)
    Dim cMsg As String
    cMsg = "Message Notification for ["
    cMsg = cMsg & objMsg.Subject
    cMsg = cMsg & "] from ["
    cMsg = cMsg & objMsg.Sender.Name & "]"
    ' send out pop-up?
    If cNotifyDialogValue = "1" Then
        MsgBox cMsg, vbExclamation, "NAPI Email Agent Notification"
    End If
    LogWrite (cMsg)
End Sub

```

## 10. Adding the MsgFwdReply routine:

```

(General) MsgFwdReply
Public Sub MsgFwdReply(cEvent As String, cDestAddr As String, objMsg As Object)
    Dim cMsg As String
    Dim objLocalMsgColl As Object
    Dim objCopyMsg As Object
    Dim cHeader As String
    Dim cSubjPrefix As String
    cTarget = CDate(cEvent)
    cHeader = "<----- Message " & cEvent
    cHeader = cHeader & " from ["
    cHeader = cHeader & objSession.Name
    cHeader = cHeader & "] by NAPI Email Agent ----->"
    cHeader = cHeader & Chr(13) & Chr(10) & Chr(13) & Chr(10)
    If cEvent = "REPLY" Then
        cSubjPrefix = "RE: "
    Else
        cSubjPrefix = "FW: "
    End If
    Set objLocalMsgColl = objSession.Outbox.Messages
    Set objCopyMsg = objLocalMsgColl.Add
    With objCopyMsg
        .Subject = cSubjPrefix & objMsg.Subject
        .Text = cHeader & objMsg.Text
    End With
    ' add recipient
    Set objRecipient = objCopyMsg.Recipients.Add
    objRecipient.Name = cDestAddr
    objRecipient.Type = mapsTo
    objCopyMsg.Recipients.Resolve showDialog:=False
    objCopyMsg.Update
    objCopyMsg.Send showDialog:=False ' delete old message?
    If cMailFwdingValue = "1" and cEvent = "FORWARD" Then
        objMessage.Delete
    End If
    If cMailReplyingValue = "1" and cEvent = "REPLY" Then
        objMessage.Delete
    End If
    objSession.Outbox.Update ' send out status
    cMsg = cEvent & " Message ["
    cMsg = cMsg & objMsg.Subject
    cMsg = cMsg & "] to [" & cDestAddr & "]"

```

There are a few things to keep in mind about this routine. First, a good REPLY routine should allow users to attach, or append, a text message to the original.



Here, that code is left out for brevity. You will also notice that only one recipient is added to the note. In some cases, it is possible that more than one person should receive the forwarded message. This can be handled by using distribution lists. Finally, there is no code here to handle any attachments to the original note. This should be added in a production environment. The other action to be handled by the Library Email Agent is copying or moving messages to other folders. This is accomplished using the .Update method. Moving messages is similar to posting them. For this reason you do not want to attempt to "send" the message.

## 11. Adding the MsgMoveCopy routine:

```

General
MsgMoveCopy
Public Sub MsgMoveCopy(cEvent As String, cFolder As String, objMsg As Object)
    Dim objLocalFolder As Object
    Dim objLocalMsg As Object
    Dim objLocalRecipient As Object
    Dim cMsg As String
    Dim cFldrID As String
    Dim cRecipName As String
    Dim cHeader As String
    Dim i As Integer ' carry sender info with you
    cHeader = "<----- " & WCase(cEvent) & " from ("
    cHeader = cHeader & objMsg.Sender.Name
    cHeader = cHeader & ") by RAFI Email Agent -----)"
    cHeader = cHeader & Chr(13) & Chr(10) & Chr(13) & Chr(10) ' look for folder
    cFldrID = FindFolder(cFolder)
    If cFldrID = "" Then
        Exit Sub
    End If ' move to folder
    Set objLocalFolder = objSession.GetFolder(cFldrID)
    Set objLocalMsg = objLocalFolder.Messages.Add ' copy from objmsg to objlocalmsg
    With objLocalMsg
        .DeliveryReceipt = objMsg.DeliveryReceipt
        .Encrypted = objMsg.Encrypted
        .Importance = objMsg.Importance
        .ReadReceipt = objMsg.ReadReceipt
        .Sent = objMsg.Sent
        .Signed = objMsg.Signed
        .Subject = objMsg.Subject
        .Submitted = objMsg.Submitted
        .Text = cHeader & objMsg.Text
        .TimeReceived = objMsg.TimeReceived
        .TimeSent = objMsg.TimeSent
        .Type = objMsg.Type
        .UnRead = objMsg.UnRead
    End With ' add recipients
    For i = 1 To objMsg.Recipients.Count Step 1
        cRecipName = objMsg.Recipients.Item(i).Name
        If cRecipName <> "" Then
            Set objLocalRecipient = objLocalMsg.Recipients.Add
            objLocalRecipient.Name = cRecipName
        End If
    Next i

```

Again, a few things worth pointing out here. First, moving or copying messages to other folders requires that you actually *find* the target folder first. This is not as simple as looking up a name in a list. MAPI message stores are recursively hierarchical. That means that folders can exist within folders. In addition, MAPI does not publish a list of the folder tree—you must traverse it each time yourself. This means you need your own FindFolder routine.

Notice that the process of moving a message really involves creating a copy in the new folder. The code here copies the most commonly used items, but does not copy any attachments. Keep in mind that some properties may not exist for some messages.

The last routine you need for the Library Email Agent application is the FindFolder function. This routine accepts a folder name and returns its unique MAPI ID value.

## 12. Adding the FindFolder function:

```

General FindFolder
Public Function FindFolder(cFldrName As String) As String
    ' see if you can locate the requested folder
    ' if found, return the unique folder ID
    '
    Dim cRtnID As String
    Dim cTempID As String
    Dim objInfoStore As Object
    Dim objTempFldr As Object
    Dim objTempColl As Object
    Dim X As Integer
    '
    cRtnID = "" ' assume not found
    '
    ' scan the folder collection
    Set objTempColl = objSession.Inbox.Folders
    Set objTempFldr = objTempColl.GetFirst
    Do
        cTempID = objTempFldr.ID
        If UCCase(objTempFldr.Name) = UCCase(cFldrName) Then
            cRtnID = objTempFldr.ID
            Exit Do
        End If
        Set objTempFldr = objTempColl.GetNext
    Loop While objTempFldr.Name <> ""
    '

```

## A.2 Building (ILEA) Actions, Tests, and Rules

We need to add test, action, and rule records to the Library Email Agent database. Start the library Email Agent program and enter the tests and actions shown in Table

<i>Record Type</i>	<i>Contents</i>
Test	SENDER Boss
	SUBJECT MAPI
	SUBJECT SALES
	PRIORITY 0
	SENDER Assistant
Action	FORWARD Boss
	MOVE Urgent
	COPY MAPI
	REPLY Sales
	NOTIFY Me

Table (A-1). (ILEA) tests and actions.

Table (A-1) shows how the library Email Agent looks after the tests and actions have been entered. Then we are ready to create some rules for the Intelligent Library Email Agent. By Enter the rules shown in Table

<i>Rule Name</i>	<i>Test</i>	<i>Compare</i>	<i>Action</i>
Bosses Mail	SENDER Boss	EQ	COPY Urgent
Save MAPI	SUBJECT MAPI	CI	COPY MAPI
Assistant Routing	SENDER Assistant	EQ	FORWARD Boss
Wake Me Up	PRIORITY 0	EQ	NOTIFY Me
Sales Handler	SUBJECT SALES	CI	REPLY Sales

Table (A-2). Adding rules to the (ILEA).

Table (A-2) shows what your screen should look like as you build library Email Agent rules.

## المخلص

"الوكيل" أو "الوكيل البرمجي" أصبحت كلمات شعبية في برامج الحاسوب ، السبب وراء ذلك أنها مستندة على الذكاء الاصطناعي لكن الأعمال فقط تكون في حقل معين. برامج الوكيل (الحريف) طوّرت للعديد من الاستعمالات المختلفة في مجالات عدة بسبب قدرتها الاستثنائية لتكثيف لحقل معين للاهتمام [10].

تقدّم هذه الأطروحة نظرة عامة عن وكيل البرامج أولاً، والتي تبدأ من مقدمة لتعريف وكيل البرامج، مصطلحاته، وأساسياته الأخرى، بعد ذلك، المفاهيم ومواصفات الوكيل 'الذكي' تؤدي لوصف كيف أن وكيل البرامج يعمل وراء هذه الصورة. هذا يتضمن عملاً، عملية، سلوك مستقل ذاتياً واتصال الوكلاء. الأسلوب الذي يتبعه وكيل برامج نتاج أهدافه من الإنجاز و تقييم تقدّمه مهم جداً أيضاً في إبقاء الوكيل يعمل في الطريق الصحيح، الأطروحة تقدّم علم منهج للتحليل وتصميم الوكيل الموجه والسماح الرئيسية لتطوير برامج المعتمدة على فكرة الوكيل، وقد تم التركيز على أحد أكثر المنهجيات المعروفة كعلم منهج Prometheus لتطوير أنظمة الوكيل الذكية، وبعد ذلك، نناقش مراحل التحليل وتصميم نظام الوكيل طبقاً للمنهجية المختارة.

في هذه الأطروحة تم اختيار وظيفة واحدة من نظام المكتبة لبرمجتها، هذه الوظيفة تتعلق بوكيل بريد المكتبة الإلكتروني، هذا لأن البريد الإلكتروني أصبح أحد أدوات الاتصال الرئيسية بين الناس حول العالم بغض النظر عن المسافات بينهم ، إن مشروع وكيل بريد المكتبة الإلكتروني هو تطبيق وكيل برامج مثالي جداً. إن مجموعة برامج وكيل بريد المكتبة الإلكتروني الذكي (أي إل إي أي) متضمنة في الملحق أي .

## الإهداء

إلى أمي الحبيبة  
إلى أبي الغالي  
إلى زوجي العزيز وابنتي  
فرحة الدنيا رشا  
إلى أخي خالد القلب الكبير  
والنبع الصافي  
إلى أحبائي إخوتي  
وأخواتي



التاريخ :

الموافق : 25 - 10 - 2005

الرقم الاشاري : 1449 / 1.1.1.1

## كلية العلوم

قسم الحاسوب

تفوان البحث

(( استخدام مفاهيم الأنظمة الذكية في تحليل  
وتصميم إدارة مكتبة ))

مقدمة من الطالبة

نعيمه موسى أمريض

\* \* لجنة المناقشة :

.....

الدكتور / عبد الحميد محمد عبد الكافي  
( مشرف الرسالة )

.....

الدكتور / علي أحمد عبد العزيز  
( ممتحن داخلي )

.....

الدكتور / محمد أبو القاسم الرتيمي  
( ممتحن خارجي )

يعتمد  
د. أحمد فرج المصوب  
أمين اللجنة الشعبية لكلية العلوم



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

(( سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا إِنَّكَ أَنْتَ  
الْعَلِيمُ الْحَكِيمُ ))

آية ( 32 ) من سورة البقرة





جامعة التحدي - كلية العلوم

استخدام مفاهيم الأنظمة الذكية في تحليل و تصميم نظام إدارة مكتبة

هذه الرسالة مقدمة لقسم علوم الحاسوب كمتطلب جزئي للحصول على درجة  
الماجستير في علوم الحاسوب

مقدمة من الطالبة:  
نعيمة موسى أمريض

إشراف : د. عبد الحميد محمد عبد الكافي

العام الجامعي 2007/2006