# Using Software Abstraction to Develop an Agent Based System

Zulaiha Ali Othman

A thesis submitted in partial fulfilment of the requirement of
Sheffield Hallam University
for the degree of Doctor of Philosophy

2004

School of Computing and Management Sciences
Sheffield Hallam University

# Abstract

The main contribution of the thesis is to present a systematic process to develop an agent-based system that assists a system developer to construct the required system, through a series of modelling activities, employing several levels of abstraction to show the milestones and produce intermediate deliverables. Current practice emphasises "downstream activities" such as implementation at the expense of "upstream activities" such as "modelling".

The research has found that the development process for an agent system consists of three phases: *agent system development, agent environment development* and *agent system deployment*. The first and second phases represent an intertwined spiral model. All three phases themselves consist of three stages. Each phase employs different development techniques and each stage uses appropriate models and tools such as problem domain model, agent use cases, scenarios, agent system architecture, plan model and individual agent model.

The proposed agent development method is applied to two case studies: *a Filtering Agent System* and *Diabetic Consultation System.* Both systems have been implemented and tested. Three distinct ways were used to evaluate the proposed method. First, comparing with the criteria of a methodology. Second, comparing it with the current agent-oriented methodologies. Third, informal observations from a potential user community.

In conclusion, the research has demonstrated an effective synthesising process to build a set of agent concepts, development life-cycle and modelling to show a systematic process for developing agent systems. Moreover, by employing a whole host of software abstraction tools and techniques in the process, two benefits accrue: the introduction of more 'up stream' activities as well as placing modelling at the heart of the process. Illustratively, we could say that the modelling presented here does for agent systems what data flow diagram and data entity diagram have done for structured methodologies, i.e. raise the level of abstraction employed.

## Acknowledgements

I would like to express my deepest thanks to my kindly supervisor, Professor Dr. Jawed Siddiqi who have supervised, influenced and inspired me throughout the duration of this research project. Similarly to Dr. Peter Collingwood who has helping me in the early stage of this research and make comment on the thesis. Never ever forget to Dr John Stone who has kindly help with the English and proof-reading of the final thesis.

Many thanks to all the staffs of Sheffield Hallam University, who were involved whether directly or indirectly especially, Paul Manning, Darren, Chris, Angela, Lind, and all CMS and CIS staff who provided necessary environment for the achievement of this research, especially to the postgraduate co-tutor, Ian Morrey, who always give his full attention to all research students. Great thanks to all other researchers in similar research agent areas such as Afsaneh, Abdullah and Saadat for their technical comments.

To all my friends that I know them in Sheffield such as Zainah, Mariani, Liana, Tracy and both Aminah, who always cheers up my life in here, our brotherhood will never be forget forever.

No amount of thanks can suffice for my husband Roslan for encouraging me all the time, from the beginning to the end of this research, sharing the hardship of doing research and helping me to use my time efficiency. To my three sons Mutasim, Muaz and Mustaqiim, a deep appreciation from your mum for being patient, understanding and tolerant for several years of motherless days. Your great behaviour really encouraged me throughout the research.
I would like to extend my heartfelt thanks to my parents, brothers and sisters, whose constantly provided great advice and encouragement. All of you are the sources of my determination to complete the PhD.

Finally, thanks to my sponsor, the National University of Malaysia, which provide financial support for this research.

May God Blessed all of You,..

## Contents Pages

*Abstract* ... i
*Acknowledgement* ... ii
*Abbreviations* ... vi
*Figures* ... viii
*Tables* ... x

CHAPTER ONE

# Introduction

1.0. MOTIVATION OF THE RESEARCH. ... *1*
1.1. THE CONTRIBUTION MADE IN THE RESEARCH PROJECT. ... *5*
1.2. TOUR OF THE THESIS. ... *6*

CHAPTER TWO

# Background Literature Review

2.0. INTRODUCTION. ... *8*
2.1. AGENTS. ... *8*
*2.1.1. Characterising Agents.* ... *9*
*2.1.2. Key Issues for Agent Technology* ... *12*
*2.1.3. Status of Agent Technology* ... *21*
2.2. SOFTWARE DEVELOPMENT METHODOLOGY. ... 25
*2.2.1. Principles of Software Development Methodology* ... *25*
*2.2.2. System Development Approach.* ... *27*
2.3. SOFTWARE ABSTRACTION ... *32*
*2.3.1. Software architecture.* ... *33*
*2.3.2. Architecture Patterns.* ... *34*
*2.3.3. Design Patterns.* ... *35*
*2.3.4. Architectural Description Language(ADL).* ... *36*
*2.3.5. Frameworks.* ... *37*
*2.3.6. Framework Development Process.* ... *40*
2.4. THE CRITERIA OF AN AGENT BASED SYSTEM ... *41*
2.5 SUMMARY ... *47*

CHAPTER THREE

# The Building Blocks Towards a Process for Agent System Development

3.0. INTRODUCTION. ... *49*
3.1. DERIVATION PROCESS FOR THE DEVELOPMENT OF AN AGENT SYSTEM. *51*
*3.1.1. Examining Software Abstraction.* ... *54*

*3.1.2. Analysis of Existing Agent-Oriented Methodologies ....................................55*
*3.1.3. Reverse Engineering ....................................................................................55*
*3.1.4. Scrutiny of the Existing Environment..........................................................57*
3.2. THE ESSENCE OF THE DERIVATION OF RESULT TOWARDS THE PROPOSAL OF AN AGENT SYSTEM METHOD. ................................................................*58*
*3.2.1. Analysis and Design Concepts for Development Agent System........................59*
*3.2.2 Software Development Process.....................................................................67*
*3.2.3 Analysis, Design and Modelling Techniques for Agent System Development ..69*
*3.2.4. Analysis, Design and Modelling Techniques for Development Agent Environment ..........................................................................................................77*
3.3. SUMMARY.................................................................................................85

**CHAPTER FOUR**

## Towards a Method to Develop an Agent-based System

4.0. INTRODUCTION.........................................................................................86
4.1. AGENT SYSTEM DEVELOPMENT.......................................................................*92*
*4.1.1. Analysis Requirement...................................................................................93*
*4.1.2. Agent System Level Design.........................................................................101*
*4.1.3. Individual Agent Design..............................................................................109*
*4.1.4. Refinement..................................................................................................113*
4.2. AGENT ENVIRONMENT DEVELOPMENT ...........................................................*115*
*4.2.1. Agent Environment Analysis.......................................................................117*
*4.2.2. Agent Environment Architectural Design. ....................................................124*
*4.2.3. Component Design .....................................................................................128*
*4.2.4. Development of Jade Agent Environment. ....................................................129*
4.3. REUSE OF 'OFF-THE-SHELF' AGENT ENVIRONMENTS. .....................................*132*
*4.3.1. Process to Reuse the Off-the-Shelf Agent Environment...................................133*
*4.3.2. Common Features of an Agent Environment..................................................135*
4.4. AGENT SYSTEM DEPLOYMENT.......................................................................*138*
*4.3.3. Deployment Design Process .......................................................................139*
4.5. SUMMARY.................................................................................................*142*

**CHAPTER FIVE**

## Development of Agent System - Case Studies

5.0. INTRODUCTION.........................................................................................*144*
5.1. FILTERING AGENT SYSTEM...........................................................................145
*5.1.1. Requirement Specification...........................................................................145*
*5.1.2. PHASE 1: Agent System Development..........................................................147*
*5.1.3. PHASE 2: Agent Environment Development...................................................173*
*5.1.4. PHASE 3: Agent System Deployment. ..........................................................178*
5.2. DIABETICS CONSULTATION SYSTEM (DCS) .....................................................*181*
*5.2.1. Requirement Specification...........................................................................181*
*5.2.2. PHASE 1: Agent System Development..........................................................182*
*5.2.3. PHASE 2: Agent Environment Development...................................................207*

*5.2.4. 'PHASE 3: Agent System Deployment*......................................................*212*
5.3. SUMMARY......................................................*213*

CHAPTER SIX

## Evaluation and Discussion.

6.0 EVALUATION PROCESSES......................................................*215*
6.1 EVALUATION AGAINST THE CRITERIA OF A SOFTWARE DEVELOPMENT METHODOLOGY......................................................*218*
6.2 COMPARISON WITH THE CURRENT AGENT-ORIENTED METHODOLOGIES......................................................*225*
*6.2.1 Agent Paradigm.*......................................................*226*
*6.2.2 Development process and Techniques*......................................................*228*
*6.2.3 Modelling Techniques and Notations.*......................................................*234*
6.3 POTENTIAL COMMUNITY EVALUATION......................................................*242*
6.4 ANALYSIS OF EVALUATION RESULT......................................................*244*
6.5 DISCUSSION......................................................*251*
6.6 FUTURE RESEARCH......................................................*253*
6.7 CONCLUSION......................................................*256*

REFERENCES *259*

INTERNET REFERENCES *278*

APPENDIX A :Analysis Existing Agent-Oriented Methodologies *281*

APPENDIX B : Reverse Engineering *294*

APPENDIX C : The Scrutiny of Agent Frameworks *319*

APPENDIX D : Software Abstraction in Agent Environment *336*

APPENDIX E : Agent Modelling and Notation *346*

## Abbreviations

| | |
|---|---|
| ACL | Agent Communication Language |
| ADL | Architecture Description Language |
| AEA | Agent Environment Analysis |
| AEAD | Agent Environment Architectural Design |
| AEAOMs | Analysis of the Existing Agent-Oriented Methodologies |
| AED | Agent Environment Development(AED) |
| AI | Artificial Intelligent |
| AR | Analysis Requirement |
| ASD | Agent System Development |
| ASDO | Agent System Deployment |
| ASLD | Agent System Level Design |
| AUML | Agent Unified Modelling Language |
| BDI | Beliefs, Desire and Intention |
| CD | Component Design |
| CORBA | Common Object Request Broker Architecture |
| DCOM | Distributed Common Object Management |
| DCS | Diabetics Consultation System |
| FAS | Filtering Agent System |
| FIPA | Foundation for Intelligent Physical Agents |
| FIPA-OS | FIPA-Operating System |
| FTP | File Transfers Protocol |
| GP | General Practice |
| GSM | Global System for Mobile Communication |
| HTML | Hypertext Markup Language |
| HTTP | High Transportation Protocol |
| IAD | Individual Agent Design |
| ISO | International Standardisation Organisation |
| JDBC | Java Database Compiler |
| KIF | Knowledge Interchange Format |
| KQML | Knowledge Query Manipulation Language |
| OFSAE | ‘Off-the-Shelf’ Agent Environment |
| OMG | Object Management Group |
| RE | Reverse engineering |
| RMI | Remote Method Invocation |

| | |
|---|---|
| SAEF | Scrutiny of Existing Agent Frameworks |
| SL | Semantic Language |
| SMTP | Simple Mail Transfer Protocol |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| UDP | User Datagram Protocol |
| UML | Unified Modelling Language |
| WCTP | Wireless Communication Transportation Protocol |
| XML | Extensible Markup Language |

## Figures

Figure 1.1: An Overview of the Research Process....................................................................*4*
Figure 2.1: Characterisation of a Multi-agent System....................................................................*13*
Figure 2.2: Components of an Agent Architecture. ....................................................................*13*
Figure 2.3: Inter-agent Communication between Two Heterogeneous Agents ........................*15*
Figure 2.4: Example of a KQML message....................................................................17
Figure 2.5: Example of FIPA language....................................................................*18*
Figure 2.6: The Evolution of the Software Development Methodologies ................................*28*
Figure 2.7: Dissemination of the Use Framework for Traditional and Agent-based Application ....................................................................39
Figure 2.8: The Ability of Agent to Communicate Across Multi-Agent System......................45
Figure 3.1: Devivation Process Towards Development of Agent System Development.........51
Figure 3.2: The Sequence of the Derivation Process ....................................................................*52*
Figure 3.3: Identification of Software Abstraction Process ....................................................54
Figure 3.4: The Reverse Engineering Process....................................................................56
Figure 3.5: Agent Overview....................................................................63
Figure 3.6: Agent Modelling....................................................................67
Figure 3.7: A Life-cycle of an Agent System Development....................................................68
Figure 3.8: The Agent System Design for Filter Agent System..............................................72
Figure 3.9: Interaction Modelling Between Two Agents....................................................74
Figure 3.10: A UserModelAgent Model ....................................................................75
Figure 3.11: Agent Design Framework....................................................................76
Figure 3.12: The Agent Environment Reference Architecture................................................83
Figure 4.1: The Artefacts of Agent Development Method .................................................. 87
Figure 4.2: Agent System Design Artefacts....................................................................93
Figure 4.3: Agent Problem Domain ....................................................................94
Figure 4.4: The Filtering Application System Goal Model....................................................96
Figure 4.5: The User Event Use Case ....................................................................98
Figure 4.6: The Use Case for Filtering Application....................................................................99
Figure 4.7: Filtering Application Agent Identification....................................................103
Figure 4.8: UserAgent Abstraction ....................................................................104
Figure 4.9: The Agent System Architecture....................................................................106
Figure 4.10: The Plan Sequence Model ....................................................................107
Figure 4.11: The Plan Activity Model....................................................................108
Figure 4.12: UserAgent Model....................................................................110
Figure 4.13: Communication Data Structure ....................................................................110
Figure 4.14: UserModelAgent Plan Model....................................................................111
Figure 4.15: The Attributes to Define the User profile.................................................... 112
Figure 4.16: The Association between FilterAgent and Learning Techniques...................... 112
Figure 4.17: Agent Environment Development Process .................................................... 116
Figure 4.18: The Artefact of Identifying the Agent Environment Requirement......................118
Figure 4.19: The Agent Environment Design Artefacts .................................................... 125
Figure 4.20 : The Structure of Relationship between the Components. ............................ 130
Figure 4.21: The JADE Organisation Architecture.................................................... 131
Figure 4.22: Reuse off-the-shelf Agent Environment.................................................... 133
Figure 4.23: UML Class Diagram to Specify Agent Behaviour............................................ 140
Figure 4.24: The Inheritance of Agent Abstract Class.................................................... 140
Figure 4.25: Assigning an Agent Role with Simplebehaviour ............................................ 141
Figure 5.1: The Filtering System Goal ....................................................................148
Figure 5.2: The Filtering Agents System Context Diagram. ..................................................149
Figure 5.3: Event User Enquiry ....................................................................151
Figure 5.4: Event User Feedback....................................................................151

Figure 5.5 : Event TimeTable .......... 152
Figure 5.6: The Filtering Agent System Use Case. .......... 153
Figure 5.7: Assigning Roles to Agent .......... 155
Figure 5.8: Agent System Architecture. .......... 157
Figure 5.9: Agent Interactions .......... 158
Figure 5.10: The Filtering Agent System Activity Diagram (the user initiator) .......... 159
Figure 5.11: Activity Diagram from Clock Event (scheduler initatior) .......... 160
Figure 5.12: The Filtering Agent System Plan .......... 161
Figure 5.13 Enquirer User Interface .......... 162
Figure 5.14: Feedbacker User Interface .......... 162
Figure 5.15: User Agent Model .......... 163
Figure 5.16: The UserAgent State Model Diagram .......... 165
Figure 5.17: UserModel Agent Model .......... 165
Figure 5.18: Internal State Diagram of UserModelAgent .......... 166
Figure 5.19: SearchAgent Model .......... 167
Figure 5.20: SearchAgent State Diagram .......... 167
Figure 5.21:Document Agent .......... 168
Figure 5.22:Document Agent State Diagram .......... 169
Figure 5.23: Indexing Process .......... 169
Figure 5.24: FilterAgent Diagram .......... 170
Figure 5.25: FilterAgent State Diagram .......... 171
Figure 5.26:ScheduleAgent Diagram .......... 173
Figure 5.27: Internal Design of a ScheduleAgent. .......... 172
Figure 5.28: UserAgent Class Diagram. .......... 179
Figure 5.29: The Diabetics Consultation System- Context Diagram. .......... 183
Figure 5.30: The Diabetic Consultation System Goal .......... 184
Figure 5.31: Patient Use Case .......... 187
Figure 5.32:Doctor Use Case .......... 187
Figure 5.33: The Diabetic Consultation Use Cases .......... 188
Figure 5.34: Assigning Roles to Agent .......... 190
Figure 5.35: Agent System Architecture. .......... 192
Figure 5.36: Agent Interactions .......... 195
Figure 5.37: The Diabetic Consultant System Plan .......... 196
Figure 5.38 Patient Enquiry .......... 197
Figure 5.39: Patient Set timetable .......... 197
Figure 5.40 : PatientAgent Data Model .......... 198
Figure 5.41: The PatientAgent Plan model .......... 199
Figure 5.42: Patient Treatment of Doctor User Interface .......... 200
Figure 5.43: DoctorAgent Model .......... 200
Figure 5.44: DoctorAgent State Diagram .......... 201
Figure 5.45: ExpertAgent Model .......... 202
Figure 5.46: ExpertAgent State Diagram .......... 203
Figure 5.47:ReminderAgent Model .......... 204
Figure 5.48: Reminder Agent State Diagram .......... 205
Figure 5.49: StockAgent Model .......... 206
Figure 5.50: StockAgent State Diagram .......... 206
Figure 5.51: Diabetic Consultation Execution architecture .......... 207
Figure 5.52 :Patient Agent Class Diagram. .......... 212
Figure 6.1 :Evaluation Process .......... 217
Figure 6.2 :Comparison of the Stages of Agent Development Process. .......... 230

# Tables

Table 2.1: Analysis vs. Design........27
Table 2.2: Software Abstraction........32
Table 2.3: Four aspects of Software architecture [Soni+95]. ........34
Table 2.4: Characteristics of Agent-oriented Developments........41
Table 3.1: The Agent Environment Abstraction of Filtering application and NewsFilter —-Application........61
Table 4.1: Event Scenario........97
Table 4.2: Agent Architecture Entities to Present Agent Behaviour........120
Table 4.3: The Principle of Providing the Physical Environment........122
Table 4.4: Agent Environment Characteristics........123
Table 4.5: The JADE Functional Requirement........124
Table 4.6: Example of Comparison Criteria of the Existing Agent Frameworks........134
Table 5.1: Event User Enquiry........149
Table 5.2: Event User Feedback........150
Table 5.3: Event Time Scheduler........150
Table 5.4: Agent Abstraction........156
Table 5.5: The Schedule Timetable........172
Table 5.6: The Comparison analysis for Filtering agent environment........177
Table 5.7: Event User Enquiry........185
Table 5.8: Event Time Scheduler........186
Table 5.9: Agent Abstraction........191
Table 5.10: The Comparison analysis for Diabetic Consultation Agent Environment........210
Table 6.1: Comparison (stages) between the Propose of Our Agent Method with the Current AOMs........232
Table 6.2: Afsaneh Personal Evaluation........243
Table 6.3: Saadat Personal Evaluation........243
Table 6.4: Comparison Result among the Methodology according Information System Perspective........248

# CHAPTER 1

# 1.Introduction

## 1.0. MOTIVATION OF THE RESEARCH

Software applications have been developed for some 50 years and their complexity has gradually increased in parallel with the complexity of the problem domains in which software is required. In the early 1970s, the problem domain of software was to perform largely routine tasks but often to facilitate the performance of large-scale tasks. Later, the complexity of software increased as it was needed to handle a massive volume of data, distributed, concurrent and so on. Software is used in increasingly complex problem domains.

A simple system solution may have several lines of implementation code, which are easy to understand, but a complex system consists of thousands of lines of code, so that it is difficult to understand and impossible to maintain without proper documentation of the process to develop the system. With complex systems, producing a systematic process for development of software applications becomes a necessity. The study of this area is called methods for software development[Hice+74].

Over decades, many methodologies been developed and introduced, specifically to overcome several kinds of difficulties for different problem domains and no methodology has been wholly successful in fulfilling its objectives, partly because computing is a highly dynamic field and the nature of the software domain problem is constantly changing in parallel with the invention of new technologies. Yesterday's solution will never completely solve today's problem.

Nevertheless, methodologies promises to increase the speed of development of software applications, reduce cost, facilitate easy maintenance and provide better quality[Jaya94]. Methodology can help to ensure that the user's requirements are met completely. They can promote communication between project participants, by defining essential participants and interactions, and can give a structure to the whole process and provide a standardisation of process and documentation[Simo+99].

[Graha+97] defines a methodology that has a 'tailorable life cycle model'. They define a methodology to consist of the following attributes: actions (techniques), process (life cycle) and representation(notation). [Jaya94] defines a methodology as comprising of a set of methods that illustrate the concept of a software solution. It is supported with techniques, tools and notation to show systematically how to develop a software application, and covers stages of the development process.

The main element of a methodology provides a method that describes a way of structuring the designer's thinking on how to solve a software problem. A method describes how to chunk the software problem into a hierarchy of sub-problems[Somm+97]. Despite many methodologies in use to develop applications for domain-specific problems[Somm+97], for our purpose three categories are sufficient to classify methodological approaches: structured, object-oriented and knowledge-based methods. These methodologies describe different design strategies or ways designers think about chunking the software problem into useful software application solutions. Moreover they indicate the abstraction employed.

The pioneering method introduced was the structured method[Your79][DeMa+78]. In structured methods, the designer thinks in terms of operation or functions to break down the software problem [Ash+90]. The executing system is made up from sequences of functions or operations using a data flow diagram. Later, this method was found to be inadequate to support several difficulties in system development. [Your89] is process-oriented while [Finkl89] is data-oriented, which can lead to difficulty in co-ordinating these two views of models. The later SSADM methodology [Ash+90] is undoubtedly very successful, but again it is only suitable to provide solutions for specific problem domains.

In later years, a new way of design thinking was introduced, so-called object technology. In this method, the designer thinks in terms of 'objects' instead of operations or functions[Wool+95][Mull96]. The system solution is shown as a result of interacting objects that maintain their own local state and provide operations on that state, resulting in benefits that have features of independence. Object-oriented design modelling uses object classes and relationships, using the object-oriented concepts of inheritance, encapsulation, polymorphism and aggregation[Booc94]. The object-oriented method has been accepted and used for development of various kinds of software in the industrial, commercial and academic domains such as Object Management Technology(OMT) [Rumb91].

The structured methodology in general remains tied to the waterfall life cycle [Royce70], while the object oriented approach is tied to an iterative cycle of development process, such as the spiral model[Boeh88]. On the other hand, the knowledge-based method was introduced for specific kinds of complex problem domain. It requires designers to think in terms of knowledge and relationships. There are several perspective categories of knowledge-based design strategy, such as rule based, case based reasoning, neural network and so on [Rusel+95] and knowledge-based approach ties in with a rapid-prototyping development process.

Based on the above discussion, we can generalise that new methods are developed associated with new technology is implemented.

Today's software applications are used to develop complex problem domains with high independent and dependent attributes that run under a complex technology. Today applications are also extensively networked and distributed, and operate under multiple jurisdictions each with their own mandate and prerogatives. Thus, the domain of software problems consists of the elements of distribution of data, decision and location, intelligence, concurrency, complex rules, high independence and dependence, and self-organisation, thus increasing the complexity of software solutions.

With such kinds of software problem domains, in the early 1990s, the term agent was introduced as a new paradigm for the design of software applications. An agent is simply another kind of software abstraction, an abstraction in the same way as functions and objects. An agent may be software or an entity that presents itself with human capability, such as autonomy, reactivity, proactivity, social-ability and intelligence. The agent paradigm is believed to be appropriate to provide the above characteristics of software problem domains. Agents present a high level of abstraction of design solutions. The agent paradigm lets designers think in terms of agents rather than objects or functions. The agent exhibits presents high dependency compared with an object-oriented approach. Such a software application needs an appropriate software development method. Existing software development methods are not suitable for the development of agent systems.

Why are existing methods not suitable for development agent system? Is the agent a totally new paradigm with a new software development method or does it adapt existing methods? If so, which method is most appropriately adapted? How do other methods play a role in the agent development method? Which agent paradigm is the most appropriate to be adopted? When and how should it be adopted? Can the existing development life-cycle be adapted? What modelling and modelling processes are suitable for an agent based system? These other questions will be answered in the following chapters.

The term 'agent paradigm' has been exploited to develop several types of complex applications in an ad hoc fashion using their different agent paradigms, exploiting existing methods and existing languages. Even though some agent methodologies have been proposed, they are tied to a specific perspective, described at a very high level of design, with an unclear process and only a partial indication of the development process.

The need for an agent-oriented methodology is apparent. Our research aims to contribute to the area of development of agent-oriented methodology. The research is focused on the use of the software abstraction concept to produce the various aspects of development of an agent-oriented methodology, such as defining the set of concepts used to illustrate the agent paradigm, defining the life-cycle of agent-oriented development, which consists of several stages of the development process, identifying suitable techniques and modelling to describe the various level of abstractions. The overall aim is to show a systematic process for development of an agent-based system and guidelines to development agent systems. However, in terms of modelling processes we focus on transformational modelling between analysis and design.

The methodology proposed follows the traditional software development method that uses several levels of abstraction to present a systematic process of development system (so called software abstraction). This is because agent abstractions and models offer expressiveness and flexibility that conventional notation lacks; it omits details throughout the development of a system from a high-level to lower level [Triem99][Shaw+96]. At least three software abstraction levels are used: requirements, design and implementation [Triem99]. Today's increasingly fast-paced and fluid demands in software engineering suggest that agent abstraction could be useful for supporting the agent development process in general.

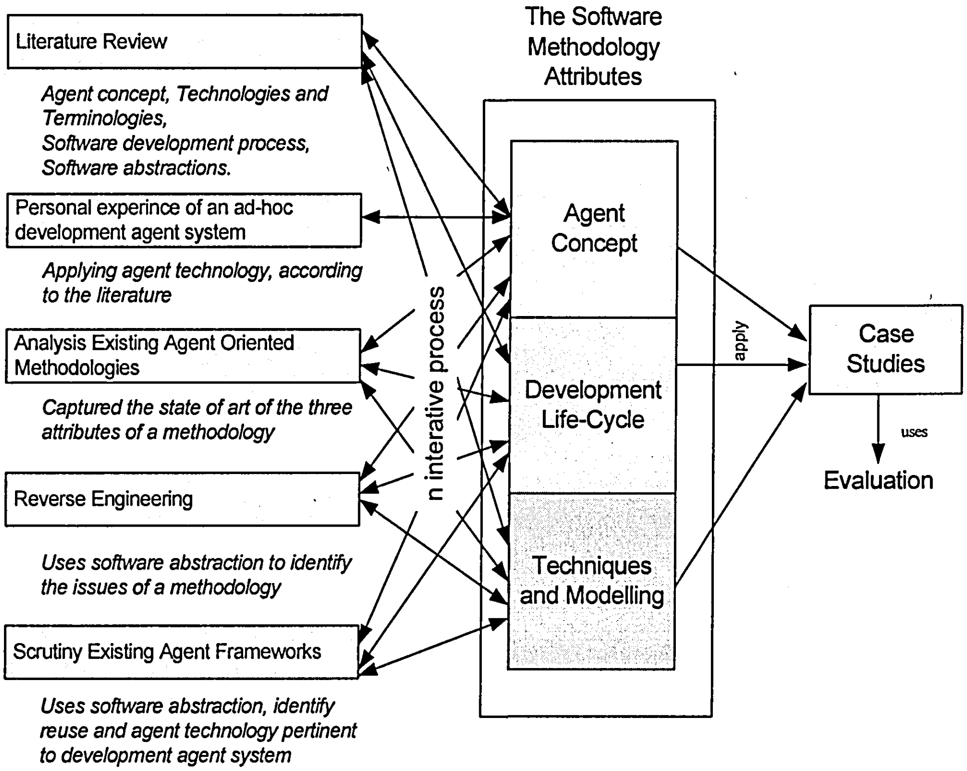

Literature Review
Agent concept, Technologies and Terminologies,
Software development process,
Software abstractions.
Personal experince of an ad-hoc development agent system
Applying agent technology, according to the literature
Analysis Existing Agent Oriented Methodologies
Captured the state of art of the three attributes of a methodology
Reverse Engineering
Uses software abstraction to identify the issues of a methodology
Scrutiny Existing Agent Frameworks
Uses software abstraction, identify reuse and agent technology pertinent to development agent system
n interative process
The Software Methodology Attributes
Agent Concept
Development Life-Cycle
Techniques and Modelling
apply
Case Studies
uses
Evaluation

Figure 1.1: An Overview of the Research Process

In this research is conducted through various combinations of activities to identify attributes pertinent to the attributes of a methodology as stated by [Graha+97][Jaya94] above: a set of concepts to describe an agent-oriented solution, the life-cycle including the stages of development process, and techniques and modelling notation in each stage. Figure 1.1 shows the integration of the research activities that leads to the research contribution in the area of development of an agent-oriented methodology. The figure shows that the research is conducted through the following activities: literature reviews about agents, software development and software abstraction including the current practice of agent development; experience in implementation of agent systems in an ad hoc fashion; analysis of existing agent-oriented methodologies; analysis of existing agent frameworks. The figure shows an '*n*' iterative processes of activities for identifying the issues and attributes of agent-oriented methodologies. The reverse engineering and scrutiny process are the core activities in identifying appropriate concepts, stages and modelling. The applicability of the proposal methodology is demonstrated in two case studies: a Filtering agent system and a Diabetic agent system. The proposed of the methodology is evaluated at three stages. First, evaluate according to the common criteria of software development methodology. Second, comparative analysis with the current agent-oriented methodologies developed in parallel with this research and the last evaluation is according to the criteria of methodology in Information System perspective.

## 1.1. THE CONTRIBUTION MADE IN THE RESEARCH PROJECT

Our research goal is to propose an agent-oriented methodology. When the research started 1998, the agent paradigm was in its infancy; the concept of agents was not clearly identified. We believe that our research makes a significant contribution in the area of development of an agent-oriented methodology as stated below.

- It shows how software abstraction is used in identifying software development process.
- The abstraction process is able to define a set of agent concepts to illustrate the agent paradigm.
- The comparative analysis of the existing agent-oriented methodologies is able to show that other agent concepts, techniques, tools (modelling) and stages may be suitable in particular domains of agent system.
- The future of the agent-oriented development life-cycle is explored, illustrating the framework of the whole process of agent analysis, design, implementation and applicability to the specific agent paradigm of agent development. The description of stages is applicable to most types of agent system.
- The thesis presents suitable techniques for analysis and design modelling.
- Two concrete case studies of the agent development process are presented.

Part of the subject matters addressed in this thesis has been published in the following conference proceedings and technical reports:

[Othm99][Othm00][Othm+02a][Othm+02b][Othm+02c][Othm+03] as shown in the bibliography. The first two are technical reports on reverse engineering of the Filtering application and NewsFilter application. The others were published in conference proceedings. Those papers describe the case studies presented in this thesis.

## 1.2. TOUR OF THE THESIS

The chapters in this thesis are structured in such a way as to show the process of development of the proposed agent-oriented methodology. The tour throughout the chapters will provide understanding of the development of an agent-based system.

**Chapter 2- Background Literature Review**

The literature review aims to provide basic knowledge about the development of an agent-oriented methodology. It is divided into three parts. The first part is the literature review on the concept of agent, agent technology, the terminology and the domain of agent based systems. The second part reviews the principles of software development methodology that can be used to identify the criteria and scope of our agent-oriented methodology. This entails a comparison of the existing software development methodologies, which leading to the identification of an appropriate approach suitable to be extended to present the agent paradigm. The third part discusses the concept of abstraction and several types of software abstraction tools used for the development of complex software applications.

**Chapter 3- The Building Block Towards a Process for Agent System Development**

The chapter is divided into two parts. The first part shows the derivation process towards the development of the proposed agent-oriented methodology and the second parts presents the essence of result of the derivation process. The derivation process consists of four analytical activities, which performed based on the syntheses of the current literature review on agent and software development methodology. The four activities are examining the potential software abstraction for agent system development, analysis of existing agent-oriented methodologies, reverse engineering of agent systems and scrutiny of the existing agent frameworks. The essence of the derivation results divided into the three attributes of a software development methodology. The first presents the analysis and design concepts for development agent system. The second shows the overview of development process of an agent system and the third presents the appropriate modelling techniques was captured. The modelling techniques are divided into two parts: development agent system and development agent environment.

### Chapter 4- A Method to Develop an Agent-Based System

This chapter aims to gather the agent concepts and techniques used, to present a proposal for an agent-oriented methodology. It is an expansion of the overview of the methodology presented in chapter three. In this chapter we demonstrate the life cycle of the development process from requirements to implementation, and show how agent modelling is performed throughout the stages. However, the agent modelling process presented here focuses only on the first part of design of the agent system. The development of an agent environment uses the component-based method adopted in the object-oriented framework approach and the development of the agent environment life-cycle adopts the software architecture development process. The software abstraction tools such as architecture pattern and design patterns, as presented in chapter four, are shown to be useful tools in developing the agent environment. In addition, we provide the criteria to analyse the reusable agent environment that can be deployed to implement the agent system. At this stage we can show the detailed design of the agent system.

### Chapter 5- Development of Agent System -The Case Studies

The aim of this chapter is to show how the methodology presented in the previous chapter is used, through the presentation of two case studies. The case studies present the deliverable of documents at each stage. The two case studies are a Filtering agent system and a Diabetics Consultation system. The agent environment reference architecture presented in chapter five is used to analyse the reusability of the agent environment.

### Chapter 6- Evaluation and Conclusion

The evaluation of the proposal of an agent-oriented methodology is performed in three stages. The first is evaluated based on common criteria of software development methodology. This method is able to view the progress made so far in the proposal of an agent-oriented methodology. The second evaluates the methodology by comparison with parallel research in the same areas such as MESSAGE, TROPOS, ODAC and PROMETHEUS methodologies. Thus, we can see the contribution made in the area of agent-oriented methodology. The next stage is to evaluate the methodology according to the Information System Methodology perspective, an expectation of what a complete agent-oriented methodology should be. Based on this, we discuss the level of completeness of the agent-oriented methodologies that the research community has achieved so far, and address possible future research directions in this area.

# CHAPTER 2

# 2.Background Literature Review

## 2.0. INTRODUCTION

This chapter presents the three areas of background to the study. The first section focuses on the concept of an agent, agent technologies and terminology pertinent to the development of an agent methodology. The second focus of discussion is literature on software development methodologies; their meaning is reviewed and several software development approaches are used in development of software applications are presented. The third topic considered the existence of software abstraction tools, which are commonly used for development of a software application. Each area is discussed in more detail in the following sub-sections in order to establish the theoretical context of the present research, which is based on the integration of these three areas.

## 2.1. AGENTS

Agents present a new paradigm for the current and next generation of system development solutions because they are is flexible[1], and bring intelligence[2] and autonomy[Wool+01][Jenn+98a]. Agent technology arises from a blend of the following technologies: network communication [Harr93], artificial intelligence [Wins79], multi-agent systems[Weis99], distributed problem solving[Less87], distributed systems[Mull93], concurrency [Rank94], knowledge engineering[Adel90], distributed artificial intelligence[Bond+88] and object-oriented technology[Booc94]. A problem resulting from the inheritance of these technologies is a multiplicity of agent definitions, stemming from different communities, and reflecting their different perspectives. With such technology, in practice agents have been used to address the diversity and complexity of many real world problems of software applications [Brad97a]. In earlier research, agent technologies were influenced by Artificial Intelligence [Rusel+95] and Distributed Artificial Intelligence leading to notions of strong and weak agents respectively[Treu+98][Wool+95]. This also leads to multiplicity of

[1] Various combinations of agent characteristics to present agent and can be located any where.
[2] Wooldrige in his book[Wool01] define reactive as intelligence, pg 23.

agent definition. Clearly, agents are not expert system[3] nor DAI[Alan88][Mamd98][Shen97] [4] but as Etzioni states, agent based systems are 99% computer science and 1% Artificial Intelligent [Etzi96].

### *2.1.1. Characterising Agents*

Despite various definitions of an agent, we will agree with [Wool+95] who takes a simplistic view, namely, an agent is *one who acts*. Therefore, an agent may represent a variety of things such as a person, a machine or a piece of software. However, this view is too general and not suitable for presenting an agent paradigm for software development. As a first step towards that goal, we concur with [Wool+95][Wool+95a] and consider an agent as consisting of a combination of the following properties:

*Autonomous*: able to act without direct external intervention. The agent has some degree of control over its internal state and action based on its own experiences [Maes90].

*Reactive*: Agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it [Wool+95].

*Interactive:* able to communicate with other agents and its environment [Wool+95].

*Sociable:* capable of two-way conversation, in which one party asks questions and the other verifies or answers. The conversation allows inter-agent co-operation which enables agents to exchange their knowledge, beliefs and plans and to work together to solve large problems which are beyond an individual's capabilities[Wool+95].

*Proactive*: having a goal-oriented ability or a purpose. The agent does not simply react to the environment[Maes90].

*Proxy*: able to act on behalf of someone or something, that is, acting in the interest of, as a representative of, or for benefit of some entity[Maes90].

*Co-ordinative*: able to perform some activity in a shared environment with other agents. Usually this activity consists of a well devised plan and workflows, including some other process management mechanism[Geor+95][Jenn93].

*Co-operative*: able to co-ordinate with other agents to achieve a common purpose (some scholars define this as collaboration)[Weih+95][Krau97].

*Competitive*: able to co-ordinate with other agents, such that the success of one agent implies the failure of another. This feature contradicts co-operation[Bacl92].

[3] Expert system does not have environment which they act but act through middleman, does not have reactive or pro-active and do not ability to socialise (cooperation ,coordination and negotiation)

*Intelligent*: able to state formalised knowledge such as beliefs, goals and assumptions and to interact with other agents using symbolic language[Wool+95].
*Mobility* - agents can move around in their environment from one place to another and carry data along with intelligent instructions and execute remotely[App+94].
*Risk and trust*- address the issue of delegation of tasks with at least a reasonable assurance that the user wants to delegate the task, and that the agent can bring back the result the user wanted[Jenn+98].
*Rational* - able to choose an action based on internal goals and the knowledge that a particular action will bring it closer to its goals[Mamd98].

There are many more agent characteristics. Up to now a standard definition has not been established to describe an agent. Jenning said agent must, at least, be an autonomous entity that can interact with its environment [Jenn+98]. But, based on the various definitions and the realisation of their usefulness for agent oriented development as we agreed with Etzioni[Etzi96] and Wooldridge[Wool+95]. An agent should possess at least the first two of the above properties to give a degree of usefulness to the agent paradigm in developing software applications. The rich combinations of these behaviours can generate various degrees of complexity of agent systems, with attributes in which an agent is able to keep observing changes in its environment, as well as provide features of reasoning, capability to act rationally, be pro-active, learn, adapt, etc. However, Kendal believes agents should be provided with beliefs, desire and intention characteristics (BDI agent)[Kend+95].

The combinations of these properties have led to the emergence of several types of agent systems: mobile agents, interface agents, reactive agents, autonomous agents, interface agents, interactive agents, information agents, intelligent agents, entertainment agents, wrapper agents, hybrid agents and collaborative agents[Brad97a][Jenn+98]. From detailed examination of these types of agent systems, agents can be identified in other forms such as broker agent, mediator agent, facilitator agent, services agent, etc. These agents play a role in the system to be developed.

Agents can be put into two categories: single agent systems and multi-agent systems(MAS)[Treu+98][Weis99]. A commercial agent system such as information search is a single agent. The agent is launched in order to return appropriate information as needed by the requester. The single agent method is unsuitable in large scale problem domains. It does not

[4] DAI only focus with distributed static intelligent component (without autonomous), limited communicates with share common goal and do not presented most criteria mentioned in footnote no 1.

reduce the complexity of the system, nor does it utilised the various agent characteristies stated earlier. The next section discusses in further detail the issues associated to agent technology.

There are various types of multi-agent system, for example, a Travelling multi-agent system consisting of various travel agents that negotiates a good price of a ticket. Such situations can be viewed in terms of agents needing co-ordination either through co-operation, competition, or a combination of both. Another speculative example involves various agents handling a household system comprising an alarm clock, house heater, coffee maker and toaster. When the alarm clock rings in the morning, as set by the owner, it co-operates with the house heater to indicate when to start the heater. At the same time, the heater co-operates with the environment to identify the right temperature of the heater. Based on the owner's preferences for the time of having a bath and dressing, the alarm also co-operates with the coffee maker and toaster, indicating when the coffee and toast should be ready. The coffee maker and toaster negotiate with each other with the aim that the toast and coffee should be ready at the same time or that the difference between the time of coffee and toast being ready should be minimal.

These two examples represent different types of multi-agent system environment that allow the agents to co-operate with each other. In the first example, the multi-agent system environment may consist of at least two layers of multi-agent environment: a software agent to present the user agent, tourist agent and travel agent, and networking is needed to allow the agents to communicate with each other. In contrast, in the second example, the appliances themselves are the agents. It may only need a physical interconnection among the agents to allow the appliances to co-operate with each other. Based upon these two examples we can see that the development of an agent system consists of two parts: design of an agent system to achieve the system goal, and identification of an appropriate agent environment in which the agent can be exhibited.

From these examples, we can see that communication is the main issue in a multi-agent system, irrespective of agent behaviour. It allows an agent to send messages to other agents and the receiver agents are able to reply to the message to enable agent interaction. Such basic interaction has an influence on several methods of agent socialisation such as negotiation, co-operation and co-ordination.

Franklin classifies co-operative multi-agent systems into two types[Frank+97]: communicative and non-communicative agents. But our research is focuses on communicative multi-agent systems.

A real autonomous multi-agent system is likely to become a future paradigm of agent systems, but now it is very much at an initial research stage [Mller+97][Jenn+98][Jenn+98a]. The recent research has narrowed down the term into a more general term; instead of a real autonomous multi-agent system, the paradigm is one of a multi-agent system composed of several autonomous components that are able to autonomously interact. Thus, this paradigm can be characterised as follows: each agent has the ability to solve a problem with its own expertise; there is no global system control, data is decentralised and computation is asynchronous.

Through the discussion of the characteristics of a multi-agent system, we are able to identify the key issues representing agent technology as discussed in the next section.

### *2.1.2. Key Issues for Agent Technology*

According to the previous discussion we abstract three key issues as necessary for agent technology. The first relates to the nature of the agent; the second relates to agent communication and the third relates to agent management. Each of these issues discussed in the following sub-sections.

#### 2.1.2.1. Nature of Agent

An illustration of an archetype of a multi-agent system can be seen in Figure 2.1. The figure also shows the features of a multi-agent system environment [Zam+00]. From the figure, it can be seen that a multi-agent system consists of several agents, which interact with each other. The figure shows that Agent1 interacts with Agent2 and Agent2 interacts with Agent3. Interaction protocol describes how these agents communicate (such as negotiate, requesting, biding). Interaction medium describes the medium by which agent interacts, e.g. HTTP, SMTP.

The issue of openness plays an important role in multi-agent systems. The openness characteristic can be presented on three levels: the high level is what we describe as agent communication language, based on speech act languages. The content of information is then transferred into different types of representation, which are understood by the middle level of the communication operating system, which follows the ISO standard communication. The lower level presents the actual physical connection between agents.

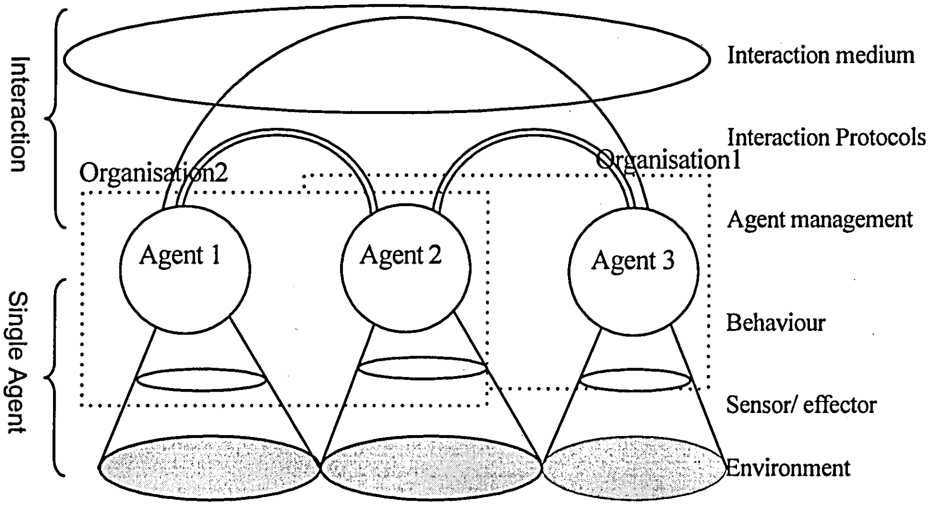


Figure 2.1: Characterisation of a Multi-agent System

At the same time, each agent may communicate with its own environment or external world, which may influence the interaction between agents. Each agent is also provided with certain types of behaviours that represented as roles or tasks. Agent management is needed to allow agents to socialise. Agent1 is registered in Organisation2, Agent3 is registered in Organisation1, while Agent2 is registered in both organisations. Even though Agent1 and Agent3 are registered in different organisations that they can interact.

Figure 2.2 shows an example of the internal architecture of an individual agent that interacts within agent components, external world and processor, and other agents as shown with dotted line, straight line and thick line respectively.

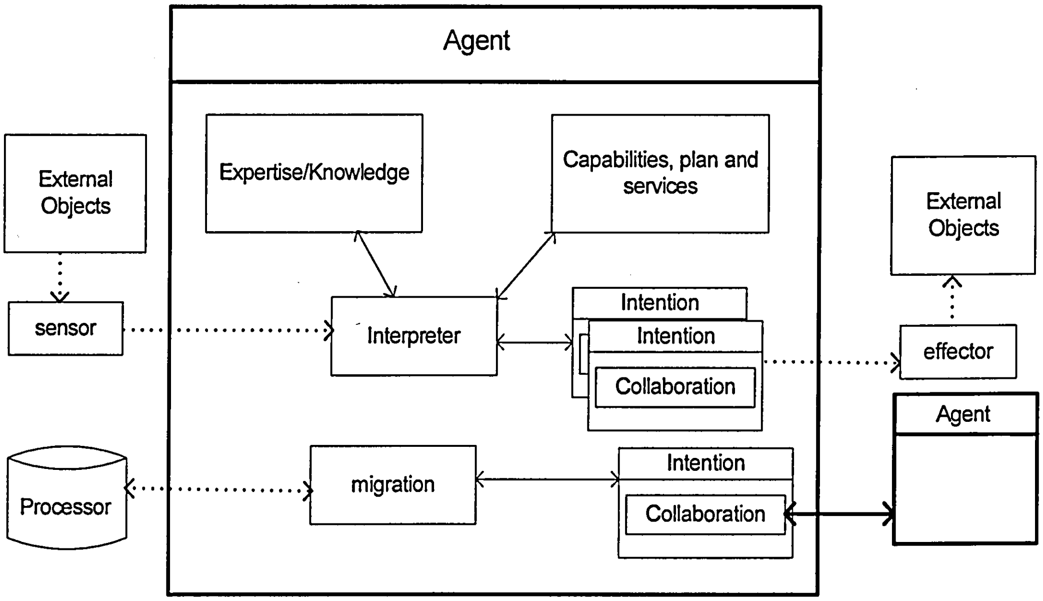


Figure 2.2: Components of an Agent Architecture

An agent uses a sensor or effector to interact with the external world and an agent interacts with other agents based on its intention. An intention is an action that leads to communication. The use of the term intentions reflects the fact that the action is not fixed, but rather may change according to several factors.

A weak agent consists, at least, of autonomous behaviour and ability to interact with other agents through its intention, a strong agent is able to carry out reasoning through its expertise, and select plans capable of achieving its goals. A simple agent presents itself with beliefs using a passive object that represents the agent's knowledge. A strong agent has the capability to retrieve a plan from a library. These plans are invoked when a triggering event occurs, according to the agents interactions. Agent intentions are designed in such a way as to occur concurrently. Figure 2.2 shows three types of intentions: involving interactions with external objects, collaboration with other agents and migration with other processors[Kend+95]. As indicated in the figure, the co-operation or negotiation agents are based on agent collaboration.

The main issue in relation to the agent is identifying the components or elements to represent agent architecture. Agent architecture shows in Figure 2.2 consists of the following elements: expertise, plan, intention and interpreter, while the external elements communicate with other agents, external objects and the processor but these components may or may not be necessary used to describe agents. An agent may consist of other kinds of components as well. In our research we use these components to illustrate agents.

**Agent Communication Issues**

Based upon the description of an agent above, two types of agent communications are identified: first, communication with external objects including human-user, legacy systems, files, etc., so called *local communication,* and second, communication between other agents, named *inter-agent communication.* Inter-agent communication focuses on a multi-agent system. Besides these, there is a special type of inter-communication, is categorised as inter-agent communication but differing in the source of inter-communication. The former communication focuses on an agent, while this communication focuses on communication as mobile agents. The two communication types has shown diagrammatically in Figure 2.1 and Figure 2.2.

Local communication is not a new issue in agent technology. It uses the existing communication technology through sensors. For example, it uses a 'socket' to communicate with the Web.

Mobile communication focus on mobile agents. The mobile agent is a special case of agent communication[Whit+97]. In order for agents to communicate, they have to move to a different physical location to negotiate, for instance, rather than sending a message through a communication channel. Therefore, mobile agents are special cases of implementation of an agent system. The difference is the agent environment level rather than in the design of the agent system itself. Inter-agent communication is the major issue in presenting agent technology. Agent technology has initiated new communication techniques, which differentiate between traditional applications and agent applications. In the next section, our discussion focuses on inter-agent communication.

### 2.1.2.2. Inter-Agent Communication

The main concept of inter-agent communication of heterogeneous agents can be described diagrammatically in Figure 2.3 below. The figure shows that discussion of inter-agent communication requires consideration of two levels of inter-agent communication: logical inter-agent communication and physical inter-agent communication.

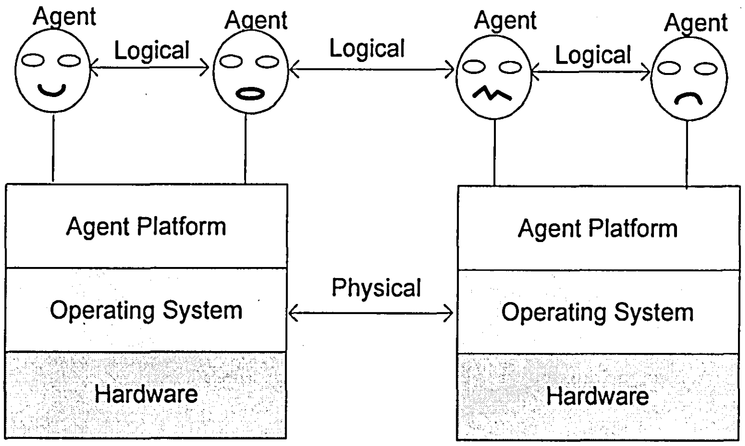


Figure 2.3: Inter-agent Communication between Two Heterogeneous Agents

The logical inter-agent communication aims to allow the heterogeneous agents to communicate with each other (interoperability). In the context of agent technology, the issues of inter-agent heterogeneity is deal with agent communication language[5]. In contrast, physical inter-agent communication is focused on provision of a communication medium that allows agents in different physical locations to communicate with each other (interoperability). Physical inter-agent communication is focused on providing mechanism that allows agents to communicate openly.

### Agent Communication Language

According to [Gene97] there are two approaches to designing an agent communication language(ACL), the procedural approach and the declarative approach. Communication in the procedural approach is based on executable content that can be accomplished using a programming language such as Java [Arno+98] or Tcl [Obje+98]. However, this approach has limitations based on the difficulty of controlling the executable content, co-ordination and merging. Recent research suggests using a declarative, which is based on elocutionary acts such as request, send, command, query, etc[Fini+93]. The two most popular declarative languages are KQML(Knowledge Query and Manipulation Language)[6] and FIPA( Foundation for Intelligent Agents)[7].

Both ACLs demonstrate three key elements of ACL[Fini+97][Huhn+98][Fini+98] [FIPA+97b]:

- a common agent communication language and its protocol,
- a common format content of communication and
- a shared ontology.

The common ACL protocol is known as *performatives* using speech act languages such as send, accept, reject, etc. that mimic human of communication. Therefore the use of an ACL system development provides a certain level of behaviour.

The common format content of communication and shared ontology exhibits the openness of agent interaction. The common format content of communication describes language for interpreting the information in the content field of the message, called 'ontolingua', whereas *ontology* identifies the ontology to interpret the information in the content field of the message. The content field provides the exact content of information to send or receive.

*Ontologies* are defined as specification schemes for describing concepts and their relationships in a domain of discourse [Fini+97][Guar+94][Guar+95]. It is important not only that agents have ontologies to conceptualise a domain, but also that they have ontologies with similar constructions. Such ontologies, when they exist, are called common ontologies. Once interacting agents have committed to a common ontology, it is expected that they will use this ontology to interpret communication interactions, thereby leading to mutual understanding and

[5] The ACL is from knowledge representation language[Gene+92] of AI research.

[6]KQML is the earlier ACL developed part of a larger effort, the ARPA Knowledge Sharing Effort which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable[Fini+93] [Gene+92],

(ultimately) to predictable behaviours. Ontolingua [Grub+93] is often mentioned in the literature as a system that provides a vocabulary for the definition of reusable, portable and shareable ontologies. Ontolingua definitions are described using syntax and semantics similar to those of the Knowledge Interchange Format [Gins+91], also known as KIF, which is a format to standardise knowledge representation schemes based on first-order logic. KIF is the language used to interpret the KQML ontology whereas FIPA uses the semantic language's ontolingua[Holt+82] , i.e. SL0 or SL1 and it is based on FIPA ontology [FIPA+97b]. This means agents using interaction in KQML and FIPA do not have a similar ontology, due to their different ontolingua. Details on FIPA can be found in [FIPA+97b].

The semantic language allows one to represent a belief, desire or uncertain belief of an agent as well as action that agents perform. It is used to define a constraint that the sender of the message must satisfy. For example [FIPA+97b]:

The semantics of the request are as follows:

$< i, request(j, \alpha)>$

*feasibility precondition:* $B_i Agent(\alpha, j) \wedge B_i I_j Done(\alpha)$

*rational effect :* $Done(\alpha)$

The SL defines B*i* as statement of believes of an *i* condition, *Agent(α,j)* means that the agent of action α is *j* (i.e. *j* is the agent who performs α ), $I_j$ as statement of intention *j* and Done(α) means that the action α has been done. Thus agent *i* is requesting agent *j* to perform action α under assumptions that agent *i* believes that the agent of α is *j* (the message is right) and agent *i* believes that agent *j* does not currently intend that α is done. Figure 2.4 shows the example of use KQML, while Figure 2.5 shows the example of use FIPA[8].

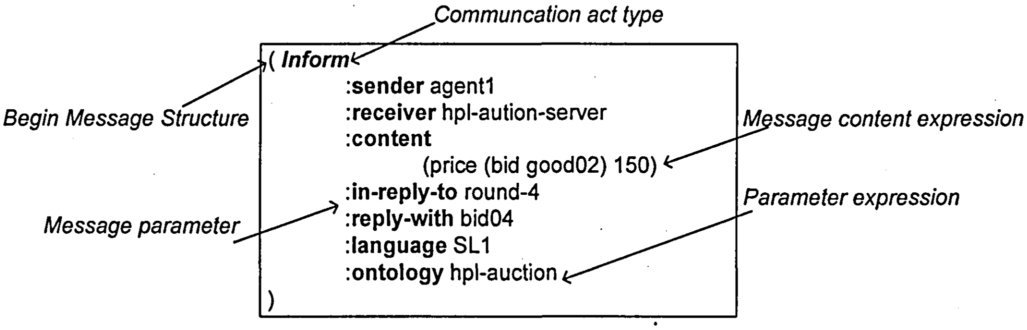
```
(register
    :sender      agentA
    :receiver    agentB
    :reply-with  message2
    :language    common_language
    :ontology    common_ontology
    :content     "(ServiceProvision Manufacturing:TaskDecomposition)" )
```

Figure 2.4:. Example of a KQML message.

[7] FIPA was formed in 1996 to produce software standards for heterogeneous and interacting agents and agent-based systems. It collaboration of various memberships[MEM02] to produce a standard ACL[FIPA99].

[8] Detail description of examples describe in FIPA Specification no FIPA00018[FIPA00].

Both figures show the similarity of using ACLs. FIPA is based on KQML [FIPA+97b], therefore they have similar syntax languages, forms of message and message parameters. However, FIPA agent communication language tied to FIPA architecture[9]. FIPA has additional functionalities such as agent management. Agents are able to register, un-register, recommend, recruit, broker and advertise, therefore Figure 2.5 do not use have 'register' command compare to KQML. With this functionality, FIPA is restricted to agent management components in the architecture, whereas KQML is not restricted to any architecture.



```
( Inform
        :sender agent1
        :receiver hpl-aution-server
        :content
                (price (bid good02) 150)
        :in-reply-to round-4
        :reply-with bid04
        :language SL1
        :ontology hpl-auction
)
```

Figure 2.5: Example of FIPA language

The differences mentioned above lead to incompatibility in communicating agents using KQML and FIPA. Current research states that FIPA has become a standard for agent interaction[FIPA00]. Although KQML and FIPA are the main declarative languages, however, other research may use KIF or XML for interaction to present the openness of the internet environment[Amun02]. The use of ACL provides the basic communication for several types of socialising agents. The pattern of agent interaction is called the *interaction protocol*[10].

Up to now we have discussed agent communication at the logical level as a basis of agent socialisation. The next discussion is focused on physical inter-agent communication.

**Network Inter-agent Communication**

Physical inter-agent communication focuses on providing interoperability of inter-network communication. It aims for agent interoperability, interconnecting from multiple network resources at different locations. Inter-network communication is not a new technology for agent technology; rather, it an enhancement of the existing network communication technology. For the purpose of the agent paradigm, we can categorise the physical inter-agent communication into three layers: agent platform, operating system and hardware. The hardware communication is usually the lowest level of communication, such as the use of RS232 or broadband, etc. The

[9] FIPA ACL can also applied to any architecture that open to FIPA architecture.

[10] There are more then dozen types of interaction protocol have been defined by FIPA group[FIPA00]

operating system level uses a standard communication protocol. Telecommunication network management usually use the ISO (International Standardisation Organisation) and OSI (Open System Interconnection) group. This network communication is needed for management.

On top of this, several middle-wares can be used for a specific purpose for agent communication. For example, CORBA and DCOM are used to provide interoperability of distributed heterogeneous agents. The middle-ware includes the agent communication management mechanism as well, which most include in the agent platform, the so-called message transport mechanism.

**Message Transportation Mechanism**

As previously stated, agent communication needs an agent environment that allows messages to be scheduled, as well as event driven. The transportation mechanism should be able to identify the unique agent address. At the same time, transportation mechanism should support other communication terms such as uni-cast, multi-cast and broadcast to support the socialising agent. These mechanisms are provided in middle-ware such as Common Object Request Broker Architecture [Corba99], Object Management Groups (OMG) message services, the Java messaging service, remote method invocation (RMI), Distributed Common Object Model (DCOM) and enterprise Java Bean Events services.

### 2.1.2.3. Agent Management

In the previous discussion, we focused on the message transport mechanism, whereas in this section we discuss agent management. The agent management provides mechanisms for advertising, finding, fusing, using, presenting, managing, and updating agent services and information[Wied+92]. To address these issues, the notion of middle agents [Deck+97] was proposed. Middle agents are entities to which other agents advertise their capabilities, and which are neither requesters nor providers from the standpoint of the transaction under consideration. The advantage of middle agents is that they allow a multi-agent system to operate robustly when confronted with agent appearance, disappearance, and mobility. There are several types of agents that fall under the definition of middle agents. Note that these types of agents, which are described below, are defined so vaguely that sometimes it is difficult to make a clear differentiation between them.

- Facilitators: agents to which other agents surrender their autonomy in exchange for the facilitator's services [Brad97a]. Facilitators can co-ordinate agents' activities and can satisfy requests on behalf of their subordinated agents.

- Mediators: agents that exploit encoded knowledge to create services for a higher level of applications [Wied+92].
- Brokers: agents that receive requests and perform actions using services from other agents in conjunction with their own resources [Deck+96].
- Matchmakers and yellow pages: agents that assist service requesters to find service provider agents based on advertised capabilities [Deck+96].
- Blackboards: repository agents that receive and hold requests for other agents to process [Nii89] [Cohen+94].

**Life Cycle Management**

An agent runs in its environment. Refering back to the kitchen appliance example, the toaster agent must always be plugged into the electric socket and turned on. Nevertheless, such agents also need a mechanism to regulate such activities as starting, stopping, waiting, being traced and removing. Mobile agents, for example, need other lifecycle issues such as permission to run, permission to perform certain tasks, and communication in different locations[Flor99].

The agent life cycle needs to be identified and managed. A variety of messaging techniques exist for this purpose. An event-style model is the most popular technique, guaranteeing message delivery. Another method is a more casual method, where an agent sees a message of interest to itself. A message may be sent based on a peer to peer approach, a store and forward approach, or a published and subscribed approach.

In distributed systems, an agent may move to another server, which changes its unique identification. In some situations, an agent may provide historical information on its action, so the agent can learn what decisions have been taken before and can use it to evaluate the prior actions.

**Agent Security, Identity and Policy**

Agent identity is becoming compulsory in agent systems, especially in multi-agent systems. An agent may be identified by its name, role or group. However at the lower level, the agent should be identified by the address where it is executed. In certain cases, an agent is identified by a logging name.

Security becomes the main issue in a distributed computing environment. Many types of security risks are considered when passing messages, which can be modified or destroyed.

Policy is a technique to reduce risk. It refers to how access to a valuable resource is controlled. For example, an agent is limited to a certain type of processing job.

The above discussion only presents the common issues in representing agent technology. However, there are more issues involves in agent technology such as mobile agent, real autonomous agent, strong agent versus weak agent, etc., which entails an additional set of to represent agent technology. However, we are not concerned here with mobile agents. Discussion of mobile development systems can be found in [Arid+98][Oshi+98] [Baum+97b][Baum+97c] [Baum+97d][Baum+98][Zam01].

The current method to represent agent technology is to use architecture. There are many different attributes to present agent architectures, such as FIPA architecture [FIPA+97a], BDI architecture [Geor+95] and Kasbah architecture [Chav+96]. In addition, there are various agent environment architectures used to represent agent technology such as KASBAH[Chav+96], ARCHON[Cock+96], JADE[Bell+99], ZEUS[Nwan+99] and FIPA-OS[FIPAOS00].

Taking into account the various issues and techniques used to present agent technology, we believe designers are free to build or chose appropriate architecture for particular domain of solution.[11]

An agent environment is an executable system that is used to implement an agent system. Differences in agent architectures may cause problems of communication between agents. For example, a FIPA agent may not communicate with KASBAH agent due to different architecture. Therefore, the criteria for choosing open agent architecture are important in agent development for future maintenance. For example JADE and FIPA-OS are developed based on FIPA architecture, so agents which are developed based on JADE can openly communicate with agents developed based on FIPA-OS. However, recent research has proved that agents can communicate even in different agent architecture [FIPA00]. For example, the ZUES agent can communicate with JADE agent in FIPA architecture. This means FIPA architecture presented as open architecture. Similarly JACK agent is based on BDI architecture, it can communicate with JADE agent in FIPA architecture. This means JACK is an open architecture.

### *2.1.3. Status of Agent Technology*

The literature review shows that agent technology moves forward by building upon existing technologies (e.g. relational, knowledge based and object-oriented). Agent technology is not a single technology or new technology but rather an integration of multiple technologies. Agent technology is not for a specific kind of application but rather for various kinds of applications which are suited to solutions with certain properties of agents. Agents can function as part of an operating system or an application environment. In other situations, agents may strengthen the human-computer interaction aspects of a system.

In practice, agents have been used in several kinds of complex software, for a wide range of applications: information filtering and retrieval, entertainment, air traffic control, network management and telecommunication, electronic commerce, management of business processes, groupware, manufacturing, and as personal agents[Bren+98][Treu+99][Janc96]. The nature of agent applications in these areas is described below:

1. Information filtering and retrieval.

   Agents are used in all the services needed to help users to find information they request more easily, quickly and accurately. Examples of such agent applications are Yahoo[Yahoo99], AltaVista[Alta99], MetaCrawler[Craw99], Google[Goog00]. They range from simple search engines like NewsWatcher [Watch99], WebCrawler[Wcraw01] and HotBot[Hotb00] to meta-search engines which provide learning behaviour, like Copernic[Coper99]. The use of agents in information filtering has reduced the user time in finding information through the Web [Smart00].
2. Entertainment.

   Agents are also used in entertainment systems such as games, cartoons, films and advertisements. For instance, in games, agents are used to provide autonomous interaction between game characters, the environment and multiple players. In film, a camera agent is used to provide automatic motion, focus and reaction, 3D graphical agents and avatars in computer animated features films, cartoons and advertisements [Smart00]. Other entertainment applications related to the Web are Lifestyle Finder [Smart00], FireFly[Fly98], Netradio and OpenSesame[Open99].
3. Air traffic control.

   In air traffic control systems, for example OASIS [Ljun+92], agents are used to represent both aircraft and the various air traffic control system loads with information and goals corresponding to real world aircraft.
4. Network management and telecommunication.

   In the network management and telecommunication application area, agent applications are used in network control[Flet94] and network services[Mage96].
5. Electronic commerce.

[11] Software architecture is a technique to design system architecture as discusses in later section.

In electronic commerce, applications involve configuration and delivery of user requested services at the right time, cost and quality of services. Among these types of applications are buying and selling services like BargainFinder, Bargain Bot, Fido and AdHound, Jango and Kasbah[Broke99].

6. Business process management.

   In the management of business processes, agents deal with the management of business tasks and resources, provision of services and carrying out business operations. Examples of this type of application are workflow management[Reic+98], financial services[Wen+98] and telecommuting[App+94].

7. Groupware.

   Groupware applications include computer conference systems such as e-mail, decision support systems and appointment scheduling programs [Bren+98]. Examples of groupware applications are Pleiades[Sycr95] and MAXIMS[Maes94].

8. Manufacturing.

   In manufacturing applications, agents are used due to the dynamic nature of manufacturing processes. Agents are used in industrial robots[Broo86] and factory automation [Bake96].

9. Personal agent.

   A personal agents system is one that uses agents to reduce the complexity of user tasks. There are several types of system categorised as personal agents. In email and news filtering systems, the used agents are focused on the interface for filtering. In personal schedule management and personal automatic secretary, the agent is used to work on behalf of the user by providing an ability to be aware of any changes that occur. In information filtering and retrieval, the agents are used for advising and focusing information using an intelligence learning capability. Examples of such applications are Letizia[Lieb95], Webdoggie[Webd99] and Copernic[Coper99].

Early applications of agent technology depended on the technology available at that time. Often this was not appropriate to the available technology and new paradigm needed to be sought. Some of the applications present the value of agent without applying the current means of agent technology as discusses in previous section.

We conclude that in order to develop agent applications, several complicated technologies are involved. Developing a new agent language for the development of an agent system is a long process. There are advantages in using the existing programming languages to present the agent technologies, since the existing languages are mature and established. However the problem with the use of existing programming languages is that they are associated with a specific paradigm. For example Pascal is associated with the structured approaches, Java and C languages are associated with the object-oriented paradigm and Prolog is based on the rule-based paradigm, none of these match the agent paradigm. An agent system represents a high level of design paradigm that can be seen as an extension of existing design paradigm.

To overcome this problem, the current practice is to develop a mediator language, which is able to present the agent paradigm. The mediator language is developed using existing development techniques. For example, AgentO is developed using script languages[Shoh93], while Aglets is developed based on Java[Agle98]. In practice, these mediator languages are called agent frameworks. Currently, there are more than a hundred agent frameworks that have been developed for commercial and academic purposes as partly shown in [Tools99]. Some of the agent frameworks are freeware. These agent frameworks are developed based on existing programming languages such as C++, Java and Script languages, but designed in such a way as to present an agent paradigm.

Despite the complexity of the development of agent systems, most of the agent systems reviewed are developed using agent frameworks as a language to implement the agent system. In addition, the literature review of the existing agent oriented approaches discussed in the next chapter shows the use of agent frameworks as a language to implement agents. However, the existing agent frameworks shown in [Tools99] are tied to specific architecture, this limits their use to specific domain of agent applications. For example, Kasbah[Chav96] and JATLITE[Jat97] were developed for internet agents; Grasshoper, Leap, Aglets are focused on mobile agent [Tools99]; JADE, Agentbuilder or Zeus are focused on multi-agent systems. The choice of the use of existing agent frameworks depends on the requirements of the agent system. Agent frameworks are limited in various kinds of requirement, such as specific communication channels, the use of a specific coordination or negotiation strategy, and the use of specific agent architectures such as decision making architecture, BDI architecture, etc. These examples of requirements are part of the functional requirements of agent systems which the existing agent oriented methodologies ignore. We see the development of agent frameworks as a part of the development of agent systems that is influenced by some aspects of the agent system requirements. Generally it focuses on the physical requirements and the choice of technology used to develop the agent, as we discuss in a detailed in a later chapter. In our view, the development of agent frameworks requires its own kind of software engineering, which we call development of an agent environment. This is because it focuses more on physical design rather than logical design.

Therefore, we believe that the development of an agent system should consist of:

- identifying an agent paradigm associated with the user's requirements;
- designing the agent system according to the agent paradigm;
- developing an agent environment (including designing appropriate architecture) to develop the agent system and
- implementing the designed agent system based on the agent environment.

## 2.2. SOFTWARE DEVELOPMENT METHODOLOGY

### *2.2.1. Principles of Software Development Methodology*

A methodology is one of the central topics in software engineering. The main role is to develop effective software application in the most efficient way. The Oxford Dictionary defines a methodology as the '*study of systematic methods of scientific research'*.

There are several definitions of methodology have been well established within the field of development software application. The following are some of the definitions of a methodology in terms of development software application.

[Hice+74] define a methodology as

> *'a recipe that enables an engineer to find a solution to a specific set of problems, consisting of a set of guidelines'.*

[Avis95] defines a methodology as

> *'a collection of procedures, techniques, tools, and documentation aids which will help the system developer in their effort to implement a new system'.*

A methodology should be sufficiently precise to enable any engineer to apply the recipe successfully to a suitable problem, while at the same time it should leave enough room for creativity.

According to [Hubm97], a methodology consists of the following components:

- *'A set of models that represent different aspects of the problem domain as well as the solution at different stages;*
- *A set of models that transform instances of one model into another model;*
- *A set of procedural guidelines that define an order for the systematic application of the methodology's steps;'*

On the other hand, those with a process model perspective, [Wyne93] assert that a methodology must have at least one complete phase (requirement or analysis or design) of system development, consisting of a set guidelines, activities, techniques and tools, based on a particular rationale of system development.

Based on the above definitions, a methodology that consists of a set of activities towards development of a software application is called system development life cycle. A well known, system development life cycle is called the waterfall model [Boeh+77]Boeh88]. This consists of the following stages: requirement analysis, design specification, coding, implementation and maintenance. [Wain+92] identify steps in the development life cycle, namely:

- Definition phase: feasibility analysis, requirement definition.
- Construction phase: system design, system building and system testing.
- Implementation phase: installation, operation and maintenance.

[Kend92] defines seven steps in a system development life cycle as: problem definition, feasibility studies, analysis, design, construction, conversion and maintenance.

The difference in the steps of these system development cycles depends very much on the author. For example, many authors include requirement/design specification as one of the steps in the systems development life cycle. With such differences in description of various stages, the most common elements represented in a development software application are: analysis, design and implementation. The life cycle describes the sequence of stages to be performed. There are several kinds of life cycle of development process, such as the spiral model, V-model and increment model[Somm99].

It is useful to understand these three common concepts used in life-cycle for the development of software applications. The implementation stage is clearly identified as one of transferring the system design into a specific language that allow the system to be executed, but the analysis and design stages are prone to misconception. Therefore, in this section we discuss the difference between analysis and design.

Analysis is essentially the process of deriving notional systems and understanding their relevance to the situation in which problems are perceived. Design is the process of deriving models that are expected to bring about the behaviour of the notational systems. The design should consist of an attribute of a creative activity of constructing many elements and organising them into not just one, but many viable unified wholes, such that each one is capable of realising the notational system. Table 2.1 describes the analysis vs. design [Jaya94].

| *Criteria* | *Analysis* | *Design* |
|---|---|---|
| Role | Problem formulation using system notions | Solution design using system notions |
| Function | To identify relevant notional system(s) to the desired state | To identify relevant elements of the notional system(s) |
| Primary concern | To define the context relevance of system | To define the content relevance of system |
| Address questions | What and why? | How and whom? |
| Measure of performance | Contribution of notion system's performance to the desired state | Contribution of the integrated elements to the notion system's. |
| Primary skills required | Critical thinking | Creative thinking |

Table 2.1: Analysis vs. Design

According to the above methodology definition, we identify the three key issues that need to be addressed in characterising a software development methodology.

*Firstly,* a software development methodology should cover certain phases of the development process[12]. Each phase provide with techniques and modelling notations that facilitate representation of the system.

*Secondly,* a good software development process provides clear guidance on movement from phase to phase.

*Thirdly,* the method should be general purpose for a specific set of philosophical agent paradigms and not focus on specific agent techniques (mobile agent or intelligent agent) or problem solution (Filtering, business process, e-commerce).

### *2.2.2. System Development Approach*

The discussion this section is concerned at the chronology of an engineering approach to software development methods from structured to agent-oriented approach as shown in Figure 2.6. It shows the evolution of software development approaches charted against various foci of analysis.

[12] at least on phases such as analysis, design and implementation.

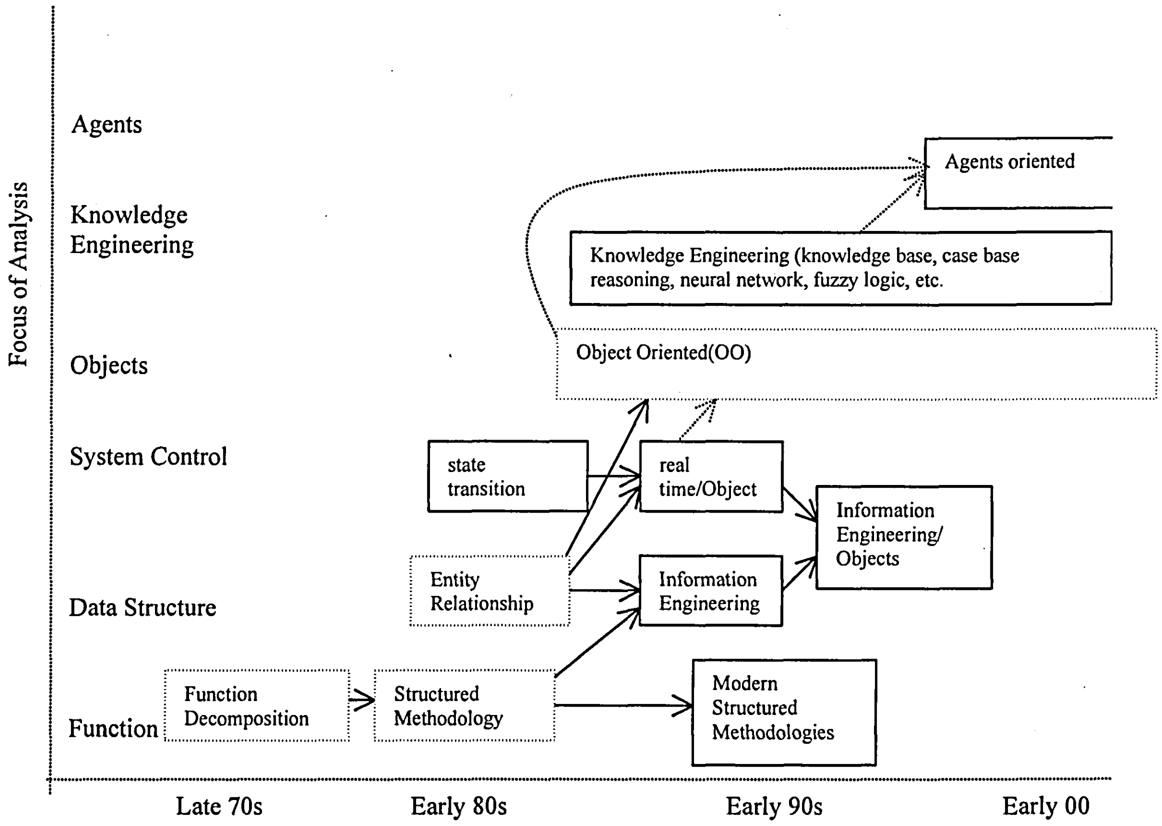


Figure 2.6: The Evolution of the Software Development Methodologies

Software development approaches can be divided into three approaches: *structured, object-oriented* and *knowledge based.* The former category was the pioneering approach of system development based on the structured school, embracing functional, data structure and control approaches, a variant of the structured school. Originally it was used for cutting-edge complex applications; however, continued enhancements were required to accommodate the difficulty posed by ever increasing size and complexity of problem domain.

The structured approach is based on top down functional decomposition of a system. It was the pioneering approach in software development methodology in the mid-to-late 70s and focuses upon what a system does, rather than how it does it. Therefore, it emphasises logical rather than physical structures. The approach focuses on data stored and processes that support the data stored, and so popularised the use of data flows and entity relationships. One of the most popular structured approaches is by Yourdon[Your79] and De Marco[DeMa+78]. It focuses on the complete life cycle development of the development process(SSADM)[Ash+90].

In the early 80s the object-oriented methodologies started to be explored and were soon being used to handle complex applications involving concurrency, distributed systems, network management and e-commerce. At this time too, the knowledge engineering paradigm was introduced with the promise of better solutions for complex systems. These attempts have been largely successful in providing solutions to various applications, particularly the complexity of rules for making decisions and reasoning in expert systems.

Agents-oriented is a combination of the strengths offers by object-oriented and knowledge engineering approach. In the early 90s the idea of agent, originated from Artificial Intelligence (many of previous agents application follow this thread). However, the latest research found that an agent-oriented approach stemmed from object-oriented ideas, while the knowledge engineering techniques aided agents to provide intelligence such as plan, reasoning, etc.

#### 2.2.2.1. Knowledge Engineering Methodology

CommonKADs is the only knowledge engineering approach that provide a full set of a methodologies[Breu+94][Tans93][13]. CommonKADs consists of seven types of abstraction to represent the analysis and design the problem domain: organisation, application, task, co-operation, expertise, agent and design[Waer+93]. The first three involve identifying the problem to be solved in an organisation, through functions that define the actual task that the knowledge based system has to perform. The co-operation model is the specification of the functionality of subtask in the task model. The expertise model is the central activity in knowledge based construction, because it specifies the problem solving expertise required to perform the problem-solving tasks assigned to the system [Breu+94]. The agent model is the abstract description of objects and operations, which comes from the abstraction of expertise and co-operation models, whereas the design model describes the realisation of problem solving behaviours described in the expertise and communication models. The model distinguishes between types of subsystems, application, architecture and platform design. In the task model, agents are identified to represent components of the task model; thus reducing the complexity of task model.

The pioneer agent-oriented development methodology being proposed is an extension of CommonKADS methodology. The strength of knowledge engineering from an agent perspective is the fact that it matured in providing intelligence techniques such as rule-based, case-based reasoning, fuzzy rules, neural networks, etc. which are useful for agent intelligence. The rules-based approach provides a technique of forward and backward chaining that allows accurate modelling of decision-making. Case-based reasoning (CBR) offers a technique for deciding how agents can act in the current situation. Several CBR tools have been used for specific tasks, for example Art-in and CBR-express. Fuzzy logic provides a technique for agents to make decisions, whereas the neural networks technique offers pattern recognition (back-propagation, kohonen map) and machine learning techniques such as genetic algorithms which enable agents to learn about their environment behaviour or learn about human behaviour[Rusel+95].

In the knowledge engineering community, the best method of creating a rule-based system is "rapid prototyping", which means creating a set of rules to solve a specific case and executing it to see whether the result solves similar cases with the rules it has in its knowledge base [Harm93].

#### 2.2.2.2. Object-Oriented (OO) Approach

The object-oriented approach is based on objects and has been presented as a proper systematic development methodology. Several object-oriented methodologies have been proposed, such as Coad-Yourdon's [Your89], Object Modelling Technique(OMT)[Rumb91], Object-oriented Design[Booc94], Object-oriented Software Engineering (OOSE) [Jaco+92]. . A brief survey of the above object-oriented approaches shows that the methodologies have different strengths and weaknesses, and presented different modelling notation, but all of them similar abstraction of objects by showing classes, attributes and relationships to present their methodology. The developer is not restricted to using any specific development process.

Despite of various kind of object-oriented methods and notations we follow the Unified Modelling Language (UML)[Rati97][Booc98][14]. UML encompasses three kinds of building blocks: Objects, Relationships and Diagrams. All these building blocks show the static and dynamic modelling. UML provides techniques to show the structure, behaviour, grouping and annotation of objects[Larm97]. It also shows several kinds of relationships such as dependency, association and generalisation. On the other hand, UML provides several kinds of diagram of class, object, use case, sequence, collaboration, state chart, activity, component and deployment diagrams, which are able to show the static and dynamic activity of system. The UML have potential features to present a static and dynamic modelling for agent-based systems.

A deep understanding of agent concepts associated with the current software development approaches has led us to our choice of the most suitable agent development namely an extended object-oriented approach for the following reasons:

[13] It has become the world wide standard for Knowledge Based System (KBS) development methodologies and the tools that implement the CommonKADs methodology such as ILOG KADs.

[14] UML now been agreed as a standard object-oriented modelling languages

*First*, the concept of agent is quite similar to that of an object, though an agent is defined as an active object. This means the abstractions for an object and agent can be quite similar. An object is abstracted by attributes and methods, whereas an agent is abstracted by behaviour (autonomous, reactive, etc.). Moreover, for our purposes an agent is extended to include a knowledge engineering perspective in that an agent is abstracted by several processes that are linked together to represent agent behaviour, where each process consists of several tasks carried forward from [Treu+98].

*Second*, extension of object-oriented approaches has been the basic of existing agent-oriented methodologies [Shoh93][Wool+95][Frank+97][Geor+95]. Moreover, at the implementation stage features of the common usage of object-oriented language such as inheritance, polymorphism and aggregation can be utilised for implementing agent-based systems.

*Third*, the maturity and advantages offered by object-oriented methodologies provides a robust starting point for development of agent-based systems[Meye+93].

*Fourth*, object-oriented languages are popular among software developers. Developers tend to put more efforts into the language that they know most. Recently, more then 70% of commercial agent systems (agent frameworks) have been developed in object-oriented languages[Tools99].

*Fifth*, the object-oriented approach is suitable for large scale complex application to present agent technology. Object-oriented provides various terms of design pattern, architecture patterns and software architecture that play role in developing a quality complex object-oriented development.

Another strength of an object-oriented approach is reuse. This is shown in the development of object-oriented frameworks[Bell+99]. There are several off-the-shelf frameworks such as communication protocol components using TCP/IP, CORBA, HTTP, Distributed Component Object Model (DCOM) that can be utilised taken advantage of, for constructing the properties of agent systems such as distribution, concurrency and communication. This is because the object-oriented approach is popular and is used in development of distributed artificial intelligence and distributed systems. The object-oriented languages provide functionality to use off the shelf frameworks. Therefore, the frameworks are easy to 'pick and plug' in an object-oriented approach.

As we discussed previously, most agent systems are developed upon an agent frameworks. An agent framework provides an environment for developing agent systems as programming languages do. But they are different because the physical design lies in the agent framework that has association with user requirement rather than a standard to presenting agent technology. We view development agent frameworks is part of development of agent systems, which is the most complex stage of development. Therefore, in order to reduce such complexity, various software abstractions are introduced. The common software abstraction tool such as software architecture, frameworks and patterns are addressed to reduce complexity of development agent framework.

In the next chapter, we will discuss more detail the means of abstraction and the potential used for design agent systems.

## 2.3. SOFTWARE ABSTRACTION

Abstraction is a key part in any software development process [Triem99]. Software abstraction is a tool or technique used to omit details throughout the development of a system. The UML used for development object-oriented system; while data flow and entity diagrams are used in SSADM. Agents offer expressiveness and flexibility since conventional notation is not enough to support agents. The use of software abstraction enables having a clean software development process, rather than having ad hoc implementation codes, as an ad hoc method is not a suitable approach for the development of a complex system. The implementation code represents a very lower level of abstraction, which difficult to understand and difficult to maintain. Moreover, it not able to show the development process that other developers can follow.

| **System requirements specification** | **Type of abstractions** |
|---|---|
| Design method | Function, Object, Agent |
| Modelling | Class, State Diagram, Context Diagram, Activity Diagram, etc. |
| Subroutine specification | Framework, components |
| Type, class | Architecture |
| Pseudocode | Pattern |
| Performance Criteria | Theory(set of statements) about a class or type |

Table 2.2: Software Abstraction

There are several techniques defined as software abstraction, which are used in different levels of complexity and stages of a system, as shown in Table 2.2 [15].

There are several terms used in relation to abstraction: refinement, generalisation, specialisation, documenting a refinement, views and compositions, which we can break down into specification, patterns, interface, type, method, class, protocol, collaboration, framework, message, design, function, etc.

The software development methodologies discussed in the previous section used software abstraction. In object-oriented approaches for instance, several techniques of software abstraction are introduced to reduce the complexity of the development process. Similarly, in the cases of our research, the aim is to identify suitable software abstraction uses for development of agent-based systems such as abstraction to describe agent and appropriate modelling suitable for analysis and design agent system. On the other hand, the software abstraction of framework, software architecture and patterns are necessary for development agent environment[16].

In this research, we focus on identifying abstraction to model agent, and identifying appropriate modelling for analysis and design agent system. However, in development agent environment we utilise software abstraction, as discussed in the above sub-section.

### 2.3.1. *Software Architecture*

Software architecture is like system architecture in that there is no general agreement on the definition. It can be interpreted differently depending on the context. For example, [Booc94] defines architecture of the system as a class structure. He suggested that part of the architecture is the way in which classes are grouped together. [Rumb91] used the term architecture to describe the overall organisation of a system and sub-system. [Shaw+96][Shaw+93 define software architecture as software at a high level of abstraction that can be described as a number of distinct elements or subsystems combined together with their interconnection and interactions. [Busch+96] defines software architecture as '*a description of the subsystem and components of software system and the relationships between them'*. Subsystems and components are typically specified in different ways to show the relevant functional and non-functional properties of a software system. The system architecture of a system is an artefact.

[15] The last four types of abstractions are define by [Triem99], while the first two are also define as abstraction.

[16] Agent environment here means Agent Frameworks.

Software architecture are used for three purposes: develop architecture for individual software system[Hofm+99], product-line architecture[Bosc99][Bosc00] or domain-specific software architecture[D'So+98].

Based on the above discussion we can see that software architecture is an approach to software development that has shifted .developers' focus from fine-grained units to coarser-grained architecture elements of their inter-connection structure. They enable developers to abstract unnecessary details and focus on the big picture of the system being developed. Software architecture may be seen as method to guide the design architecture for development agent environment. [Soni+95] identified the four view different aspects of software architecture as below:

| **Type of Architecture** | **Examples of elements** | **Examples of relationships** |
|---|---|---|
| Conceptual | Components | Connectors |
| Module | Sub-systems, modules | Export, imports |
| Code | Files, directory, libraries | Includes, contains |
| Execution | Tasks, threads, object interaction | Uses, calls |

Table 2.3 : Four aspects of Software architecture [Soni+95].

There are various kind of applications which use software architecture such as framework applications[Fayad+01], enterprise applications[Hamu+98] or family types of applications[Bosc00]. All these types of application provide environments for a specific type of application. The agent technology, as discussed in the previous section, is the functional requirement for the agent environment and focuses on the aspect of reuse similar to these kinds of applications.

The issues associated with agent technology presented the potential of use software architecture to model structure of agent environment (agent framework) that provides functionality to create and executes them and also allow them to socialise.

### *2.3.2. Architecture Pattern*

Architecture patterns are developed based on experience of a rapid software development to produce good software architecture. Therefore, the essential contribution of architecture pattern is in providing an indication as to which architecture style is appropriate to a system. Software architecture is an architectural choice rather than a development life-cycle.

Several architectural style solutions have been identified, such as client-server systems, a pipeline design, layered architecture, blackboard, broker, model-view-controller, reflection and presentation-abstraction-control [Shaw+96]. Each architectural style is described in terms of structural organisation, called architectural patterns. The architecture pattern is a fundamental design decision when developing a software system, given that the designer is familiar with the patterns. Several architecture pattern catalogues have been developed such as pipes and filters, distributed system broker, blackboard, model-view-controller, adaptation system and microkernel [Busch+96]. The architecture patterns guide the developer on identifying abstraction. However, not all architecture patterns exactly match a system solution. Some of these architecture styles are commonly associated with each other for a particular solution of a software system. For example, adaptable systems use Microkernel and reflection architecture patterns, while the interactive systems use the Model-View-Controller and Presentation-Abstraction-Control architecture patterns [Busch+96]. Architecture patterns provide guidance to produce a quality architecture design and it will be used for developing agent environment.

### *2.3.3. Design Patterns*

Design patterns are tools used to guide the object-oriented designer in producing a quality design. It has similar aims to architecture patterns but it focuses on a object level design[Prec+97][Gam+95].

A pattern is an abstraction from a concrete form, which keeps recurring in specific non-arbitrary contexts [Rieh96][Gam+95]. The problem occurs within a context and has a proposed solution that involves a structure, which balances this concern, which is most appropriate for the given context. Pattern form is used to describe the solution and tries to capture the essential insight, which it embodies, so others may learn from it and make use of it in similar situations. The solution can be applied in a different situation and has to be adapted to fit the needs of the specific situation[App97].

A pattern should describe a single kind of problem, it should specify the context in which the problem occurs, the solution as a constructable software entity, design steps or rules for constructing the solution, and the forces leading to the solutions, provide evidence that the solution, and resolves forces and describe details that are allowed to vary [Lea93].

Patterns have been used in the application, development and documentation of software applications. In the development process, patterns play an important role in the various stages of analysis, design and implementation[Cunn+87].

According to the pattern definition, a pattern is concerned with a subject. The subject focuses on the problem domain, a system design or a program implementation. The subject describes the classes or components, objects and relationships between them. In other words subjects could be models of problem domain, interaction diagrams and source code. Therefore the term pattern has been widely used in relation to architecture patterns and design patterns.

Design patterns can be presented in two ways: in a design pattern catalogue like a design handbook, which contains a number of problems and corresponding solutions; or by incorporating design patterns into the language in which the designer thinks, as a design language.

Design patterns are divided into three categories: patterns dealing with flexible object creation, patterns dealing with structural aspects of object systems and patterns dealing with behavioural aspect of an object situation. Design patterns are independent of application domain but rather map to a specific situation implemented before in specific classes of systems.

A design pattern is presented as a template. Each template consists of the following attributes which are used to describe design patterns: Name, Problem, Context, forces, Solution, Examples, Resulting Context, Rationale, Related Patterns, Known Uses. Several design patterns have been produced and catalogued [Gam+95][Gran98][Gran99][Nels99], which are used for general purpose object-oriented design solutions. Idiom is a kind of design patterns which ties to a particular language. Its useful to transform an object-oriented design into an implementation code.

We believe the development of agent environment is mainly uses object-oriented technology, therefore design pattern plays important role in development agent system.

### *2.3.4. Architectural Description Language(ADL)*

Architecture style presents a very limited architecture of a system and does not clearly define how to model for a very specific condition. ADL is a modelling language that support architecture-based development, similar to the way UML supports object-oriented development. However, ADL focuses on the high-level structure of the overall application rather than implementation details of any specific source modules[Ves93]. It allows for a precise description of the externally visible properties of software architecture, supported by different architecture style at different levels of abstraction. Externally, visible properties refer to properties of a component, such as its provided services, performance characteristics, error handling and shared resources.

ADL is a language that provides features for modelling a software system's conceptual architecture. It provides a concrete syntax and a conceptual framework for characterising architecture.

A number of ADLs have been proposed for modelling architecture within a particular domain, Aesop, MetaH, LILEANNA, Artex, C2, Rapide, UniCon, Darwin, SADL and ASME[Medv+97]. Each ADL mainly consists of component, connectors and architecture configurations. A collection of ADLs is used for development of software architecture [ADL00]. Understanding these domains and their properties the key to better understanding of the development of software architecture. ADL has been classified into the following: Editors/Viewers, Static Analysis, Runtime Analysis, Simulation and Monitoring, Transformation to Implementation, Interface or Code Generators, Reengineering, Recovery, Correspondence, Conformance Testing, Evolution, Test Case Generators - Design, Documentation and Tradeoff Analysis. Each classification describes the ADLs used [ADL00]. Architecture patterns are used for a high level design while ADL presents a more precise design. ADL is used for designing and developing the agent environment, rather than designing the agent system itself.

### *2.3.5. Frameworks*

A framework is a partial implementation system that can be executed to provide certain kinds of functionality. It is a partially complete software or sub-system that is to be instantiated[Booc94b]. Frameworks may be small systems or may consist of the integration of many sub-systems. Development framework tied with a specific platform execution. Frameworks are generally a hybrid of architecture-level information and implementation. A framework defines the architecture for a family of sub-systems and provides the building blocks to create them.

Agent environment is a framework that consists of various sub-systems. Software architecture and architecture pattern are the key tools for the development agent environment, while in this case a framework is a special case of the software architecture approach used for the development of an agent environment.

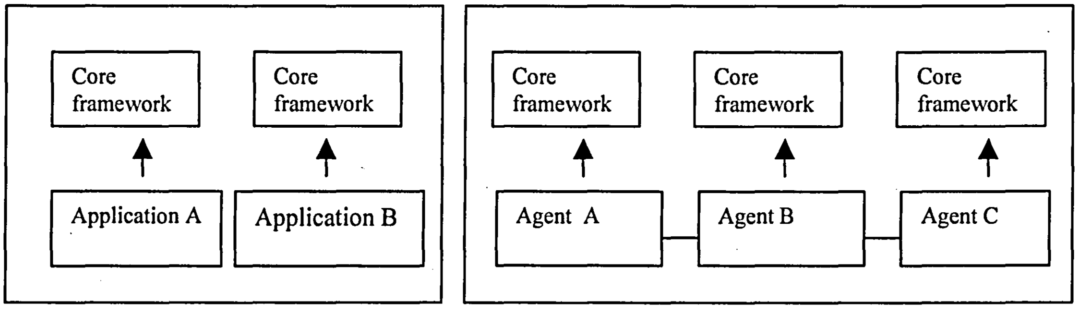
Frameworks are useful for the development of specific-domain applications that can be reused. In an object-oriented community, a framework comprises abstract and concrete classes. The A framework is instantiates by composing and sub-classing of existing classes that are easy to use for a specific domain. Frameworks enable the skeleton of an application that can be customised by an application developer.

Frameworks are useful for the development of software applications that involve the design and implementation of complex software, especially in view of the heterogeneity of hardware, operating systems and communication platform which make it hard to build and expensive built from scratch[Booc94b].

A framework is considered as a semi-complete application target for a particular domain, whereas an object-oriented reuse technique is based on class libraries. Some examples are Microsoft Distributed Common Object Model(DCOM), Java-Soft's Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA) and Microsoft Foundation Classes (MFC). Frameworks are also used for more complex applications such as telecommunications, distributed medical imaging and real time. Frameworks have become the next generation of object-oriented applications, targeting complex business and application domains[Fayad+01].

The benefits and design principles underlying frameworks focuses on reuse, for example in system infrastructure framework (operating system[Russ90]), middleware integration framework (database management [Bato+89], network protocol software [Huen+95]) and enterprise application framework[Meyr86][Baum+97], routing algorithm[Goss90]).

The most important part is that a framework is developed using abstract and concrete classes which are useful for presenting the functionality of the agent environment. Two abstract classes are useful for the development of the agent environment: core agent and communication, whereas, the other elements such as behaviour and interaction protocols can be defined as a library of reusable components. The relationships of these two reused components are shown in Figure 2.7. It also shows the difference between the use of frameworks in traditional applications and the use of frameworks for an agent based system. The figure shows that all agents are composed of inherit the core framework depending on the choice of framework design methods, such as Holonic perspective[Sou+98], Consolidate perspective[Paru+01] and AALAADIN perspective[Paru+01]. In addition, the instantiation of a framework by composing and sub-classing the existing classes is useful for presenting agents and creating many agents. A framework design provides features at two levels: domain features and structural characteristics. The domain features describe domain-relevant features useful in the application and structural characteristics describe the features facilitating the adaptation and the evolution of the framework. The structural characteristics are concerned with both the logical and physical structure of the framework[Pree96]. The structural characteristics are important because a framework's implementation and design are the actual interface used by the framework's user. Therefore, the design of a framework's logical structure is essential to the application of the framework.



Traditional framework application Framework based for agent based system

Figure 2.7 Dissemination of the Use Framework for Traditional and Agent-based Application

The development of this type of framework is strongly associated with software architecture, architecture patterns and design patterns. The abstraction makes it necessary for development to deal with an increasing number of concerns that are needed for framework development.

The four issues that are characteristic aspects of an architectural abstraction for development frameworks are structure, functionality, abstraction and reuse. Structure is focused on the structure of composition of classes, whose collaboration and responsibilities are specified. This describes the whole structure of the framework. Functionality is concerned with domain features that describe a common functionality of the application. Therefore, the functionality can be reused. Abstraction represents the abstraction of structures and functionality in the domain as generalised structures and functionality. This is the core of the framework allowing instantiation or inheritance.

Framework development process has been matured, it has been used in development of various kinds of application that aims for reused such as library system, Visual Banker and Hotel system[Dyson97], Fire Alarm System[Molin96], Measurement System[Bosc99] and Gateway Billing System[Lund96], in which the similarity of functionalities have been established. These approaches describe the framework development process but are not focused on what components are exhibited and how they connect with each other.

Therefore, architecture patterns and design patterns are used as guidance on how the components or objects are structured in a specific context. In next section describes a framework development process.

### 2.3.6. *Framework Development Process*

Framework development is different from a standard object-oriented application[Rieh96b]. The important distinction is that the framework has to cover all relevant concepts in a domain, whereas the application is concerned only with those concepts mentioned in the application requirements. To set a context for the problem experienced and identified in framework development, the following activities are part of the framework development model:

*Domain Analysis.* This describes the problem domain to be covered by the framework. It captures the requirements by referring to various applications in the similar domain, experts in the domain, and captures any standards, which exist in the domain. The result of the activity is a domain analysis model that contain the requirements of the domain, the domain concepts and the relations between those concepts[Booc94b].

*Architectural Design.* This takes the domain analysis model as input. The designer has to decide on a suitable architecture style to underlie the framework. Based on the selected style, among other considerations, the top level framework is designed. The use of architecture style or architectural patterns can be found in [Shaw+96] and [Busch+96].

*Framework Design.* This is the stage in which the top-level framework and association classes are designed. It may be described as the association between abstract and concrete classes. There are various techniques of framework design include composing modelling -catalysis [D'So+98], hot-spot driven [Pree96], systematic generalisation [Schm96], and structuring large application framework [Baum+97].

*Framework implementation*: The implementation stage is the coding of the abstract and concrete classes to an executable system reuse system.

Domain analysis presents problems unlike those of normal application development. For example, behaviour and attributes are very likely to change in several cases. The analysis of such behaviour and attributes needs to be emphasised in the development model, since it is the nature of frameworks to provide reusable design and implementation that cover the variations and hot spots in the domain.

During development of the framework, an architecture must be selected or developed. The criteria for framework architecture depend on the domain and the domain variation. The architecture must provide a solution to the problem without blocking possible variations in the domain or different solutions to other domain problems.

During framework architecture design, decisions need to be made on whether some part of the framework should be designed as a sub-framework or whether everything should be designed as a single, monolithic framework. Also, interoperability requirements need to be established, for example, whether the framework should cooperate with other frameworks, such as user interface frameworks or persistence frameworks.

On the other hand, [Matt+96] proposed the activities of framework development as below:

- Requirement analysis is carried out by collecting and analysing the requirements of the application that is to be one or more frameworks.
- Based on the results from the analysis of requirements, a conceptual architecture for the application is defined. This includes the association of functionality to components, the specification of the relationships between them and the organisation of collaboration between them.
- Based on the application architecture, one or more frameworks are selected based on the functionality to be offered by the application as well as the non-functional requirements. If more than one framework is selected, the developer may experience a number of framework composition problems.
- Having decided what framework to reuse, the developer can deduce the specific applications that need to be defined that is specify the required application-specific increments(ASI).
- After the specification of the ASIs, they need to be designed and implemented.
- Before ASIs and the framework are put together into the final application, the ASIs need to be tested individually to simplify debugging and fault analysis in the resulting application.
- Finally, the complete application is verified and tested according to the application's original requirements.

## 2.4. THE CRITERIA OF AN AGENT BASED SYSTEM

From the literature review on agents, we have synthesised four agent characteristics that are useful for agent development solution: *behaviour, distribution, communication* and *openness* as shown in Table 2.4.

| | |
|---|---|
| Agent Behaviour | Autonomous, reactive, proactive,<br>believe, reasoning, etc. (refer chapter 2) |
| Distribution | Conceptual<br>Locational |
| Communication | With external world,<br>inter agent and inter entities |
| Openness | the use of ACL<br>not limited to specific communication protocol<br>free to communicate between agent environment |

Table 2.4: Characteristics of Agent-oriented Developments

The literature view reveals that many of the existing agent systems focus on providing agency to agent, which this only part of the agent. Distribution is the highest criteria being utilised for an agent system, which is useful for the development of large complex system with high interaction. In addition, it provides mechanism of socialisation and utilise the criteria of openness and autonomy.

*Behaviour*

Behaviour is the main aspect differentiating agents from other paradigms such as objects or tasks[17]. The combination of certain behaviours presents different agent types rather than a standard view. Agent presents a dynamic paradigm, while object is static, described by its attributes and method.

This dynamic agent paradigm forms the main bottleneck in the development of an agent-oriented methodology that is suitable for all agent systems. Agents may be presented as autonomous, proactive or having intelligence, or a combination of the agent behaviours. This is the reason why the existing agent-oriented methodologies tend to adopt a specific architecture or perspective, rather than a general one. Currently three combination of behaviour acceptable of agent paradigm are autonomous BDI agents, autonomous reactive, and real autonomous. The combination these behaviours represent the compulsory attributes for an agent, while others are optional behaviour added into an agent.

*Distribution*

Distribution is classified into two categories: conceptual and locational distribution. Conceptual distribution is utilised from the agent criteria as a method to identify agents from the analysis requirement, whereas locational distribution is focused on providing architecture for the agent environment in which the agents exist. This will be discussed in Chapter 4.

Conceptual distribution allows development of agent based applications involving various characteristics, such as organisation [Wool+00], distance[Paru+97], location[Paru+97], decision making[Buss+00] and logic[Igle+98][Glas96][Elam+99]. Such conceptual distributions are utilised in agent development for several reasons, to reduce the complexity of the system, to reduce transaction cost, and to increase reliability, openness, reusability and performance. These criteria are a part of the non-functional requirements that influence the design. The agent paradigm is a significant step ahead of distributed artificial intelligence,

[17] Just as the object paradigm, task paradigm and functional paradigm are associated with the object-oriented, knowledge-engineering and structured approaches respectively, an agent paradigm is based on behaviour.

multi-agent systems and distributed systems [Brad97a]. Distribution is important for agent systems because it can facilitate the distribution of task and decision making processing, reduce communication traffic, information loading and the communication costs associated with central problem solving, allow gains in speed and computation resources through parallelism, and provide greater robustness through lack of dependency on a single computation node.

In addition to agent behaviour, the characteristic of distribution also contributes to a part of the method of viewing the agent paradigm that has an effect on designing agents. However, for our purposes, we have identified the characteristic of distribution with the aim of reducing complexity. This choice of agent distribution with the aim of reducing complexity is reinforced by [Maes90][Jenn+98]. For example, a system is considered as complex if it involves too much decision-making. The complexity can be reduced by distributing decision-making into several smaller units and each decision making result contributes to the achievement of the global decision making.

Similarly, if a system has too large an information load, it is considered as a complex system. We need high processing systems and bigger space to load a large quantity of information. Distribution of the information into small information units will increase the system performance.

Four types of conceptual distributions are identified as a result of analysing the existing agent-oriented methodologies: organisation[Wool+00], location[Paru+97], decision making [Buss+00], logical or task [Paru+97][Igle+98]. These types of conceptual distributions are accepted as perspectives on identifying the attributes that describe the agent abstraction.

The distribution perspective aims to provide smaller agents that are less complex and have higher performances. It is essentially a modularisation concept for agent-based systems. This modularisation concept is similar to traditional development, such as the structured or object-oriented approaches, which use procedures or functions and interface or library respectively to reduce the complexity of the development of such systems [Booc94]. However, the modularisation in an agent-based system is not restricted to the development process but is also able to reduce the complexity of the system itself. The traditional approaches are not able to reduce the complexity of the system, but are restricted to the reduction of the complexity of the development process. This is because in agent systems, an agent is represented as an independent system.

*Communication*

Communication is the basis of agent socialisation. It is derived from the agent distribution characteristic. Three types of communication are identified. The first is communication of the agent with its environment (external world) using a sensor to get information from the external world or belief about the environment [Treu+99]. The external world is the subject that is being observed by the agent. This external world, eg. user interface, sensor, signal, etc.

The second type of communication is the communication among agents in a multi-agent system[Wool+95]. This communication is derived from conceptual and locational characteristics. This type of communication is a key requirement for agents to socialise, that is, to interact, cooperate and collaborate through the interaction protocol. Technically, the interaction protocol is described using standard agent communication via agent communication languages.

Jenning states that any system that uses agent communication language is classified as an agent system [Wool97]. Although some agent systems are developed without using agent communication language for communication, in general, the agent community has agreed, agent communication language has become a standard language for agent communication [Wool97].

The third characteristic of agent communication is the ability of agents in a multi-agent system to communicate with external entities [Moul+96]. These external entities are identified as neither agents nor as entities. An example of external entities is a legacy system.

*Openness*

In order to allow heterogeneous agents to communicate with each other, the openness characteristic plays an important role. From the literature survey we identified three types of openness. The first type is identified from conceptual distributions which use ACL.

The second type of openness provides agents with a choice of interaction mechanism [Cohen+94]. This means agents can interact using a variety of interface mechanisms such as TCP/IP sockets, HTTP, Simple Mail Transfer Protocol (SMTP) or Global System for Mobile Communications (GSM). This kind of message transportation is located at a lower layer of agent communication. This type of openness only influences aspects of the designing of the agent environment. This is because the design of the agent system is focused on the logical design aspect, rather than the physical aspect, and this type of openness only applies to the physical aspect of design only.

The third type of openness describes how open the agent services exhibited in a multi-agent system environment are reused by other agents located in another multi-agent system environment[Dale01][Bell+99]. Figure 2.8 shows how two multi-agent systems for Marketing and Production use the services of the Pricelist Agent, which is located in the Production system. The marketing system openly uses the Pricelist Agent services in order to achieve the Marketing system goal without creating its own Agent Pricelist. This type of openness aims to provides an open multi-agent system architecture, that allows agent interoperability.

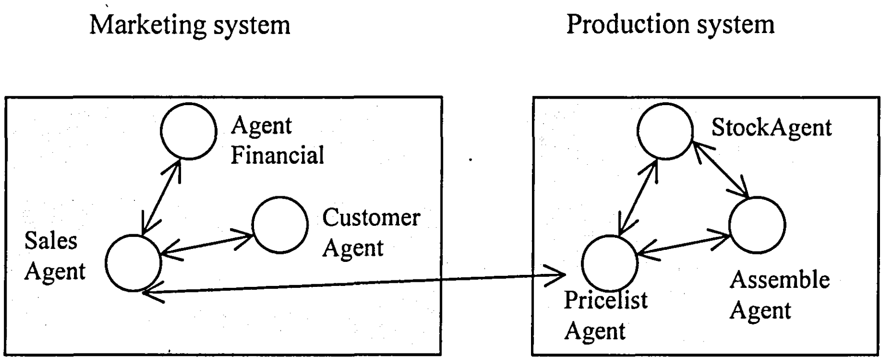


Figure 2.8: The Ability of Agent to Communicate Across Multi-agent Systems

These four characteristics are the key principles used in designing agent systems. The characterisation of behaviours is the core principle in agent system development because it is used to identify whether an agent is an appropriate approach to system solution, whereas the distribution characteristics provides four perspectives on how the system is structured and decomposed to form agents. Decisions on these matters influence the determination of the agent communication and the way agents interact is known as the interaction protocol.

The key issue for development agent environment to provide an appropriate technology for a development agent system which be divided into two kinds of environments: physical and communication as each of them discusses as below.

*Physical environment*

The concept of a physical environment for an agent is analogous to the ecology of an animal, that is, an environment that enables a species to survive. For example, fish ecology consists of water that is described through the variables of temperature, food items and salinity[Wils75]. While artificial agents can have different environments for their survival, they still need an "ecological niche" or physical environment. The agent environment should provide the principles and processes that govern and support the lives of the entities. Different kinds of agents require different kinds of physical environments and vice versa. A Tourist agent system depends on geographic position and uses a satellite as a vehicle for communications, whereas an agent based supply chain is more focused on resources and orders as major components of the system. We realise that artificial agents are much more dependent on application rather than physical conditions, because they are associated with environmental information. Return to the ecology analogy, the quality of a fish's environment is important for the fish to alive. However, what constitutes quality depends on the type of fish. Therefore, the aquarium is designed in a way similar to the fish's ecology. Similarly, artificial agents require quality aspects for their environment.

Based on the literature review, we found several characteristics to describe the physical aspects of an agent environment, as proposed by [Weis99][Rusel+95]:

- Diversity- how homogeneous and heterogeneous the entities in the environment?
- Locality - does the agent have a distinct location in the environment, which may or may not be the same location as other agents sharing the same environment?
- Accessibility – to what extent does the environment know the availability of agents?
- Determinism- to what extent, can agents predict events in the environment? How far is the environment determined according to the agent's current state and action selection?
- Controllability- to what extent, can the agent modify the environment?
- Volatility- how much can the agent environment change while the agent is deliberating?
- Temporality- is time divided in a clearly defined manner? Do the agent actions occur continuously or discretely or episodically?
- Openness- how far does the agent environment provide opportunities of using any communication medium for the agent to communicate?

*Communication Environment*

The communication environment is only needed for a collective of independent agents in a multi-agent system. These independent agents need to communicate with each other. Therefore, they need a communication environment that allows them to communicate effectively.

As a result of the literature review on agent principles in chapter two, we discovered that the communication environment consists of two elements. First, it provides the principles and processes that govern and support the exchange of ideas, knowledge, information and data. Second, it provides strategic socialisation of agents, that is, those functions and structures that are commonly employed to enhance communication, such as roles, groups and patterns of interaction (interaction protocols) between roles and agent groups.

The communication environment should provide the following elements: communication, interaction and socialisation environment. Communication is a method of conveying information from one entity to another and it changes the state of the receiver. Interaction is two-way communication. The sender receives at least an acknowledgement from the receiver that it has received something. Interaction not only defines exchange of information but it also confirms the original sender of information, even though it is received by the other agent. The socialisation environment is even more complex, in that communication and interaction are employed together. It describes the patterns of interactions or interaction protocol, for example, bidding. Other types of social interaction are coordination[Bond+88], cooperation[Haug+98] and competition[Hayn97]. The interaction and socialisation environment are only needed for the development of a multi-agent environment.

Several principles are required to facilitate the communication environment in order to model multi-agent systems.

- Communication language - defines the three aspects of communication: syntactical, semantic and pragmatic. These aspects are required to define the type of messages such as send, request, etc. and the ontology. Examples are FIPA [FIPA+97b] and KQML[Fini+97].
- Interaction protocols - describe communication patterns as allowed sequences of messages between entities and the constraints on those messages. An example is contra-net protocol; FIPA has defined more than ten agent interaction protocols [FIPA00].
- Coordination strategies - describe the method by which groups of agents socialise to achieve their goals. The common strategies are cooperation, competition, planning and negotiation.
- Social policies- describes the permission and obligations according to the social behaviour [Wool+95].
- Culture- influences the differences of set of values, desires, intentions and trust, which are represented in agent communication languages such as FIPA versus KQML. Agent communication languages are free from differences of culture or language [Gene+92].

## 2.5. SUMMARY

This chapter aimed to provide literature reviews towards the development of a systematic method of development agent system. It started by giving a clear picture of agent theory as a new paradigm for the development of software applications by describing the agent characteristics and its technology. The discussion of software development methodology identified three attributes of a methodology that agent oriented methodology should fulfilled: agent design concept, software development process and modelling techniques. The discussion in the relationship between agent and object, and object-oriented modelling is the essence towards agent-oriented method. Beside that we provided various software abstraction techniques, which may be used for development agent system such as software architecture, framework, architecture pattern includes architecture description language and design patterns as appropriate.

In the next chapter we discuss the building blocks towards development of an agent-oriented methodology.

# CHAPTER 3

# 3.The Building Blocks Towards a Process for Agent System Development

## 3.0. INTRODUCTION

The purpose of this chapter is to present an integration of various derivations of activities leading to production of the proposal of a method for developing an agent system. When the research started, agent paradigm was relatively new, the analysis of existing agent-oriented methodologies was proposed based mainly on the developer's thought about agents, which focused on certain aspects of the agent. They focussed on certain aspects of modelling and did not make clear the transaction from analysis to design. Some of the methodologies were very close to object-oriented design techniques, while some of them focused only on certain stages rather than the whole development process and some made assumption that agent framework provide all agent technologies rather than as part of agent development process.

Our proposed agent method is at least a step ahead from them because we propose a method based on analysing the current agent oriented methodologies, justifying them and prove them through the reverse engineering process and experience in developing agent systems to identify appropriate modelling for development agent system. Other differences in our method is the fact that we believe agent frameworks part of agent development, a similar strand to [Zam01], but not yet showing to be the agent framework development process.

The literature review in chapter two clearly discussed that the use of an agent framework is not as an implementation language. It represents an agent infrastructure, which contains appropriate attributes of physical and communication environment allowing agents to exist and play it role. The derivation process has discovered that agent infrastructure is part of an agent system requirement, which may be identified at an early stage.

In addition, an agent framework not only differs in terms of the agent infrastructure but also in the use of a specific technique or mechanism to represent the agent infrastructure, such as socialisation mechanism influenced by the requirement itself. For example, the decision to use centralised or decentralised agent organisations depends on the patterns of agents interactions.

In addition, the scrutiny of existing agent frameworks has found that there are various functionalities to represent agent technologies, which is associated with the requirement of the agent system such as a different techniques to represent the four agent system characteristics[1], different pattern of agent socialisations and different of choosing the use of technology to represent the functionality. Beside that we identify another stage before implementation, called deployment stage, which integrates the design agent system with components presented in the agent framework.

The reverse engineering found that there are various components used in the implementation code that are not modelled while designing the agent system. These components are used to provide the functionality that allows the agents to perform their roles and present the features of multi-agent system as the result of designing the agent system it self. The implementation stage for us means an automatic transformation from the design to implementation using tools such as Rational Rose [Rati97], which transforms object-oriented design into the implementation code of Java language.

With such method we are able to achieve one aim of software development criteria in chapter two to provide a general-purpose methodology with a systematic process that helps a developer to develop agent systems. Dividing agent development into three phases we believed is able to present a clear process to develop agent systems. This mean the gap between analysis and design stage is reduced using modelling.

Our methodology focuses on a hard-methodology approach. Therefore, we focus less on identifying what the system does, but more on translating the requirements through analysis and modelling[2]. This means, we make the assumption that the user requirements have justified using agents by utilising the four agent system criteria. However, we do not specify what kind of agent paradigm is most suitable in an agent solution (as is the current practice of the existing agent-oriented methodologies). Therefore, in the analysis stage, the user requirements are to be analysed to find out the most appropriate agent solution, which influences the design of agent system and identifying agent environment functionality.

The next section shows the derivation process towards, the development of the agent-oriented method, as integration of various derivation activities. Each derivation activity is discussed as to what and how those activities are performed. The last section presents the essence of the

[1] Combination of behaviours, distribution, socialisation and openness as discussed in chapter two.

[2] This is because there are several analysis methods, that can be used such as Soft System Methodology [Chec90][Somm+97][Jiro94][Ash+90], TELOS[Mylo+90] and agent-oriented methodology [Dubo+94].

concepts for analysis and design agent system, stages for development agent system, captured appropriate modelling techniques and notations for analysis and design agent system, and design agent environment as the result of the derivation activities.

## 3.1. DERIVATION PROCESSES TOWARDS THE DEVELOPMENT OF AN AGENT SYSTEM METHOD

The development of the proposal of agent-system method is established from an integration of five key pieces of work: literature review on agents and software development process, analysis of the existing agent-oriented methodologies, reverse engineering of agent systems, an empirical case study involving the scrutiny of the existing agent frameworks, and examination appropriate software abstraction for agent system development as shown in Figure 3.1.

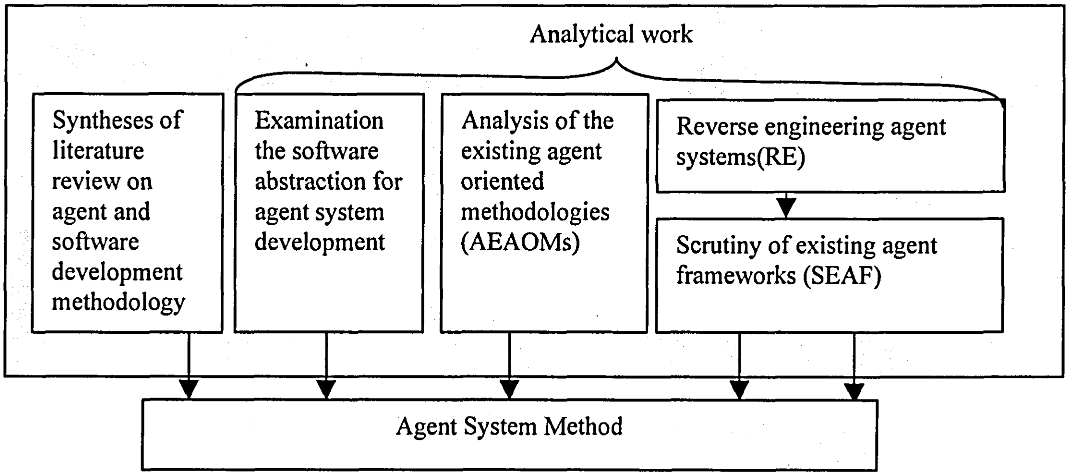

Analytical work
Syntheses of literature review on agent and software development methodology
Examination the software abstraction for agent system development
Analysis of the existing agent oriented methodologies (AEAOMs)
Reverse engineering agent systems(RE)
Scrutiny of existing agent frameworks (SEAF)
Agent System Method

Figure 3.1: Derivation Processes Towards Development of Agent System Development

The proposal of the agent system method is strongly influenced by synthesis and reflection from all these five underpinning components. Figure 3.2 shows how these five processes are integrated towards developing a method for *development agent system* and *development agent environment.* The development agent system part was derived from the literature review, AEAOM, RE process and examining appropriate abstraction for modelling agent, and analysis and design process modelling[3]. The development agent environment was derived from the

[3] Observation the modelling techniques provided in structured, knowledge-engineering and object-oriented approaches.

literature review, reverse engineering, scrutiny of existing agent frameworks and investigation of the use of software abstraction[4] in the existing agent frameworks.

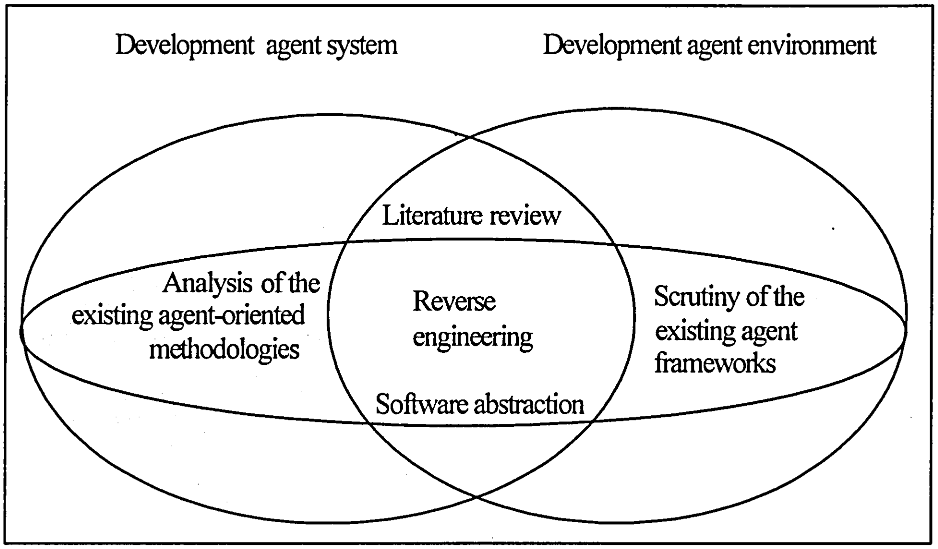

Development agent system
Development agent environment
Literature review
Analysis of the existing agent-oriented methodologies
Reverse engineering
Scrutiny of the existing agent frameworks
Software abstraction

Figure 3.2: The Sequence of the Derivation Process

The literature review on agents and software development processes was the fundamental starting point for proposing the agent-oriented methodology as discussed in Chapter 2.

Reviewing literature on agents has provided a general understanding of agent concepts, agent technology and how agent systems developed in practice. The literature on existing software development processes as discussed in Chapter 2 provides an insight into the stages and intermediate deliverables required for an effective software development process. This includes identifying appropriate modelling techniques and notation in each stage. The literature on agent technology provides the boundaries and limitations for the development of agent systems. This understanding has lead us to discover a set of characteristics that an agent based system should satisfy. The set of characteristics provides further detail of the scope that an agent system method should cover, and includes the surrounding technologies related to development agent environment.

The objective of examining the existing software abstraction is to identify what kind of software architecture can be used to develop agent system in order to reduce the complexity of

[4] Focus on frameworks, architecture patterns and design patterns as discussed in chapter two which focus on development object-oriented framework.

agent development. It also indicates what, when, where and how software abstractions would be used. This includes observation of the current modelling techniques of the traditional approach which may be appropriate for modelling agent system, and observing potential software abstraction for development agent environment. The identification of these abstractions is the key elements while performing the other three derivation process: analysis existing agent-oriented methodology, RE and scrutiny existing agent frameworks.

The purpose of AEOM is similar to that found in the literature review, but more closer to agent-system development. This includes identifying agent design concept, potential stages to representing an agent system development process, and identification of appropriate modelling techniques in each stage. Analysis of each existing agent-oriented methodologies is novel and provides insights into the strengths and weaknesses of these methodologies, as well as benchmarks for our proposed methodology.

RE is a powerful process for development agent system method. This is because when our research started in 1998, the agent paradigm was relatively new in the technology era and the agent research was focused more on theoretical rather than practical. RE process is able to eliminates this gap. Many of agent-based systems were developed in an ad hoc fashion and did not present a proper development process. RE provides deep understanding how agent system is modelled and developed technically, guiding us in developing the Filtering application(case study shows in Chapter 5). It also a partial process to produce a clean process to develop agent system.

Having background knowledge of agent design concepts, general idea of agent software development process and potential modelling techniques, the used reserve engineering techniques is the main process to prove these potential modelling for development agent system. The RE some part of the components of the existing agent framework is also performed to prove the use of software abstraction in the development agent framework.

The scrutiny of the existing agent frameworks aims to obtain a deep insight how an agent environment is developed. The scrutiny process is able to abstract various common components, which are used in the agent environment, which represents the common functionality given to the agent environment. It is able to identify the common techniques to represent agent technology. The following sub-sections illustrate in more detail the activities in each derivation that involves empirical work.

### 3.1.1. Examining Software Abstraction

Examining software abstraction is performed throughout the three derivation activities: AEOM, RE and SEAF. Each activity has it own aims of identifying software abstraction as discussed in previous section. Figure 3.3 shows the process identifying software abstractions to develop agent-oriented method. Seven types of abstractions are observed: agent concept, analysis and design modelling, stages of development process, design patterns, framework, architecture pattern and software architecture, captured from the literature review. These types of abstractions are the main attributes for development an agent software methodology. The figure shows how those abstraction types were developed. For example, the agent concept is identified from literature review (Chapter 2). This agent concept is then discussed in more detail using agent model as the result of AEOMs and RE processes. A similar process is performed on other abstractions types. Examining how software abstractions performs through the three derivation activities is shown on the right side of the figure.

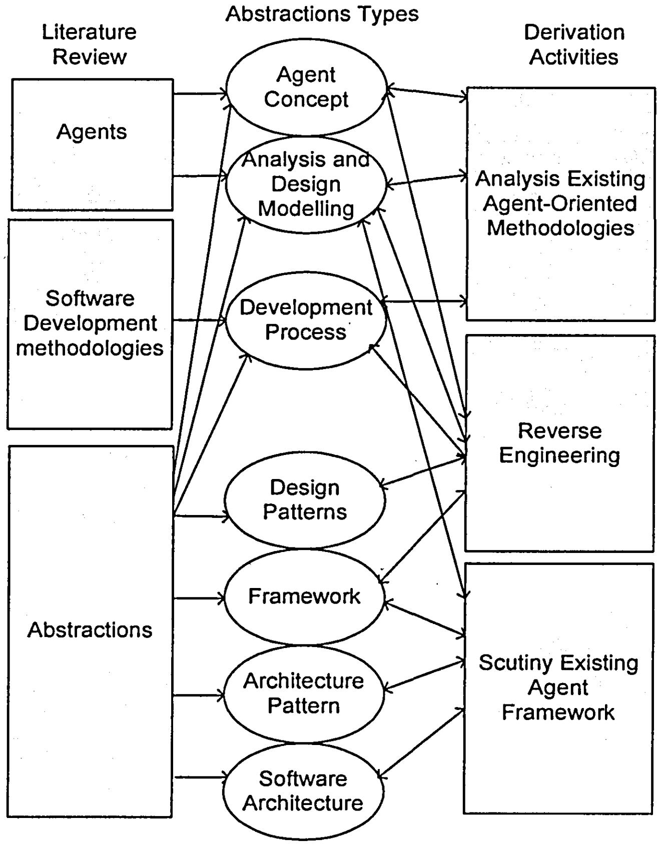


Figure 3.3: Identification of Software abstraction Process.

### *3.1.2. Analysis of Existing Agent Oriented Methodologies*

Ten existing agent-oriented methodologies have been analysed, and may categoried into one of two groups of approaches[5]: extension from knowledge-engineering and extension from object-oriented approaches. Those methodologies were published before 1999[6]. The CoMoMAS[Glas96], MAS-CommonKADs[Igle+98] and DESIRE[Treu+99][Treu+98] were categorised into the first group, while agent-oriented analysis and design [Burm96], Agent Modelling Techniques for systems of BDI agents [Kinn+96], Multi-agent scenario-based method [Moul+96], An Agent-Oriented Methodology: High-Level and Intermediate Models [Elam+98][Elam+99], Agent-oriented methodology for enterprise modelling [Kend+95], Methodology for design of agent-based production control system [Buss+00] and Gaia [Wool+00] are categorised in the second groups. The general descriptions, critical evaluation of each methodology and analysis result are presented in Appendix A. From such a process we are able identify the five following issues:

- various techniques to describe the concepts for designing agent
- identify the potential stages and the life cycle of agent development process
- how far the methodologies have focused on analysis, design and implementation stages
- what modelling techniques were used for analysis and design agent system
- how far they are incorporated with the four criteria of agent system.

### *3.1.3. Reverse Engineering*

RE is a process of analysing an existing system to identify the system's components and their inter-relationships [Bell+98]. As a result, higher levels of abstraction can be represented in another form [Chik+90]. It produces a descriptive model that represents the system *as-is*. [Wood+01][Theo98] have suggested that the RE process is understood as a "reverse" progression through implementation, design architecture and requirements phases. However, the RE process does not follow the sequence of the reversed waterfall model, to abstract the lower-level system to the higher levels of abstraction. RE was performed in part using an automation tool named FUJABA [Nick+99]. The use of FUJABA is able to present an accurate

[5] Some researchers used structured modelling to model agent such as data flow diagram[Hun+95], workflow process[Klie95] but later [Farh96] stated that structured modelling is not suitable for modelling agent based system.

[6] Our research is started. Even Gaia is published in 2000 but it the same stage of others as well but it has describes a clear concept of analysis and design agent.

RE process to produce a class diagram [Nick+00]. In order to fulfil the purpose of our research, the RE process is performed with several activities as shown in Figure 3.4.

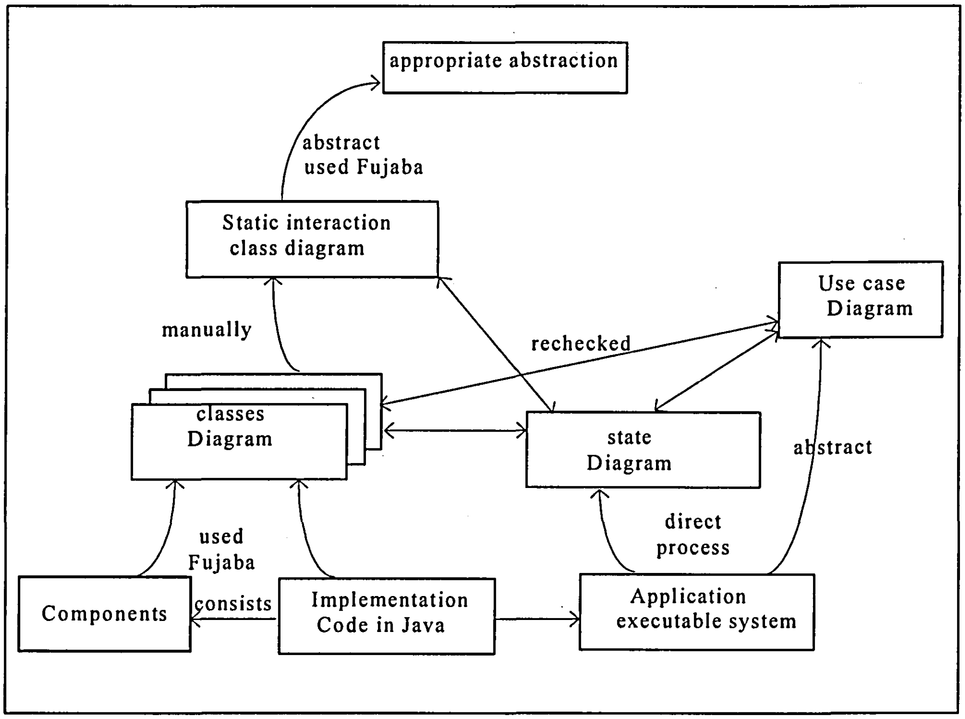


Figure 3.4: The Reverse Engineering Process

The RE processes were performed in two stages. First RE process aims to identify agent model and modelling techniques appropriate to develop agent system,. The second RE process is to find and show the use of software abstraction in development agent environment. The first process reverse engineered two agent applications which developed in ad hoc fashion: NewsFilter Application[Bigus+97] and Filtering application[Othm+02b][7]. The NewsFilter Application is developed using CIAgent Framework[Bigus+97] while Filtering application is developed using JADE[Jade98]. The second process reverse engineered some components in both agent frameworks such as JADE agent and CIAgent component, in particular to identify design patterns and agent architecture. The CIAgent separates the components into 'Agent' and 'Communication' component. The Newsfilter application only used the CIAgent component because it applied to a single agent approach. In order for it to be applied to multi-agent systems, the communication component is used for communication using ACL(KQML). However, an 'inter-mediator' component is provided that allow agents to socialise (such as

8. Newsfilter developed by [Bigus+97] but without providing any clear documentation to represent the how the system is developed while the Filtering application we developed in ad hoc fashion.

negotiate). JADE framework is a multi-agent system environment, therefore the component of 'agent', 'communication' and 'inter-mediator' components are already built in JADE. Both of the RE processes are presented in Appendix B.

### *3.1.4. Scrutiny of the Existing Agent Environments*

As stated before that the development agent framework is part of agent system development. Analysing various agent frameworks allows to make distinction between various agent frameworks, identify common functionality, and leads to the identification of a process to develop it. Currently various agent frameworks are developed [Tools99][Toolkist03].

In order to fulfil our research purpose, four agent frameworks were scrutinised: JAFMAS[Jaf97], JATLITE[Jat97], JADE[Jade98] and CIAgent[Bigus+97]. These frameworks were chosen because, at the time our research started, they were the only ones that provided open source code, which enabled to us to investigate, execute and test the frameworks. Another reason is because they provide a significant amount of requirements functionality to facilitate the development of a multi-agent system agent environment.

The scrutiny on the existing agent frameworks is focused on an abstraction for multi-agent system environment rather then focused on providing intelligence or mobility issues.[8]

Furthermore, they are developed in the same programming languages since we assume assumption that they use the object-oriented approaches to develop agent frameworks. Therefore, we use similar abstraction techniques to abstract the component.

The scrutiny process is performed from documentations, which were given. The functionalities are proved by installing and executing the examples of agent systems developed using those agent frameworks. However, in certain cases, we carried out some RE when the documentation was unclear or incomplete. Most of the documentation shows how to use the framework rather than providing documentation of the system and does not provide documentation on how to develop the agent frameworks.

The scrutinizing process is able to find out the functionalities of the agent frameworks, the reference architecture for the development of agent frameworks and the components used to represent the agent framework architecture. The following are attributes abstracted from the agent frameworks:

- Agent Environment Requirements according to the criteria of agent characteristics, physical environment and communication environments. These two environments are the abstractions of agent environment requirement. The physical environment

[8] Some of the agent frameworks in [Tools99] are focused on intelligence and/or mobility functionality, which out scope of the research.

requirement is abstracted into at least three layers. The communication environment describes the functionality for agents to communicate, including the techniques used to provide efficient interaction. The following points are the abstractions that influence agent environment design.

- Organisation of the agent system describes the method of agent organisation.
- The architecture style used, because the organisation of the agent system has an influence on the appropriate architecture style of architectural design.
- The abstraction of agent environment architecture is based on the layered architecture approach. This presents the components and where they are located. The layer architecture approach is useful to show the layers of the physical environment.
- Internal agent architecture, which describes the agent architecture for the agent framework.
- Component structure, which describes the structure of their components and their relationships. The component structure is able to portray the design of agent environment.

These attributes are used whilst scrutinising the agent frameworks so we can differentiate various functionalities provided in agent framework to be reused and to be suitable for the particular solution of agent system. The description of each agent framework is presented in Appendix C.

Based on the analysis of the document, we assume that some parts of agent environment requirements are identified from the results of agent system design, some are as stated as the requirements, and some are aspects of the functionality of multi-agents itself. This is because those agent frameworks can be differentiated in according various attributes as stated above. The scrutiny also found the architecture patterns and design patterns as we discussed in the section on distillation of results.

## 3.2. THE ESSENCE OF THE DERIVATION OF RESULT TOWARDS THE PROPOSAL OF AN AGENT SYSTEM METHOD

The essence of the derivation result presented here is tailored with three attributes of a software development method: build agent design concept[9], build stages of agent system development

[9] as what object design concept do for object-oriented approach. Object describes by attributes and methods.

process[10] and identified appropriated modelling in each stage. Each attributes is discussed in the following sub-sections respectively.

### *3.2.1. Analysis and Design Concepts for Development Agent System*

This category distils the issues related to the analysis and design concepts for agent system development. The following are the issues related to the analysis and design of agent systems that we have learned from the RE process.

1. Development Approach.

We discovered the elements of the two approaches to agent development: the single agent approach in the case of Newsfilter system and the multi-agent system approach in the case of the Filtering application (See Appendix B). The Figure B1.9 shows interactions between agents not objects (adopted the UML sequence diagram).

2. Differentiation of the criteria of the approaches.

The choice of solution approach is identified based on the agent characteristics captured in the requirements. The single agent approach only presents the behaviour criterion and no inter-agent communication criterion. The Newsfilter system is designed based on the behaviour of autonomous and reactive agents of CIAgent type, while the Filtering application system presents the three agent criteria: behaviour, distribution, communication. The Filtering application agent design solution is also based on autonomous and reactive behaviour of JADE agent. The distribution and communication criteria can be seen in Appendix B(Figure B1.8 and B1.9). The figure shows that interaction between agents uses communication act abstraction. Communication, here, means that the agent's actions are dependent on each other. Distribution, here, means that each agent can be located anywhere. Openness, here, means that each agent can freely negotiate among agents using the second type of openness. JAFMAS and JATlite present different meaning of distribution and openness. JATLITE defines openness by allowing the agent to freely communicate using HTTP or/and SMTP, while JAFMAS not provides openness criteria(through TCP/IP) rather focused on openness of using ACL.

3. The two levels of agent system design.

The RE of the Filtering application clearly identified the two levels of design agent system, agent system design and individual design stages (similar statement with Gaia methodology).

4. Agent Model.

[10] as waterfall model, spiral model and rapid prototyping in traditional approaches.

The abstraction of the Newsfilter application's element is similar to the elements of abstraction of the individual agent in the Filtering application as shown in Figure B1.10. This means the Newsfilter agent modelling is represented as an individual agent design stage.

The element abstractions are: agent, roles, resource, specific task, intelligence and goal. But not all agents comprise these elements. We found that most agents comprise agent, roles and specific tasks. This means that the resource, intelligence and goal are optional elements associated with agents. The relationships between these elements is discussed in further detail later.

3. Representation of the system goal.

The Newsfilter system goal is achieved as the result of interactions between objects, while the Filtering application goal is achieved as the result of interactions between agents. The Filtering application shows two levels of behaviour provision: agent behaviour for each agent and behaviour as the result of agent interactions.

4. The use agent provides less user intervention.

The Newsfilter system presented a high level of user intervention. The User need to choose the search agent used, take action in creating a user profile, save documents and select two times of action to filter the articles. In the Filtering application, the user profile is automatically created for new users and updated every time the user searches articles. An agent is assigned to capture the criteria of the user and what kind of automated profile is suitable to be for particular user. The system automatically updates the profile at the two levels of search without the user's awareness. The User only knows the result displayed and the feedback on articles will be continuously updated to capture the user's behaviour. We can see here how, in an agent system, the load of user intervention is reduced by assigning the user's tasks to the agent role.

5. Inter-media process before implementation.

The Newsfilter used Java language functionality to implement the system. The 'core agent' is an abstraction of the CIAgent, which is represented as having capabilities of autonomous and reactive behaviour to the agent system. It uses a thread to apply this function. In the Filtering application, we see that the implementation uses several components whose names have 'Jade' prefixes. In the implementation stage, these components are reuse components, which we define as inter-mediator language. This means these components are deployed with the design agent to allow individual agent designs to be executed. The inter-mediator language is developed and design based on the agent architecture solution as part of development agent environment.

6. Agent is a dynamic paradigm.

The analysis of the existing agent-oriented methodologies and observing the current agent frameworks found various paradigms of agent-oriented solution which associate with selection of agent framework as we defined as dynamic paradigm such as such as BDI agent[Kinn+96], active object [Elam+98], autonomous and reactive[Wool+98], and autonomous and decision maker[Buss+00]. These views influence the design aspect of agent development and each paradigm represented a way of thinking on agent as problem solution. As an example, [Elam+98] viewed agents as active objects. They use an object-oriented method to identify agents and refine objects by selecting the object presenting the active behaviour, representing a very lower level of agent abstraction.

These agent paradigms influence the design of agent systems. Therefore, we use the term agent abstraction to identify the agent paradigm suitable for the solution of particular problem domain. Even though in the agent overview the agent paradigm is described as the 'core agent' which represents the agent capabilities, it has a great impact on the design of agents, especially on agent identification. For example, the BDI agent is not a suitable agent paradigm solution for the Filtering application. This is because the BDI agent consists of representation of the attributes of belief, desire and intention, which are not found in the requirements of the Filtering application.

| | |
|---|---|
| Agent behaviour | Autonomous in its own thread and able to react with any event captured<br>*React with events come from internal and inter-agent* |
| Communication | with agents - using speech act |
| Distribution | presented location and distribution |
| Openness | provide various option channels of communications |

The agent environment abstraction for Filtering application.

| | |
|---|---|
| Agent behaviour | Autonomous in its own thread<br>*belief*( statement true of false) |
| Communication | no inter-agent communication |
| Distribution | No |
| Openness | No |

The agent environment abstraction for NewsFilter application.

Table 3.1 : The Agent Environment Abstraction of Filtering application and NewsFilter application.

The result of RE of the agent systems presents the detailed meaning of agent abstraction. The criteria of distribution, distribution, communication and openness describe the abstraction of the agent environment. The agent environment abstraction used in the Filtering application and NewsFilter application are shown in the Table 3.1.

The differences of attributes on abstracting agent and agent environment have influenced the agent design process and modelling process. Focusing on the two parts of agent abstraction is enough to illustrate the agent paradigm influence method for identifying agent system.

The use of agent abstraction is applied in our methodology due to the cyclic nature of the agent development process. It provides different levels of agent development. In the case where a designer does not need to focus on the criteria of the agent environment and has decided on the use of a specific existing agent environment, the agent abstraction is exactly defined, and the designer can omit earlier steps and follow the detailed of design agent system.

7. Identify Agent Concepts.

In order to provide a detail design modelling process, we have to define a specific agent paradigm suitable for development agent. In Chapter two we have stated that behaviour is the first agent characteristic as a paradigm of the agent system and the most meaningful behaviour to present the agent paradigm as autonomy and reactivity [Wool+98]. The RE result presented the appropriate analysis and design modelling for autonomous and reactive agent.

We believe these two behaviours provide a general purpose of methodology as stated in section 2.2. The modelling presented here provides the basis of agent concept for all other agent paradigms as well. The BDI agent provides belief, desire and intention attributes, but should present the basis of autonomous and reactive attributes. Based on agent characteristic discussed in section 2.1 we predict that agents may consist of various combinations of behaviour leading to different in agent modelling and different kinds of analysis and design modelling. Therefore, research scope provides modelling and design concept for agent autonomous and reactive, but provide additional room[11] for other agent paradigm as well.

The RE process has found an additional behaviour, pro-activity which particularly valuable for agents. In order to be pro-active, an agent needs to plan to achieve its goal. For example, the UserModelAgent has pro-active behaviour as it is associated with a goal. It is pro-active in continually generating the genetic mutation to produce an accurate user profile in planning how long to generate the mutation. However, the pro-active behaviour is an addition to the autonomous and reactive behaviour. Therefore, in this research we apply an agent concept

[11] as part of methodology criteria proposed by [Chec81].

based on these two basic behaviours, which turn agents from very complex into to simple independent entities.

Autonomous behaviour meets the criterion of concurrency, since each agent as an active object has its own control of thread and operates independently. However, communication between agents and their access to shared data are dependent operations, which we describe as communication, as discussed later.

Autonomous behaviour manifests itself in the non-deterministic nature of the agent's behaviour, which means it can be triggered by any internal event (resource) or external event such as agent and external non-agent. The reactive behaviour is the agent's capability to perceive its environment and react to changes, while pro-active behaviour refers to an action taken in order to achieve an agent goal, depending on the agent state. The second agent characteristic is communication. The concept of agent communication was discussed in the previous section as the basis of agent interaction. Agent interaction is captured when there is dependency on the same resource or information between two agents. This that role presents the most consistent concept used to analyse and design agents.

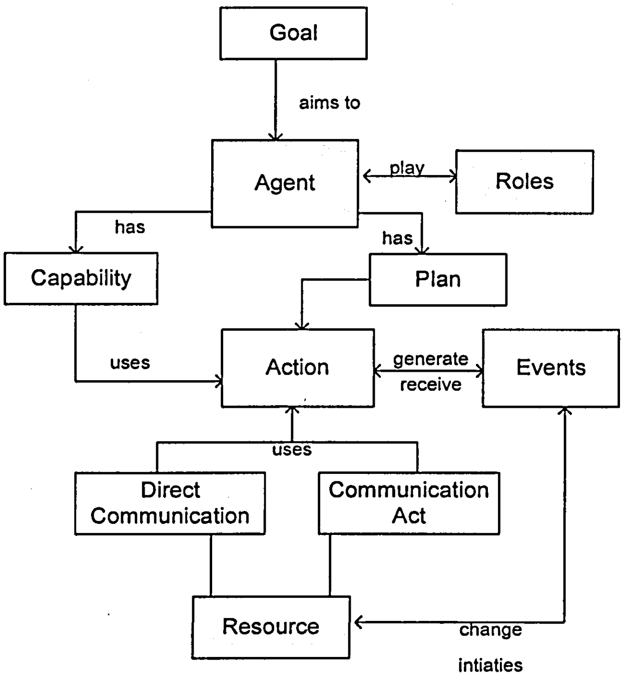


Figure 3.5: Agent Overview

The agent overview as portrayed in Figure 3.5 shows the relationship between the entities to illustrate the agent model (autonomous reactive). This agent overview is captured as the result of the process of abstraction of all agents in the Filtering application (AppendixB-Figure B1.8) and the Newsfilter application (AppendixB-Figure B1.4). As discussed earlier, in the RE process, we found two levels of abstraction. The elements presented in the figure have some similar attributes as concerned by [Jenn00].

At the first level of abstraction, we found the following abstraction of user interface, agent, roles, resource and supportive. The analysis of these elements revealed that an agent consists of four core elements: agent, roles, resources and capabilities. The user interface abstraction is defined as an element of event source, which is only present in the UserAgent.

The second level of abstraction is focused on the abstraction of supportive components, which consists of two abstractions: intelligence and specific tasks. The capabilities abstraction is defined as a combination of the following conditions:

- core agent (basic agent behaviour), in these case studies are based on autonomous and reactive behaviour.
- additional behaviour such as pro-activity, intelligence.
- specific task.

The third abstraction is a communication abstraction, which consists of direct communication and communication acts, which are used to generate action by the resource, user interface (as external object) and agent roles through events. The relationships between the entities describe the agent overview and are used as a pattern throughout analysis and design of the agent system. As a result of above discussion of those entities we abstract the three core entities to describe the agent. The core entities are *Agent, Role* and *Capabilities.*

***Agent.*** This is an entity having **capabilities** to perform the function of its roles. The agent capability is the ability to provide the agent's **services.** An agent must at least present autonomous behaviour, which means the ability of the agent to generate **action** or capture events according to external events, interactions or its own resources, whereas other behaviours refer to the capabilities assigned to the agent. Agent action uses direct communication or communication act to generate action. Agent behaviour properties are identified by analysis of all the agent roles.

***Role.*** The role concept as we found here is similar to the role as described in [Vally+98]. However, we accept the concept found in GAIA methodology[Wool+00] as well. He describes the role as the part played by an agent that is separated logically from the identity of the agent itself. This role describes the external characteristics of an agent in a particular context. This role concept is applied in UserAgent in the Filtering application. The role of UserAgent may be as enquirer or feedbacker. Vally stated that a role is represented as **<goal, properties, resource** and **state>**. For us, the properties are described as capabilities. Therefore a role can be described as functionality, services, activities, specific task and responsibilities. By combining these role concepts we see that a role is represented by many agents and an agent consists of several roles. This concept allows the creation of an-hierarchy agent. In some situations a role can be also used as an indirect reference to the agent as well. The agent role is to perform **action.**

***Capabilities*** **.**As discussed before, capabilities describe the attributes given to an agent. The agent uses its capabilities to perform its role. The RE result shows the capabilities may be defined as the core agent or specific task, intelligence, pro-activity, communication acts combinations of these attributes. The common capabilities are functionality may provide to agent.

The following are the other concepts related to agents:

***Resource.*** Resources are related to role and indirectly, to agent as well. The term resource is used to represent the non-autonomous entities such as databases, external programs and specific functionality used by the agent, as shown in Appendix B(Figure B1.10). The resource modelling uses the standard object-oriented concepts.

***Goal.*** We differentiate two types of goals: first, the goal of the agent system and second, the goal of each agent. The former goal is called the system goal, or it can be described as the role of the agent system. Roles are assigned to agents. Therefore the system goal is achieved as the result of interactions of the agents. The latter goal is associated with individual agents. This goal is integrated with agent capabilities and agent states to represent the agent role. The system goal is identified through analysis of the requirements for how the system will play its role to achieve the system goal.

Some goals define the agent's identity, which derives from the agent role, while some agent goals allow the agent to provide service when it agrees with other agents to do so. For example, the state of the agent as 'waiting' that allows it to perform services.

***Task.*** This is a single activity associated with action to perform the agent role. An agent role may consist of several tasks. A task describes the pre-condition and post-condition based on the agent state. Task is indirectly used to perform agent roles. There are two types of tasks that generate action: communication act or direct communication. Direct communication is related to resources, while communication act refers to communication with other agents. An action generates on event. Changes of resource may also generate an event. On the other hand, agents also receive events by a similar process to action in order to generate the agent role.

***Plan.*** A plan is a concept that describes the sequence of agent roles performed to achieve the agent's goal. Two kinds of plans are identified. The first plan describes the sequences of the agents' interactions and socialisation to achieve the system goal. It describes which agents are initiators and trigger agents to interact with each other as shown in Appendix B(Figure B1.9).

The second plan is associated with an individual agent goal as shown in Appendix B(Figure B1.11). It describes what act the events capture and when, and links this to the nature and timing of agent actions.
The following are the concepts related to agent communication as part of the agent capabilities. It is captured from the communication act abstraction as discussed in relation to the RE process.

***Interaction Protocol.*** This describes a collective of participants to achieve a certain goal through several interactions. The aim is to present a consistent view of some aspect of the problem domain. The interaction protocol is defined as a pattern of agent interactions between two or more agents. The interaction protocol is useful, in analysis of requirements. It can also be used to define roles. For example FIPA-auctions involve the roles of buyer, seller and bidder.

***Message.*** A message describes the information that is exchanged between the agents while communicating or negotiating. It uses a communication act to communicate with other agents to present information. The message involves the use of agent communication languages that specify the sender and receiver, type of speech act, reply to the sender and indicate that the message content is sent.

*Agent Model.*

The overview of agent as shown in Figure 3.5 representing an agent model. The entities used to illustrate agents are the entities that need to be identified while designing agents. The notation of agent model is illustrated using Object-oriented model. We identify two perspectives of agent abstraction. First, it describes the relationship of the entities used to describe agent, while the second describes the agent design model the so-called agent model. The second agent abstraction is used to present the identity of agent. This agent abstraction is identified through the RE process. The static class diagram of the UserModelAgent of Filtering application as portrayed in Appendix B(Figure B1.10) shows that the agent can be abstracted into the following elements: agent name, roles and tasks involved in achieving the goal, as illustrated in Figure 3.6 (adapted the UML modelling).

The relationships between the elements illustrate that each agent is assigned a goal. A goal must be associated with at least one role and each role must consist of at least one task. A Plan is a sequence of roles associate with a goal.

| Agent name |
|---|
| Goal |
| Roles |
| Tasks |

Agent : Goal ==> 1: 1 , Goal : Roles ==> 1: 1..n , Roles : Tasks ===> 1: 1..n

Figure 3.6: Agent Modelling

The derivation process toward development agent environment shows that the design process is similar to the software architecture approach, which identifies agent environment functionalities then chooses the appropriate architecture to represent the functionalities, shown as an abstraction of components and relationship in Appendix C.

### *3.2.2. Software Development Process*

The derivation process has proved that the agent development process consists of two parts that may performed in parallel: *development of the agent system* and *development of the agent environment* as shown in Figure 3.7.

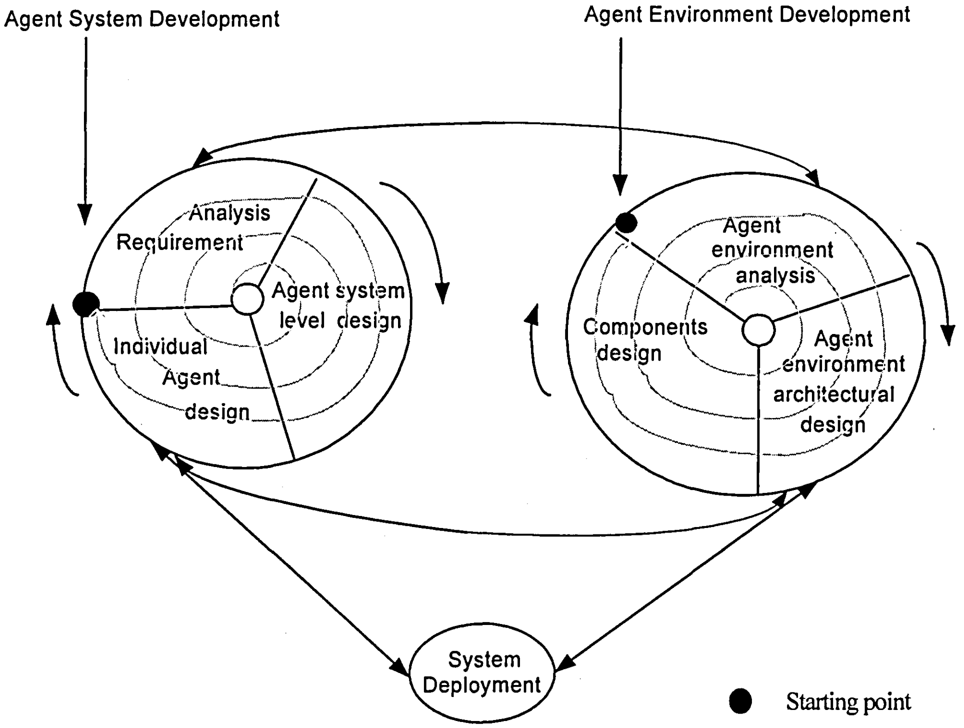

Agent System Development
Agent Environment Development
Analysis
Requirement
Agent system level design
Individual Agent design
Agent environment analysis
Components design
Agent environment architectural design
System Deployment
● Starting point

Figure 3.7: A Life-cycle of an Agent System Development

The first part consists of the following stages: *analysis requirement, agent system level design* and *individual agent design*, whereas the second part consists of the following stages: *agent environment analysis, agent environment architectural design* and *components design*. The third part is system deployment, which integrate both design process into executable agent system. The life-cycle for agent system development can be describes as a double spiral model. The description of each stage is summaries as below:

*Analysis requirement.* It is responsible for capturing the user requirements into an agent analysis model. The analysis process will transfers the user requirement into system goal, scenarios, use cases and identifies the agent four characteristics from the user requirement and identify an appropriate agent paradigm.

*Agent system level design.* The main activity is to identify agent (agent paradigm as identified at previous stage) to produce an agent system architecture that describes agents and their possible agent interactions in order to achieve the system goals. At this stage the focus is on how the system achieves the system goal. We use a system plan to represent the dynamic model and a system activity model to show the decision model. The modelling process once again uses the agent paradigm as identified previously.

*Individual agent design.* This describes the internal design for each agent in order to achieve the system goal(s). Each agent is assigned a goal. Each goal is assigned to a plan to achieve the goal. Each goal may consist of many roles. Each role represents an agent capability.

*Analysis agent environment.* The agent environment partly can be identified as the result of the previous activities and is derived from the user requirement specifications. It focuses on identifying the functional and non-functional requirements for development of the agent environment to represent the agent characteristics need to provide.

*Agent environment architectural design.* The agent environment architectural design is the process to identify the organisation of components to represent the architecture of agent environment. It aims to provide the functionality for developing agent system. Software architecture and architecture patterns are applied in this stage.

*Component design.* A low level design includes implementation of the components which mainly adapted component-based system approach.[12]. Design pattern is applied in this stage.

*Agent system deployment.* Agent system deployment focuses on integration of each agent with components identified in previous stages. The result of agent system level and individual agent design is refined into detail level of design. At this stage, it is possible to identify the actual number of agents and identify all agent communications. All agent communication is translated

[12] Component based approach is popular approach to develop reuse component for specific functionality[Bato+89][D'so+98][Schm96b].

into an agent communication language template, and the internal data structures are defined for internal agent processing, events for each agent and agent capability are defined. At this stage, each agent is designed in more detail. The design associates the components with the objects and classes to present the agent capability.

### *3.2.3. Analysis, Design and Modelling Techniques for Agent System Development*

#### 3.2.3.1. Requirement Analysis Modelling

We identify four analysis models appropriate for requirement analysis as follows: Event based Context diagram, System goal, Scenarios and Use cases. Each kind of modelling is discussed as below.

*Context diagram*

The RE of the Filtering application shows the continuity of interaction according to the planner, rather than returning back to the user to make decisions. This design concept has the advantage of agent independency, for example, the agent's ability to make its own decision. On the other hand, we use the basic concept of agent that uses sensors to capture any event that influences the flow of the system. The event may come from outside or within the system.

*System Goal*

The aim of the system goal is to present the agency of the agent system. We discovered that identifying the system goal is the most important criterion in differentiating between agent systems and non-agent systems. We also agree with various perspective presenting system goal such as organisation perspective, goal oriented, task or decision based perspective. However, the RE presented a certain level of objective rather than just objective. For example, the objective of the Filtering application is to provide articles from the web based on keywords provided. But the goal of the filtering application is to provide articles from the web *that are closest to the user's interest.* This criterion represents a higher level of objective. The system goal is associated with how to achieve it. We do not focus on identifying the user requirement, but rather extract and transfer the analysis requirement specification into a form that is able to provide a goal oriented analysis model for the agent system.

*Scenario*

Scenario is a precise method to capture the user requirements. Scenario is suitable for the user requirements when we do not capture the potential agent while analysing the requirement. As we mentioned in the previous section, the use of the role method is applied from high-level to low-level agents. In our methodology, we use a role-based scenario to present the user requirements to identify the agents at low level. The organisation perspective in GAIA methodology is suitable for agents at high level. The role-based scenario is described based on the events captured. The scenario should match with the system goal.

*Use Case*

Use case is a useful technique for analysis user requirement close to design stage. We adopted the existing use case proposed by [Jacob+95] as a primary analysis of the user requirement. There are two types of use cases. One type of use case is called an *Essential Use Case* and the other type of use case is called a *Real Use Case*. The Essential use case may be expressed as :

> '*is an ideal form that remains relatively free of technology and implementation detail; design decisions are deferred and abstracted, especially those related to the user interface.*'[Cons97]

and the real use case as

> '*concretely describes the process in terms of its real current design, committed to specific input and output technologies, and so on. When a user interface is involved, they often show screen shots and discuss interaction with the widget.*' [Larm97]

We chose to use the use case method to analyse the requirement, because the use case has several advantages [Heid+92][Larm97]. However, we viewed and developed the use case differently. Traditionally object-oriented use cases are developed focusing on the use case of an actor and cases related to an actor. The agent use case analysis in this work, in contrast, is developed based not only on the actor but also on the events assigned to the actor. The event may come from the external environment or from inside the system.

The use case of the Filtering application in the Appendix B(Figure B1.5) is the result of the use case based on the traditional object-oriented use case that will match the agent design solution. So, we propose to use the role based use case. At this stage, agents are not identified. The agents are identified by assigning the roles to agent. This use case can be described as an essential use case. The concrete use case is when the agent is identified and the roles are assigned to agents. For us, the concrete use case comes at the agent system design stage, as discussed later.

### 3.2.3.2. Agent System Design Modelling

The RE result shows two kinds of models appropriate for modelling agent system design: Agent system architecture and System Plan. Each kind of modelling is described as below.

### *Agent System Architecture*

The agent system design is known as the real use case. It is identified based on agent use case and agent model. The agent system design describes the agents and their interaction. This design stage is abstracted from the RE of the Filtering Application as shown in Appendix B(Figure B1.9). The figure shows that the Filtering Application consists of several agents that interact with each other in order to achieve the global system goal. This static class diagram is abstracted into a form of a high level design by omitting the classes related to each agent incorporated with the abstraction of agent model. The associated classes for each agent are transformed into individual agent design stages.

The modelling notation of the agent system design is proposed using a combination of high level conceptual design of DESIRE[Treu+98], allied with a software architecture approach[Shaw+96], in which the software architecture community defines an agent as a component of a system.[13]. The conceptual design shows a similarity with the central design of the NewsFilter application.

The components defined in the detail design of DESIRE modelling are similar to some of our components for designing agent environment.

The advantage of the agent system design is its ability to portray the architecture of the system. It is able to disseminate the conceptual distribution and physical distribution. The Filtering application class diagram shown in Appendix B(Figure B1.8) is abstracted to present a high level design of Filtering application as portrayed in Figure 3.8. The diagram shows the architecture of the Filtering Application solution including the decision of locational distribution(dotted-line - agent similar location). Its presented agents and interactions, resources and goals. The decomposition into adjacent levels uses similar techniques to the DESIRE methodology, but the decomposition of each agent is focused only on internal agent design and communication with other agents.

[13] The DESIRE conceptual design is different from our agent system design, even though the design shows distribution criterion, it is based on centralised design

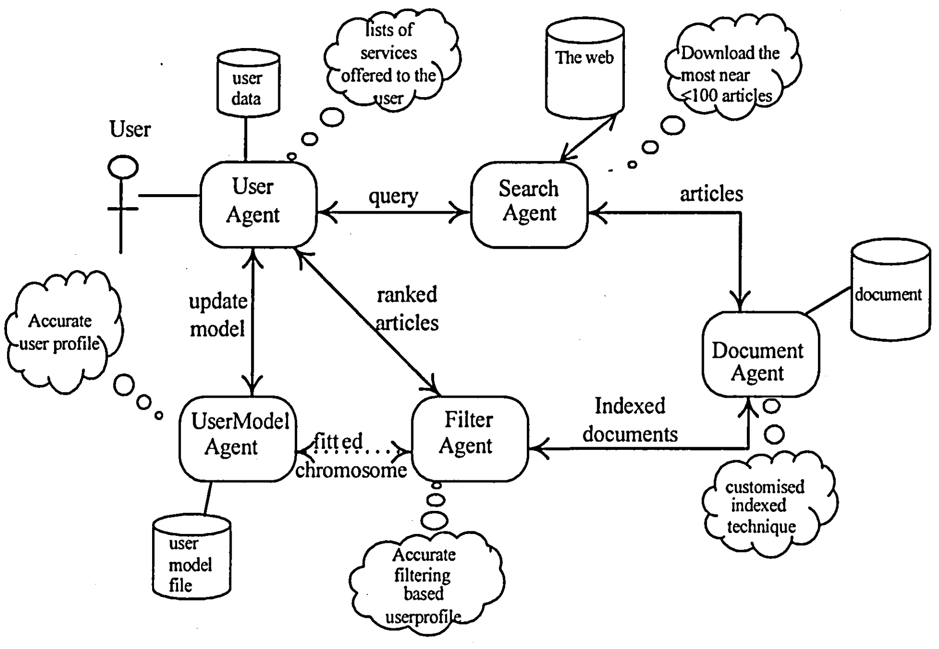


Figure 3.8: The Agent System Design for Filter Agent System.

The diagram shows the agent system architecture model produced in the agent system design stage. The modelling shows an abstraction of agents and interactions. The model shows each agent presented by its agent's name, resource and goal. It shows the external events as initiators to generate the system. The model also shows the distribution location as well. This agent model represents the agent system solution.

*System Plan*

The agent system architecture is only able to present the static agent interactions. We need to have a suitable modelling technique to present the dynamic modelling at the high level of system design.  We chose the UML sequence diagram to present this purpose. The agent interaction shows interaction between agents, not objects.

Appendix B(Figure B1.9) shows the Filtering Agent System sequence diagram. This diagram is able to show the plan of the system. It shows the events that occur and presents the sequence of the agent interactions. It describes the cycle of the system, from start to finish. This diagram represents a global plan of the system. The system plan is associated with the system goal.  An agent system may consist of more than one goal. We can also use an activity diagram to show part of the system plan but the advantages of an activity diagram that is it able to represent the general decision-making that occurs inside the agent that causes action or interaction with other agents.

### *Agent Interaction*

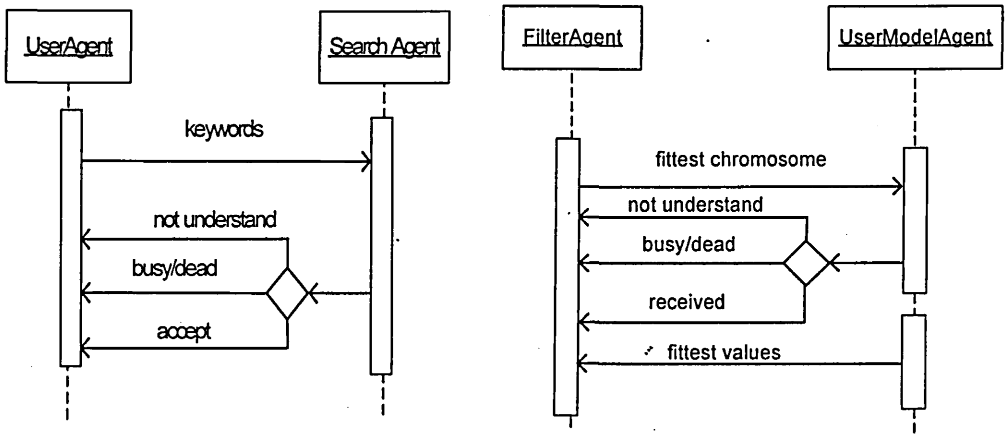
Agent interaction is a part of presenting the agent roles. It is a concept used to describe the agent interaction between two agents and the consequence of such interactions. The agent interaction is able to provide a clear technique to define the agent interaction and hinder the conflict of several approaches of agent interaction such as cooperation, competition, etc. and utilise concurrency. We propose agent modelling to present the agent system design.

Even though the previous section has described the static and dynamic modelling to present the high level architecture of the system, it has focused only on the global agent interaction. Here, we discuss the concept of agent communication as a basis of interaction to present the socialising agents in order for each agent to achieve its own goal. The literature review on agent socialisation reveals several types of agent interactions [AIP99][Aare+96][Barc+96] so that we can classify the basis of interaction between two agents.

The term communication describes a one-way communication between two agents and agent interaction is two-way communications that continues until both agents are satisfied or have achieved their goals. There are several different patterns of agent interactions. The selection of the appropriate interaction type is a criterion of a good design. The agent interaction describes the agent interaction patterns between two agents so each agent achieves its own goal.

The Figure 3.9 shows a few examples of agent interactions, expanded from Figure 3.7, but focused on interaction between two agents. We have not invented a new way of modelling interaction but follow the James Odell's agent interaction modelling as shown below [Odel+00][Odel+01a].

The first agent interaction in the figure shows that the UserAgent keeps sending the message until the Search Agent confirms that it has received the query and understands the message. This example presents as a basis FIPA-Inform agent interaction pattern. In this cases, understand, ready(not busy or dead) and accepts criteria are part of the internal SearchAgent to produce further action. The UserAgent will be notified whether the SearchAgent is accepted, not understood or busy.



Interaction Modelling for CA-1 Interaction Modelling for CA-5

Figure 3.9: Interaction Modelling Between Two Agents

The second agent interaction shows that the FilterAgent requests the fittest chromosome values from the UserModelAgent. The figure shows the agent interaction using the FIPA-request interaction protocol. The UserModelAgent has to make sure it understands the request and UserModelAgent will reply to the fittest chromosome request as soon as possible.

*Individual Agent Design Modelling*

The individual agent design focuses on designing the internal agent. Individual agent design is a decomposition of agent abstraction to present the identity of the agent as shown in Figure 3.9.

The RE process of each agent is able to identify the individual agent design stage and identify the modelling are needed. In the RE process, we identify two types of agent model to present the individual agent design stage: *Individual agent model* and *agent plan model.*

*Individual Agent Model*

The agent model is captured from the diagram portrayed in Appendix B(Figure B1.8). It shows the static class diagram of Filtering application consisting of five agents. The class diagram of each agent represents the lowest level of individual agent design. This class diagram is abstracted. We found that each agent is associated with its goal. The internal design is focused on how to achieve the agent goal. Each agent is associated with at least one role. In the Filtering application the UserAgent has two roles, while the other agents only have one role. Each role is associated with tasks. We identify two types of tasks: tasks for interaction with other agents and internal computing tasks. Let us take as an example the UserModelAgent as shown in Figure B1.11. We found that the UserModelAgent can receive three kinds of receiving messages: from UserAgent in login and providing keywords; from FilterAgent to provide the fitness of chromosomes; and from UserAgent while providing feedback an articles' value. The internal computing tasks are to create the user profile and apply the genetic algorithm while creating or updating the user profile. The UserModelAgent is modelled as Figure 3.10.

| **UserModel Agent** | | | | |
|---|---|---|---|---|
| **Goal:** | Provides accurate userprofile representative | | | |
| **Roles:** | Usermodel | | | |
| **Tasks** | **Collaboration** | **Communication** | | |
| | | **Name** | **Message type** | **Content** |
| Send message | UserAgent | CA-1-2 | *Inform* | *User login name or new user* |
| Create initial user model | | | | |
| Send message | UserAgent | CA-4.2 | *Request* | *Update user profile by using score value* |
| Update user model | | | | |
| Receive query | FilterAgent | CA-5.1 | *Request* | *findfittest chromosome* |
| Apply genetic algorithm | | | | |
| Send user model | FilterAgent | CA-5 | *Inform* | *Best chromosome vector* |

Figure 3.10: A UserModelAgent Model

These three received messages are associated with one message to be sent out as a reply to the request for fitness of chromosomes from FilterAgent. Each message may be received or associated with the internal agent processing.

*Agent Plan Model*

The investigation of each agent class shows possible events that may occur to agents. How the agent conducts the events and presents the type of action depends on event. This condition describes the plan of each agent. We decided to use a state diagram to present the internal agent plan, which we called the agent plan model as shown in Appendix B(Figure B1.11). The UML sequence diagram is also used to present dynamic interaction between objects to present the dynamic internal agent plan model.

*Agent System Deployment*

The UserAgentModel static class diagram shown in Appendix B(FigureB1.10) is an abstraction of an agent class model at lower level of design. This class diagram represents the combination of UserModelAgent design with components defined by the Jade agent environment. Therefore, we define another stage after the agent development phase and agent environment development phase to integrate these two phases of development, which we call the *agent system deployment* phase. The propose agent system method shown in Figure 3.6 illustrates how the *agent system deployment* phase is located in the agent system development life-cycle. The agent system deployment adapted UML modelling. The Object-oriented concepts such as inheritance and instantiate are used at this stage. The class diagram in Appendix B (FigureB1.10) defines a detailed design level, which can be transformed automatically to an implementation stage.

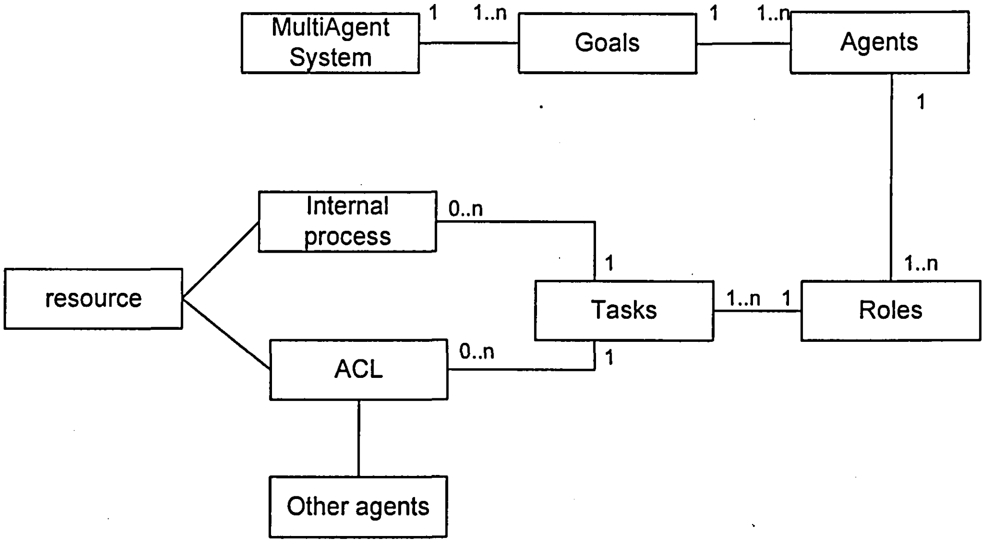


Figure 3.11: Agent Design Framework

Based on the previous discussion, we are able to identify the relationship between multi-agent system goals, agents, roles and tasks as shown in Figure 3.11. The relationships are extracted from Appendix B(Figure B1.8) and the agent overview(Figure 3.5). The relationships between these elements provide a framework to design a multi-agent system.

A multi-agent system consists of at least one goal. The Filtering application has one agent system goal. The forward engineering of Filtering application in Chapter Six shows the two agent sub-goals lead to the achievement of the agent system goal.

The interactions between agents are assigned based on agent roles. An agent role may consist of several sequences of tasks due to the autonomous behaviour. Two types of tasks are classified: tasks used to communicate with other agents and tasks associated with internal operations: specific functionality (e.g. calculation, applied intelligence, fuzzy logic, etc.) or communication with resources(i.e. databases or sensor to the external objects, i.e. the Web )

The framework describes how a multi-agent system may consist of several goals. Each goal consists of several agents that interact with each other in order to achieve the system goals. There are several issues of sub-goals, for example related to the achievement of intelligence as part of rules, but these goals are not our concern in this research. We believe it is part of a process of a rule-based approach.

### *3.2.4. Analysis, Design and Modelling Techniques for Development Agent Environment*

The RE and SEAF process found that agent environment can be different in two issues: functionality are provided, architecture of agent framework that illustrates the components and their relationships. Its involves the choice of technique used to implement the functionality that lead the choice of different component are used[14]. Therefore we propose the three development agent environment consists of three stages: *agent environment analysis, agent environment architectural design* and *component design.* In this section, we extract the common issues of agent environment requirement, including the technologies used to develop the system which shows the agent environment reference architecture. The rest of the sub-section is mainly related to the issues of designing the agent environment architecture. The SEAF process has the use of framework, architecture pattern and design patterns in architectural design. Therefore, we believe these software abstractions may be used for development agent environment.

#### 3.2.4.1. Agent Environment Analysis

The agent environment analysis is the first stage of development agent environment as shown in the overview of the agent development life-cycle in Figure 3.7. The literature review on the agent environment revealed the two issues of physical and communication environment for agent environment development, while from RE of JADE and SEAF processes we found the three key issues for agent environment development, which become a key reference for agent environment requirements (see Figure 3.12).

In order to support the various physical environment requirements of such an agent based system, a common processing platform is useful. This platform provides a foundation upon which agent system applications could build to leverage their own specific environment requirement. Agents may be implemented as hardware, software or a combination of both. The scrutiny of the existing frameworks shows three basic layers to capture the requirement for development agent environment:

[14] These problem domain are similar to the development of object-oriented framework as discussed in chapter two.

- **Application layer** contains the applications for all management and support services for entities supported by the environment. This includes the mobility, directory, ontology services, query, security, firewalls and interaction protocol.
- **Communication and transportation layer** defines the routes, verifies and transmits data required from the application layer. This design provides a general-purpose service that has no dependencies on applications and type of data.
- **Physical layer** specifies the physical and electrical characteristic of the bus. It involves the hardware that converts the characteristics of messages into electrical signals for transmit messages and electrical signals into characters for received messages. This will include a standard physical interface such as controller, sensor, actuator and receptor.

On the other hand, the scrutiny captured the requirements for the communication environment. In order to enable the agents to interact productively, in development of an agent environment, consideration should be given to the following issues.

- Interaction management-used to manage the interaction among entities to ensure that they conform to the selected agent interaction protocol. At the environment level the control of the interaction management is identified as AIP-manager agent.
- Language processing and policing- to make sure that the language parses the information correctly, semantically and in the appropriate agent context.
- Co-ordinate strategy services that describe the strategy of how an agent is launched. There are two types of coordination strategies. The first is *directory services* providing methods by which an agent is located, as captured in the scrutiny process: white-pages (individual)(in JATLITE, JADE and JAFMAS), yellow-pages (industry/group)(in JADE and JAFMAS) or green-pages (offered services)(in JADE and JATLITE). A good agent environment should consist of all of these services. In terms of physical environment, this directory is used to provide information on where the agent is physically located, describes the boundary of the agent's social environment and provides information on the agent role's or the services provided by the agent. The white-pages directory is used to provide information about where the agent is physically located. The yellow-pages describe groups of agents which have a similar business type. The green-pages is used to launch the agent services that can be reused by another agent. Green-pages is an entity that provides openness to the services level. Any agent can reuse any agent services registered in the green-pages. The second is *mediator services*. This acts as an inter-mediator among agents. Several inter-mediator agents may be established in an agent environment to act as communication intermediaries for transaction management activities or ontology translation.

- Policy enforcement services control the agent by its environment or social group. This service provides the possible mechanisms for enforcing policy mechanisms whether among agents or supporting services for non-conforming agents[Wool+00]
- Social differentiation describes the separation of the socialisation process by grouped agents. The successful differentiation is dependent on how the groups are created in such a way that an agent can play multiple roles in multiple groups[Wool+00][Zam+00].
- Social order is the result of production of the structure of relationships among social agents [Caste00]. Social order is identified as the result of formal policies and may also emerge via self-organising mechanisms. This means a system evolves its own social pattern, as in the case of the stock market. These social patterns, at first, from managing the condition of society as a whole, employing non-accidental or non-chaotic patterns of interaction. For example auctions employ strict social patterns. Such mechanisms can be employed to control undesirable emergent patterns that need to be resolved. For example when the stock prices rise or fall by too many points in a session, trading curbs are triggered[Chav+96].

The following sections discuss the issues related to designing the agent environment as a second stage of the agent development life-cycle (Figure 3.7). We capture that the process of development of the agent environment involves is to designing the architecture of the agent environment, which we can see as a recursive process to structure the components captured from the requirements analysis, using patterns as a guide. The architectural design is an art. The following section discusses the factors that influence architectural design, the types of modelling needed to present the architectural design and the role software architecture, framework and patterns, as captured through the scrutiny and RE of those elements in existing agent frameworks.

### 3.2.4.2. Agent Environment Analysis and Design Modelling

The derivation processes were able to us abstract the five types of models are needed for analysis and design agent environment. These models present several views of modelling.

- *Organisation model*
- *Agent model*
- *Interaction Model*
- *Task Model*
- *Domain Model*

Examples of those models are shown in Appendix B. The organisation model describes the organisation of agents to present a solution to agent coordination as shown in Figure Appendix C3.1(JADE), Figure Appendix C2.1 (JATLITE) and Figure Appendix C1.1(JAFMAS).

The Agent Model describes the agent architecture of the agent system solution. It presents the details of agent abstraction. Scrutiny revealed the details of the agent model as shown in Figure 5.22(JADE), Figure Appendix C2.5(JATLITE) and Figure Appendix C1.4(JAFMAS). This agent model describes the organisation of components.

The Interaction Model is identified from all agent interactions presented as the result of agent system design(e.g. FIPA interaction protocol). The interaction model shows relationships between agents, agent task and agent organisation to handle agent messages.

The Task Model is identified based on the analysis of all agent tasks to present the capability of agents, such as the capability to query agent status, capability to communicate with other agents, capability to listen to new events, capability to refuse any request when busy, capability to use expert rules and mobility. These capabilities are used to perform the agent's role.

Another model called Domain model is necessary. This model presented the different domain of agent interactions as presented as ontology. Different agent domain presents different domain that leads on identification design of the ontology of agent interaction.

### 3.2.4.3. Design of the Architectural of Agent Environment

We discovered that the main activity of agent environment development is architectural design. The SEAF process shows the architectural choice of each framework. The JAFMAS architecture framework is developed based on COOL architecture[Cool99]. The JATLITEbeans architecture framework is developed based on Internet client/server architecture and JADE is developed based on FIPA architecture[FIPA+97a], in which FIPA architecture is based on CORBA architecture. We found two properties that influence the agent environment architectural design, similar to the two types of environment physical and communication found in the literature review. The following paragraphs discuss the issues that influence agent architecture design as part of the abstraction of the physical environment and how agent coordination influences the architectural design as communication environment abstraction.

*Agent Architecture*

The agent architecture is identified either as the result of agent system design or as stated in the requirements. The agent architecture describes how decision-making is performed. As a result of the scrutiny process, we found that each agent presents different agent architecture. Even though all the frameworks present similar types of behaviour capability (autonomy and reactivity), the refinement of the agent architecture design is influenced by the four agent characteristics

- Agent architecture is mainly identified based on the abstraction of minimum behaviours that may be present in all the agents being identified. These minimum behaviours become the paradigm of the agent solution, used in refinement of the agent system design. Then additional behaviour identified as the result of agent system design for each agent is added to these minimum behaviours. At the agent design stages, the agent abstraction is enough to identify the minimum agent behaviours present among the agents, but in the designing agent architecture, the focus is on the mechanism by which the shows agent shows its capability. The following points influence the identification of the different types of components to be integrated in the agent architecture.
- Distribution influences the part of other components in integration with the agent architecture; whether the agent is only focused on conceptual distribution or locational distribution and whether the agents are heterogeneous or homogeneous.
- The choice of communication channels used for sending messages among agents influences the identification of the platform environment as part of the agent architecture.
- The criterion of openness also influences the identification of other components that influence the structure of the agent architecture. The issue of providing open architecture is discussed in further detail in [Dale01][Brad97b][OAA01][Mart+99].

*Agent Coordination patterns*

The agent coordination patterns, has influence the choice of agent management. The scrutiny of the existing agent frameworks showed different methods for agent coordination. JAFMAS organises the agent coordination according to a hierarchy of a group of agents as shown in Figure Appendix C1.1. Agents with similar services are grouped in the same group. With such organisation, JAFMAS defines a special class as a facilitator agent that manages the agent address. The facilitator agent provides a directory service to all agents.

JATLITE only provides a group agent to which all agents are assigned. It specifies a router server that allows all agents to be registered. This router server provides a directory service for the system. Any agent can communicate with the agent by browsing any agent available in the server. Figure Appendix C1.4 shows the centralised organisation of agents.

JADE presents a similar organisation agent to JAFMAS. It provides decentralised agent management. Each location provides its own agent management services as directory services. Agent groups can be created to differentiate between similar agent services. However, JADE presents an additional functionality as green pages, on which the agent services are launched so that they can be accessed by any agent, whether registered in the same agent environment or in a different agent environment.

### 3.2.4.4. Components Design

Components can be identified in various levels of abstraction. So identifying components as discussed in previous sections is performed recursively when designing architecture. This is because agent environment architecture is represented a structure of components that aim to performs various kind of functionalities. Component design used traditional object-oriented development guided by design pattern. At this stage, the idiom[15] is useful as guidance to implement the component.

The SEAF process has identified the components relevant to agent technology, as shown in Figure 3.12. We have abstracted the three high level abstractions for agent environment development: agent, inter-agent communication and agent management, as captured from the scrutiny process and RE. The combination of the scrutiny process in the previous section and literature review of existing architecture patterns and design patterns enable us to produce the agent environment reference architecture associated with the three agent environment abstractions. The figure shows the techniques used in JADE, JATLITE and JAFMAS marked with the signs '*', '+' and '#' respectively, associated with those three abstractions. The figure shows that the requirements and choice the components will influençe the design of the agent environment architecture.

Each component may be decomposed into smaller components. In Appendix B, Figure B2.1 shows CIAgent component while Figure B2.4 shows JADE component. In Appendix C, Figure C1.3 show JAFMAS components and their relationship, while Figure C1.4 shows the decomposition of JAFMAS Agent component, Figure C2.3, C2.4 and C2.5 shows some components of JATLITE framework.

[15] Implementation based design patterns[Gam+95].

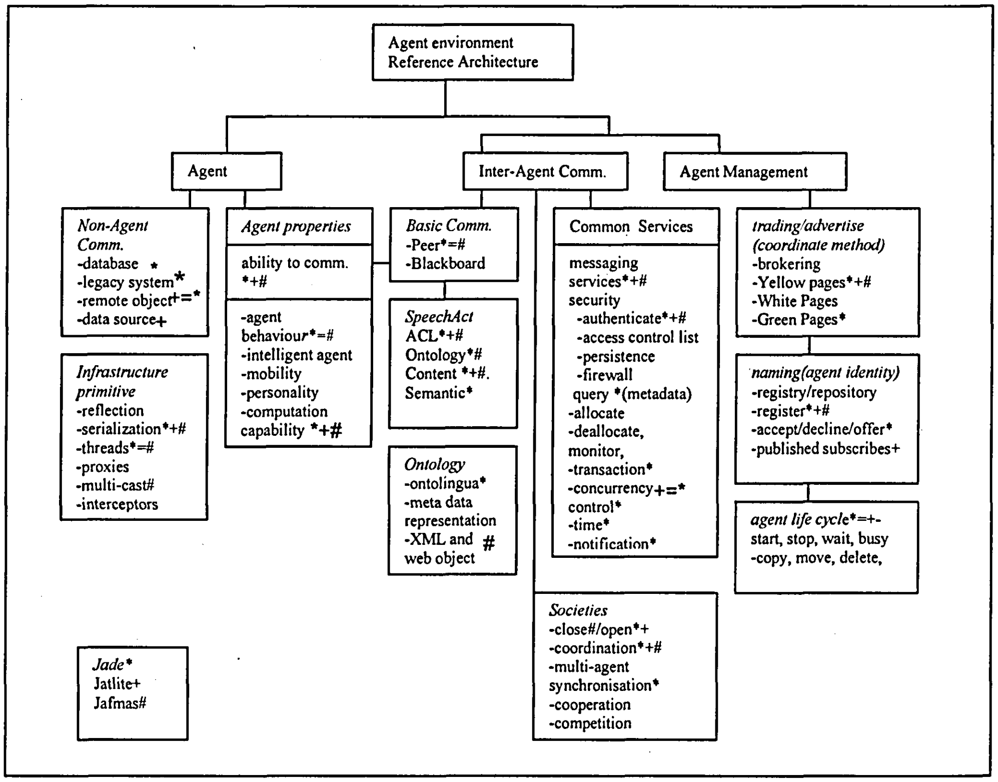


Figure 3.12: The Agent Environment Referent Architecture

These components shows that agent environment is to develop architecture that consists of components and relationship and it also show components decomposed into sub-components.

### 3.2.4.5. The Used of Software Abstraction in Development of Agent Environment

The SEAF captured several abstractions and sub-abstractions from different angles as components of the agent environment.

- There are at least three layers of abstraction for agent environment: application, communication and physical. The scrutiny in Appendix D, Table D1.1 shows these three abstractions. This abstraction called layer abstraction.
- The application layer is abstracted into three abstractions: agent, inter-agent and agent management (see Figure 3.12). This called vertical abstraction.
- The agent component is an abstraction of the agent architecture, which consists of three layers of abstraction: interface, interpreter and collaboration. The interface layer consists of three sensory abstractions: the sensor adapter, the database adapter and the effectors adaptor. These sensors captured internal, external or inter-agent events respectively. The interpreter layer is abstracted into three components: capability, actions and knowledge abstraction. This abstraction means that the agent takes action

based on its knowledge and capability, which are attributes similar to the abstraction found in the Figure 3.8. The collaboration layer is part of the inter-agent abstraction. Appendix B-Figure B2.4 shows JADE component, B2.1 shows CIAgent component, Appendix C-C2.5 shows JATLITEbean component.

- Inter-agent abstraction consists of two communication abstractions: Content language and performatives, and ontology. This abstraction is based on the features of agent communication discussed in Chapter two. JAFMAS and JATLITE only support the content language and performative but not ontology, while JADE support all inter-agent abstraction.
- Agent Management can be abstracted into three abstractions: inter-communication services, inter-agent mediator and security services. The inter-agent mediator services is the high level abstraction. In Appendix C-Figure C1.1 shows JAFMAS organisation agent, Figure C2.1 shows JATLITE organisation and Figure C3.1 shows the JADE organisation agent that leads the choice of agent management. The inter-communication services is close to the technology choice for interaction such as JAFMAS use socket UDP management, JATLITE uses client-sever inter-communication services while JADE use CORBA inter-communication service management. Security services are closed to the choice of technology as well.
- The use of Framework. Appendix D-3 shows that agent environment is developed using object-oriented framework approach. Figure D3.1 shows an example of a structure of framework, which consists of components and interactions that can reuse.
- The use of Software architecture and Architecture Patterns. The identification software abstraction process has identified the used of software architecture and architecture pattern in existing agent framework (see Appendix D-1, Figure D1.1). It compares the current architecture pattern [Bush+96] with the current agent framework architecture.
- The use of Design Patterns. Design pattern is identified while RE process. In Appendix B, Figure B2.1 shows how design pattern is captured. The figure shows the CIAgent components consists of various kind of design patterns such as Composite patterns and Reactive Patterns. Similar patterns are found in Figure B2.4 in JADE component. Beside that, design patterns also apply for structuring the agent environment architecture. In Appendix D.2 shows the used of current design patterns in development the architecture of JAFMAS, JATLITE and JADE.

The derivation processes discussed above show the use of various software abstraction to reduce the complexity of development process. It consists of several levels of design abstraction. This method provides technique to reduce the transaction from high level components into a lower level design. It clearly shows that the use or software architecture and architecture patterns applies for high-level design, while design patterns applies for lower-level design. Since the main process of agent environment development is to identify components, the next stage of development is design the components as recursive process.

## 3.3. SUMMARY

This chapter has discussed the derivation process and presents the essences of derivation result towards development of agent method. The derivation processes involves integration of various derivation activities to the produce the proposal of a method for developing an agent system, which divided into two parts: agent system development and agent environment development. Four of the derivation activities involves four type empirical works: analysis existing agent-oriented methodologies, reverse engineering, scrutiny of existing agent frameworks and examining the used of software abstraction in the current development. How each derivation activity is performed is also discussed. The essence of each derivation processes also presented (in Appendix A to D).

The essence of derivation result is presented. It aims to answer the three attributes of a methodology: the key concept to illustrate agent (agent model), life-cycle of development process (double spiral model of three phases and each phase consists of three) and modelling techniques for agent system development (such as plan model, system goal, scenario, agent model, use case and agent system architecture) and agent environment development (organisation model, task model, interactions model, domain model and agent model). The description of each model is described briefly as well.

We show the development process of development agent environment by identifying the common agent environment functionality that leads to the choice of architecture design (references architecture) and shows the role of software abstractions in designing architecture of agent environment.

In the next chapter these essences of derivation result are used to develop the proposed of agent oriented method.

# CHAPTER 4

# 4. Towards a Method to Develop an Agent-based System

## 4.0. INTRODUCTION

This chapter presents a method to develop an agent-based system consisting of development process and modelling processes. The development process describes the flow of development process, stage by stages, while the modelling process discusses what kind of modelling is needed at each stage and shows how to construct them. The essence of derivation results presented in Chapter 3 (Section 3.3) is the requirement to understand the method. The description in this chapter focuses on the modelling process and how modelling is developed rather than discussing the concept of modelling itself.

The four characteristics of agent systems are used throughout the development process. [O'Mall+01] provides more criteria of agent use. At an early stage of analysis, designers must justify an appropriate approach to develop the system such as single agent[1], mobile agent[2] or multi-agent system. The method presented in this chapter focuses on the development of a multi-agent system approach.

The agent development method covers the whole process of development agent system from analysis, design and implementation, and provides a modelling process from analysis to design stages, shown in both paths of the development process. The development process encompasses two layer abstractions. The first layer presents a recursive process of three following phases: *agent system development, agent environment development* and *agent system deployment* as

[1] focus on agency to agent (such as autonomous, reactivity, pro-activity, intelligence, etc.), but do not presented social-ability. If there is, but limited to changes in the external environment or resources such as the web, which are observed through a sensor or an interface. The RE of Newsfilter application is a explicit process to develop a single agent.

[2] special case of a single or multi-agent system approaches. It presents a different nature of communication due to the locational distribution criterion, which used for reasons of transaction performance, such as to reduce the traffic in network transactions. Instead of the agent sending messages and returning the result while interacting, the agent itself moves to the destination where the other agent or external environment is located.

shown in Figure 3.6. The second layer is the decomposition of each phase, called stages, as the relationships between those phases show in Figure 4.1.

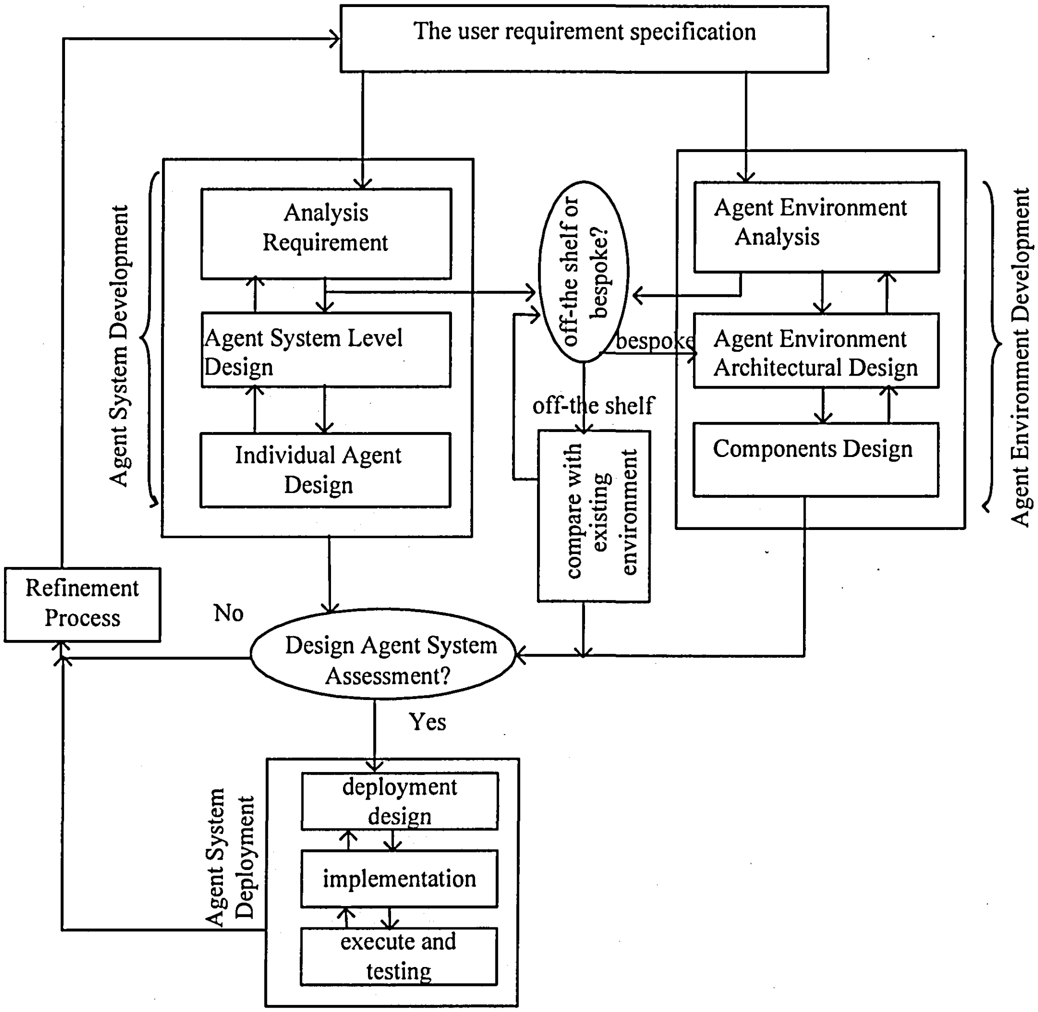

The user requirement specification
Agent System Development
Analysis Requirement
Agent System Level Design
Individual Agent Design
off-the shelf or bespoke?
bespoke
off-the shelf
compare with existing environment
Agent Environment Analysis
Agent Environment Architectural Design
Components Design
Agent Environment Development
Refinement Process
No
Design Agent System Assessment?
Yes
Agent System Deployment
deployment design
implementation
execute and testing

Figure 4.1: The Artefacts of Agent Development Method

The following describe briefly the aims of each phase:

- *Agent system development (ASD)* focuses on the solution of the agent system. It focuses on what the agent system looks like, and how the design agent system achieves the requirement of the system. It mainly focuses on identifying agents and interactions, similar to identifying objects and their relationships, in an object-oriented approach.
- *Agent environment development (AED)* focuses on where to execute the agents. The main activity is identifying the functionality and technology to build the architecture of the agent system. Figure 2.1 in Chapter 2 shows the elements needed to produce an agent environment. It focuses on development of reuse components and the

components, which provide functionality similar approach to development of an object-oriented framework application.

- The *agent system deployment (ASDO)* is an integration of the two previous phases in order to perform functionality of ASD. The agent environment functionalities (components) are deployed with all the individual agent design. It applies the object-oriented concepts such as inheritance and instantiation. The execution of all agents in parallel fulfils the requirement of agent system.

These three phases represent different stages of development process, and different analysis and design methods. This guides the designer's thinking when developing agent system. It represents the whole life-cycle of agent system development. The development process can be illustrated as a spiral model. It may starts from the ASP, AED and ASDO phases in sequence. The spiral model combines the positive aspects of the waterfall model and the prototyping model. This combination is a form of a rapid development version of software development process. This illustrates a series of steps in each phase. The rapid development process here means rapid development within the stages in each phase. The changes in any phases may influence the changes of other phases as well. For example, the changes of version of agent environment by adding new requirement may change the ASDO. The figure also illustrates that the development of agent systems can be start from the AED phase, followed by ASD and then ASDO, when it utilises the reuse of agent environment.

This means, the same agent environment can be reused for the execution of a similar kind of agent system. The choice of the route depends on the use requirement. However, in reality the agent development life cycle shows a *bi-spiral parallel* process of the first two phases. The parallel process can be seen as providing the common agent environment requirements for analysis agent environment. Chapter 3, section 3.3.4.2 (agent environment analysis) provides a list of requirements, which should be provided to agent environment, while in Figure 3.12 shows the agent environment reference architecture that provides the technology choice for designing agent environment. Changing some parts of the spiral model produce changing in other parts. For example, the changes in agent system design may change the architectural design. The changes in architectural design may also identify a new component, remove certain unnecessary components, or change the component design that may change depending on what requirement was changed. For example, if the type of communication channel is changed, this may not change the agent system design. Similarly, any changes in ASD may change the deployment agent phase, for example, the changes of agent interaction will change the ADSO as well.

The figure shows the artefacts of design activities in each phase and shows the relationship between them. The use of different levels of abstraction in each stage of the development process makes it possible to present the deliverables of specifications, documents and modelling in each stage. This method is able to show a clear transformation between the stages.

The ASD process is performed until an optimum of design agent system is produced; while the agent environment phase is performed until it produces an executable component to reuse. The component provides the functionality that is used by agents to perform their roles.

The development of agent system applies the refinement and contingent approaches. Both ASD and ASDO apply the refinement approach, while AED applies the contingent approach. The former approach shows the refinement from a high-level abstraction to a lower level abstraction, while the latter approach uses several views of design modelling that link together to represent the agent environment design modelling.

The following discusses briefly the modelling process in each stage.

**The first phase** presents a recursive process consisting of the following three stages: *Analysis Requirements (AR), Agent System Level Design (ASLD)* and *Individual Agent Design (IAD).* The modelling techniques of system goal, scenario, use case, agent system architecture, agent system plan and agent data model are used throughout this phase. The agent concept as discussed in Chapter 3, Section 3.3.4 is the core concept used throughout this phase. The *agent system design artefact,* discussed later presents the transformation and deliverable models in each stage. It also shows the transformation from each design model to another model, in different stages. Some models are developed using a specific rule, whilst others need to be justified.

The result of the ASD phase is assessed to produce an optimal design for an agent system. There are several issues to be considered to produce an optimal design of agent system as discussed in the refinement section. The refinement is carried out in order to fulfil the four agent characteristics. At this stage, the optimal design is based on the user requirements and the agent paradigm. After the AED phase, the agent abstraction is clearly identified. It includes aspects of the communication environment, such as media communication, and the physical environment, such as agent platform. The refinement of design of the agent system follows the ASD phase. At this stage, it is possible to identify the total number of agents.

**The second phase** presents a similar life-cycle process, which is an iterative process consisting of the following three stages: *Agent Environment Analysis (AEA), Agent Environment*

*Architectural Design* (AEAD) and *Component Design (CD)*. The AED phase represents a different approach to the development process. The central process of the AED phase is architectural design, which is developed based on the agent environment requirement. The components design is a lower-level architectural design of each component, which ties in with implementation as well. The AEA aims to produce the requirements of the agent environment. In Chapter 3, Section 3.3.4 presents the five viewpoints of models for analysis of the agent environment that used to design agent environment architecture. The method uses an object-oriented software abstraction, such as architecture pattern, frameworks, design pattern and UML modelling to develop an agent environment and as discussed in Chapter 3. In this chapter, we present the agent environment criteria leading to the identification of components.

Besides that, we also propose the development process or guideline for reusing an off-the-shelf agent environment. It proposes the elements needing to be analysed when more than one existing agent environment meets the agent environment requirements, and discusses how a decision is made.

**The third phase** represents the actual integration process of each agent design with the components of the agent environment. At this stage, the agent model is abstracted at the object level. The object-oriented concepts such as inheritance, association and generalisation are used in the deployment process. The UML notations such as a static class diagram, sequence diagram and collaboration diagram are used to show the internal design of each agent. The agent abstraction at object level (based on JADE agent abstraction) shows the deployment process.

The agent modelling method is similar to the object-oriented approach. An object is modelled according its attributes and methods, while the agent is modelled according to its goals, roles and tasks, as shown in Figure 3.6 (Chapter 3).

The AR elicitation aims to identify the list of goals and roles as the core elements to identify agents using modelling as proposed in Chapter 3. The roles are associated with agent goals, capabilities and resources. The analysis of the roles is able to identify the capabilities needed to perform the roles and vice-versa. Identification of the role and capabilities are the precise way to identify agents and their interactions at ASL design. Those agent attributes are refined in more detail at the IAD stage.

The decision of the basic behaviours that present the agent capabilities is defined as an agent paradigm for the solution of the agent system. It should decide after the analysis requirement and before ASLD.

The description of agent properties as stated in Chapter 2 presents various kinds of behaviours as possible capabilities assigned to agents to enable them to perform their agent role. This combination of agent behaviours shows that agent approach presents a dynamic paradigm (flexible). For example, if the agent paradigm was based on BDI agents, then the developers analysed the system requirements also based on BDI agents [Kinn+96]. In other words, the agent role utilised the capabilities of the belief, desire and intention.

Using this technique, we are able to present a general method to develop an agent system rather than be limited to a specific agent paradigm. This is because it provides a stage between RA and ASLD, at which the designer can make judgement of the appropriate of agent paradigm. The different agent paradigms lead to difference in detailed design modelling attributes but do not change the stages of the development process. The analysis elicitation process is able to identify the appropriate behaviours assigned to agents.

However, in order to show a clean development process, the method presented in this thesis focuses on the issues related to agents utilising autonomy and reactive behaviour[3]. Moreover, we found the combination of the two agent properties of autonomy and reactivity presents a valuable basis for an agent-based system, as most researchers have found [Jaf97][Jat97][FIPAOS00][Bell+99][Nwan+99]. We believe that the other properties such as desire (stated in Chapter 2, section 2.1) are the additional properties of these two behaviours, since the concept of agent with autonomous is not enough to present the "agency" of the agent.

The AR stage uses an event-based scenario, role based use case and system goal contextualised to the attributes of the agent. The scenario and use case presented here are an extension from the traditional scenario and use case in accordance with the agent overview itself, and the system goal is an additional complementary technique in analysis of agent-based systems that are associated with a goal due to a high level of independency. The integration of these three models is used to develop the agent system architecture model as built in the ASLD and several other models may be appropriate at the IAD stage.

The figure also shows the parallel process of the analysis requirement for ASD and AED that leads to the decision whether to develop a bespoke agent environment or reuse an existing agent environment. Later in this chapter, we provide the criteria[4] and an guideline to reuse of an existing agent environment. Even if at an early stage it is decided to reuse an existing agent

[3] Modelling presented in the method is limited to modelling appropriate for autonomous reactive agent. [Giun+02] [Kinn+96] present appropriate modelling for BDI agent.

[4] Schoepke has provides an overview to choose agent environment but focus on business[Scho98].

environment if the result of analysis of the existing agent environment does not match the agent environment requirements, it will be necessary to develop a bespoke agent environment.

The modelling process and the deliverable modelling in each stage are discussed in more detail in the following sections.

## 4.1. AGENT SYSTEM DEVELOPMENT (ASD)

The prerequisite for designing an agent system is that it understands about agent concept such as goal, roles, resource, actions, events, agent capabilities, communications and their relationships, and the relationships between agent capabilities with the four agent criteria: behaviour, distribution, communication and openness (Chapter 2). The agent framework design (Figure 3.11) is used as a guideline to design a multi-agent system. It shows the relationships the elements goals, roles and tasks with an agent.

Note here that we also accept some of the techniques proposed in the existing agent methodology as well as discussed in Chapter 3, but not focus in this research on concept such as permission, responsibilities and activity. The modelling techniques presented in here aims to show a clear transaction process.

The agent system design artefact as portrayed in Figure 4.2 shows the activities in each stage, the deliverable of modelling of each activity and the flow between stages shows previous activities are used as input for modelling of the next activities. For example, the activity of scenario modelling referring to system goal and events is captured. The agent analysis modelling is used as an input to the *ASDL* stage, while the *IAD* stage modelling is developed from the previous stage of design modelling. These three stages consist of nine activities, which these activities are discussed more detail in following next sub-sections:

- *Requirements Analysis*: problem domain, identify system goal, identify scenario, identify use case, and identify appropriate agent paradigm.
- *Agent System Level Design*: identify agent and interactions, identify agent system plan.
- *Individual Agent Design*: identify agent data structure and identify agent plan.

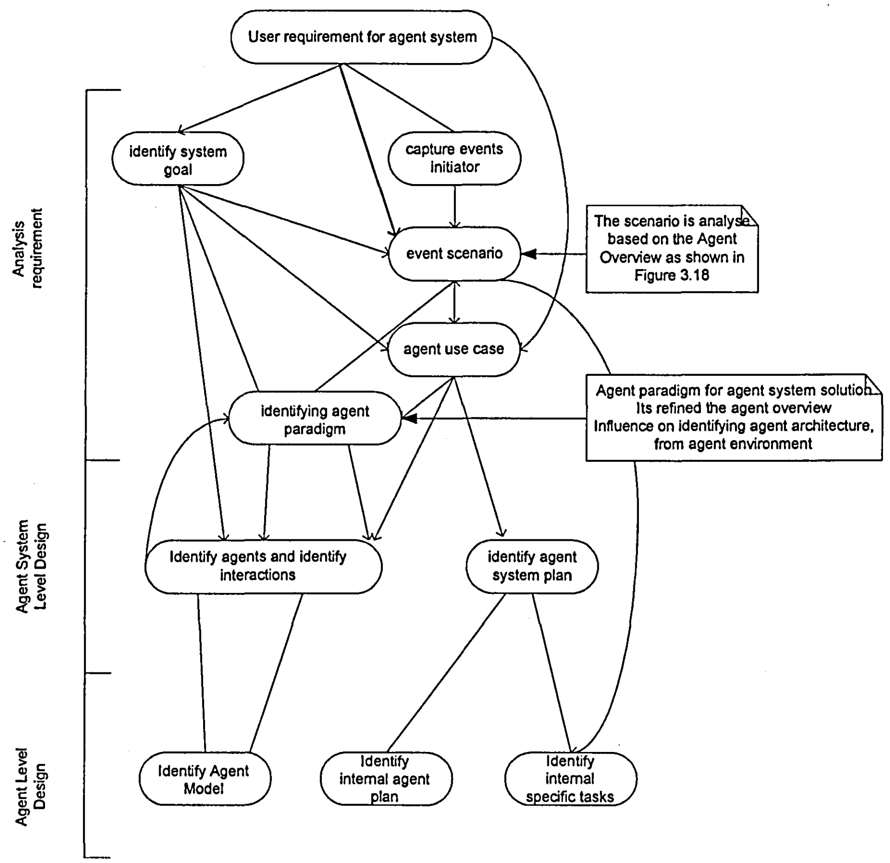

User requirement for agent system
identify system goal
capture events initiator
event scenario
The scenario is analyse based on the Agent Overview as shown in Figure 3.18
agent use case
Agent paradigm for agent system solution Its refined the agent overview Influence on identifying agent architecture, from agent environment
Identifying agent paradigm
Identify agents and identify interactions
identify agent system plan
Identify Agent Model
Identify internal agent plan
Identify internal specific tasks
Analysis requirement
Agent System Level Design
Agent Level Design

Figure 4.2: Agent System Design Artefacts

The arrows show how modelling influences other design modelling. For example, a system plan activity is developed from the scenario model and agent use case model, while the architecture agent system is developed from the use case. The figure shows the parallel modelling process of producing scenario and use case which leads on to identification of the right decision of the agent paradigm of the agent system solution.

The *ASD* assesses whether it produces an optimum design solution, as discussed later in the assessment section. The following subsections discuss the activities, modelling process and deliverables of each agent design stage.

### *4.1.1. Analysis Requirement (AR)*

The AR stage aims to analyse the user requirements in order to produce a concrete requirement specification pertinent to the agent solution through four activities: identification of system goal, role base scenario, agent use case and agent abstraction.

The agent overview portrayed in Figure 3.5 presents the agent. The designer should have these agent attributes in his/her mind along the analysis process.

We use a context diagram model to present the high-level analysis based on events. It aims to present the boundary of the system, that captures all possible events could occur in the system and captures the term associated with the entities used to describe agents from the user requirement specification. The events are any events that influence or generate the activities of the system, or cause changes of any knowledge or states in the system. The event may come from a source such as user, a machine, etc. A source may consist of several types of events. For example, in a security monitoring system, the events are captured by sensors, e.g. a smoke detector, an alarm is smashed, the button in a disabled person's toilet pressed, an elevator emergency button is pressed. Each event is associated with activities for the solution of each event. When the smoke detector rings, a decision on what action should be taken, who should take it and how to negotiate with the fire brigade system to identify which station should take action. Similarly, a set of actions is triggered if an alarm has been smashed. Events are associated with scenarios as discussed later.

The context diagram aims to present a very high level of system domain. The agent context diagram is different from a traditional context diagram[5]. The agent context diagram is represented as a square box with the arrows pointing into the box. The arrows represent all possible events that may occur in the system. The events in this context are events coming from outside the system that influence the flow of the system. The 'can' notation represents knowledge/resources representations, or another legacy system. The first arrow shows all possible events captured in the system that initiate the system. The context diagram of the security monitoring system is modelled as Figure 4.3.

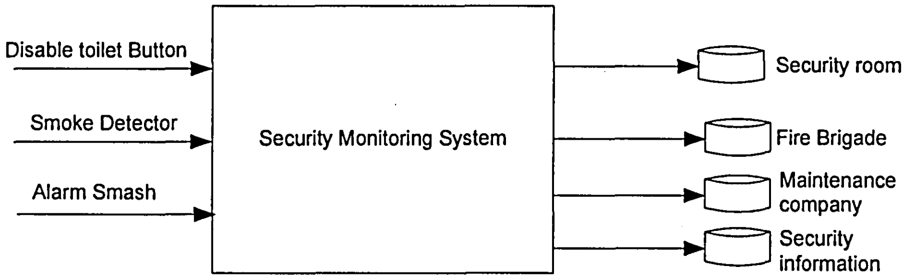


Figure 4.3: Agent Problem Domain

The system goal is developed according to the user requirements, but it focuses on how the system achieves its goal, this is used as guidance to produce the scenario. The scenario describes the sequences of activities that may influence the process of achieving the system

[5] The traditional context diagram focuses on the flow of information in and out of the boundary of the system [Ash+90].

goal. Each event (in figure) is associated with each scenario. Use case is developed based on each scenario and finally, agent paradigm is identified by analysing the potential use cases to represent as roles.

#### 4.1.1.1. Identified System Goals

Identifying the system goal is an analysis activity, which captures the user requirements that influence the achievement of the system. The identification of the system goal is to show the agency of the agent system. In Chapter 3, we discussed the differences between the objective of a system and the goal of a system such as goal-oriented [Chen+92], analysis patterns [Fowl96], task-oriented etc. Any of these methods are acceptable, because we believe that no specific method is the best. However, the research found a system goal represents a certain level of objective. Therefore, the system goal is developed using a similar process that used to identify the objectives of a system.

Extracting the essence of a set of system requirements identifies the objective of a system. The system goal is identified based on the objective, but it should present *a certain level of objective* that is supported with other requirements stating how to achieve the goal. The objectives are analysed by identifying the objectives that represent criteria for the system. Thus, identifying a system goal provides the criteria of the agent system. Agent system goals are describes in a hierarchical manner. Each goal may consist of several sub-goals at adjacent levels, but the adjacent levels usually describe the series of sub-goals needed in order to achieve the upper level goal.

Figure 4.4 shows an example for a system goal model for the Filtering application. The system goal is developed based on the Filtering application requirement. The objective of the Filtering application is to provide relevant articles found on the Internet, while the Filtering application goal is to provide the user with the most interesting articles found in the internet. Several activities are identified that influence the achievement of the system goal. The first filters the downloaded articles according to an accurate user profile before displaying to the user. An accurate user profile is achieved based on the reinforcement relevant feedback from the user. The more users provide feedback on the value of the interesting articles, the more precisely user profile describes the user interest and the more accurate is the result of the filtering. The second activity uses learning techniques mechanisms while updating the user profile based on feedback the user provides, as in genetic algorithm. This technique provides a mechanism to create the finest fittest chromosome while updating the user profile, so that the user profile presents the accurate information on the user's interest. The frequency of the feedback value also influences the accuracy of user profile.

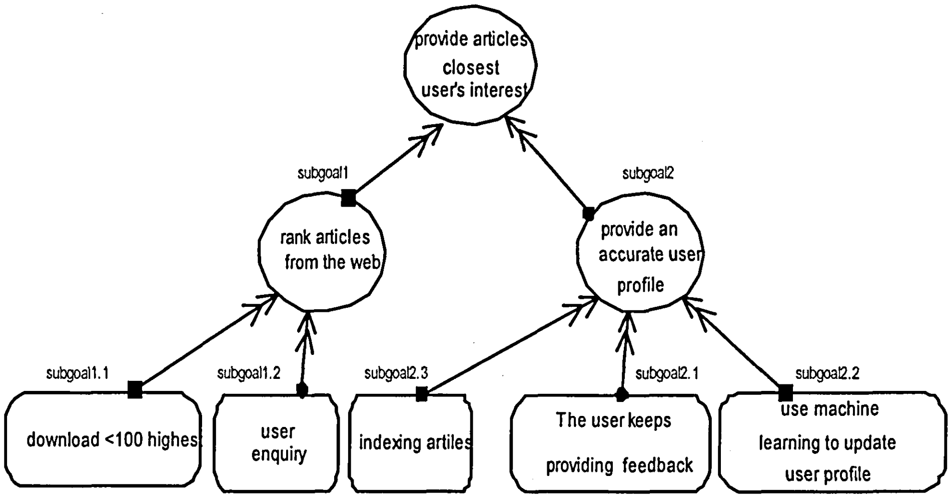

provide articles closest user's interest
subgoal1
subgoal2
rank articles from the web
provide an accurate user profile
subgoal1.1
subgoal1.2
subgoal2.3
subgoal2.1
subgoal2.2
download <100 highes
user enquiry
indexing artiles
The user keeps providing feedback
use machine learning to update user profile

Figure 4.4: The Filtering Application System Goal Model.

The figure shows that the goal of the system is achieved according to subgoal1 and subgoal2, while the sub-goal2 is achieved based on how frequently the user provides feedback and provides a function to use learning techniques while updating the user profile. The sub-goals at adjacent level are those that must be achieved in order to achieve the goal at the upper levels. The system goal is more readily created using a top-down approach. It is more difficult to develop using a bottom-up approach (see Appendix E:3 to the description of the notation).

The system goal analysis aims to be used to identify agents in the later stage of assigning roles to the agent. The system goal describes activities that influence the achievement of the system goal, rather than describing the associations of the activities or decomposition of activity. The system goal is related to the next stages of how activities are described in the event based scenario. For example, the sub-goal2.1 is related to events from the user and other sub-goals are related to the user enquiry event.

### 4.1.1.2. Identify Scenarios

Identify scenarios aims to transform the user requirements into scenarios of the events that occur in the system. The scenarios describe the consequent activities when an event occurs in a table. The scenarios describe the entities of the agent concept. For example, referring to Figure 4.3, events such as smoke detector or alarm smashed is associated with scenarios named as Scenario smoke detector and Scenario alarm smashed. Each scenario shows the activities performed as a consequence of an event occurrence. In addition, there are scenarios, which comprise combination events. For each activity, the name of the activity is described, as well as the relationship to the user requirement and the association with knowledge (databases/resources).

Referring back to the example of Filtering application, the context diagram consists of two types of events that come from one source, i.e. the user. The external environment is the articles in the Web. The two events are the user enquiry and the user feedback. A scenario technique is used to describe the flow of activities of each. In this case, two events scenarios are developed to describe the requirements of the Filtering application.

| **Scenario 1** | | |
|---|---|---|
| Event | **User enquiry** | |
| Source | **User** | |
| Activity | | |
| 1. | Login system(R1)<br>(userlogin(R2), password(R2),<br>keyword(R1)<br>offers common keyword categories | Role<br>subgoal 2.1 |
| 2. | Validate the user (R2)<br>(Userid, password)<br>UserloginDB | |
| 3. | automated creating user profile<br>UserprofileDB | Role<br>part of subgoal 3 |
| 4. | Search from web the (R2)<br>userid, keyword | Role<br>subgoal 1.1 |
| 5. | If match then download articles | |
| 6. | Inform user if not matched | |
| 7. | indexed articles(R4)<br>userid (article's title, address, content),<br>ranked value) | Role<br>subgoal 1.2 |
| 8. | request latest userprofile (R3)<br>userid<br>uerprofileDB | part of subgoal 2.2 |
| 9. | Capture indexed articles | |
| 10. | filter articles(R2)<br>indexed articles, user profile | Role<br>subgoal 1.3 |
| 11. | Display to the user<br>(articles, ranked value) | part of subgoal 2.1 |

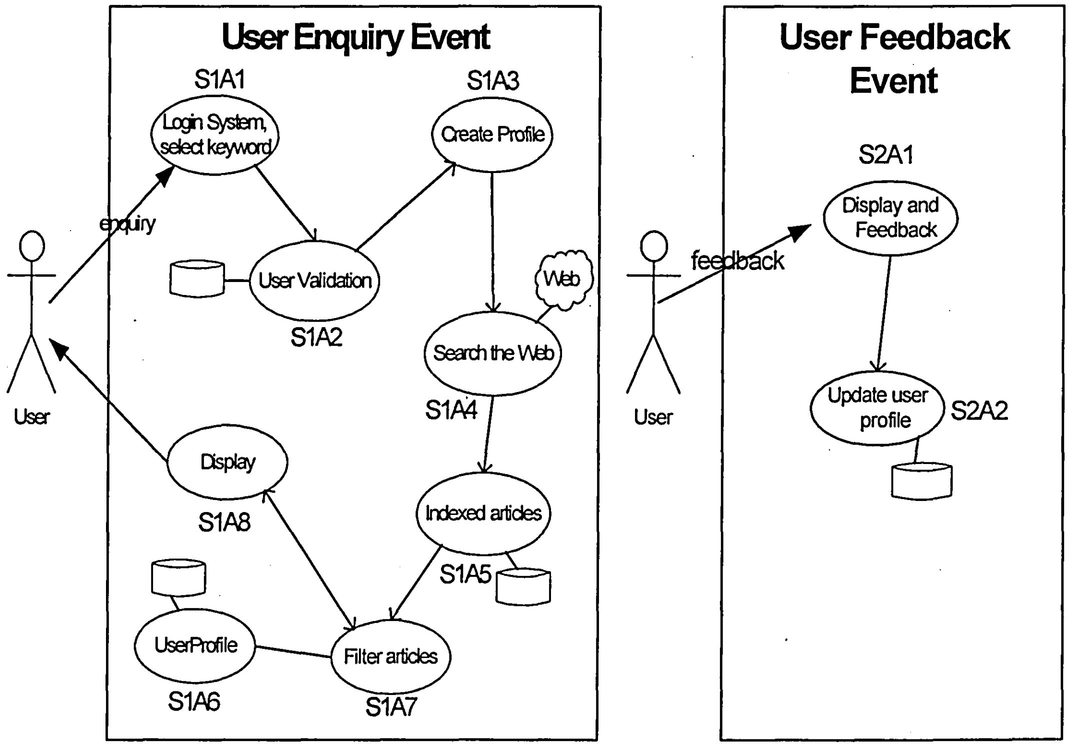
Table 4.1: Event Scenario.

Table 4.1 shows the user enquiry scenario. Each activity describes the activity name (i.e. login system, validate system), the number of requirement stated in the user as R# (i.e. R1, R2) and any databases related to the activity (i.e. UserprofileDB). A similar process is performed for the other events.

The role is identified from the activities presented in the scenario. Role can be identified from high level to low-level abstraction. In certain problem domains role is easily identified from the requirement. However, in more precise methods, the identification of role uses the concept of role presented in Chapter 3, where role is identified based on *goal, capabilities, resource* and *state.* So, the role is identified in relation to the system goal. Another clue to the identification of role is that it performs action. Various kinds of actions are performed: internal action such as change resource or state and interaction with other agent or change external object. The Table shows that the activities are assigned as a role as the result of elicitation analysis[6] as the requirement specification for the system.

### 4.1.1.3. Identify Agent Use Cases

The end product of *use case* describes the relationship between roles, events and resource. The event-based scenario only describes a set of activities without showing the relationship between the roles, but do not define the role. The notation of *use case* is similar to the object oriented use case, but the object oriented use case presents the use case associated with an actor [Cons97][Jacob+95]. We define an actor to be inside the use case. All scenarios associated are transferred into a *scenario use case.* Each scenario is associated with a use case.



a. Event Use case: User enquiry

b. Event Use case: User Feedback

Figure 4.5: The User Event Use Case

[6] refine the user requirement, it may change the appropriate requirement to produce concrete requirement by utilising agent.

The real *use case* is created by the combining all of all scenario *use cases* that are used to achieve the system goal. The *real use case* is aims to identify the unique role presented in the real use cases. The real use case is developed by grouping the similarity of roles exists in each scenario use case and remains the relationship of between the use cases. The following process shows the example on how to create the real use case, which define as agent use case.

The candidate role in Table 4.1 transfers into use case. The Scenario 1 (user enquiry event), is transferred into the event use case shown in Figure 4.5(a) and Scenario 11 is transferred into Figure 4.5(b). The oval notation is the use case as potential as role and the line shows the flow of action. The 'can' notation is a resource and the 'cloud' is the changes of environment. The figure shows that the *scenario use case* represents the association of the role with resource. Role No.1 in Scenario 1 becomes use case S1A1. A similar process is performed for all roles in the scenario.

The User Enquiry use case consists of the following use case elements: {S1A1, S1A2, S1A3, S1A4, S1A5, S1A6, S1A7, S1A8} while the Feedback use case consists of following elements {S2A1, S2A2}. The real use case represents unique entities of use case built from the combination of all the scenario use cases.

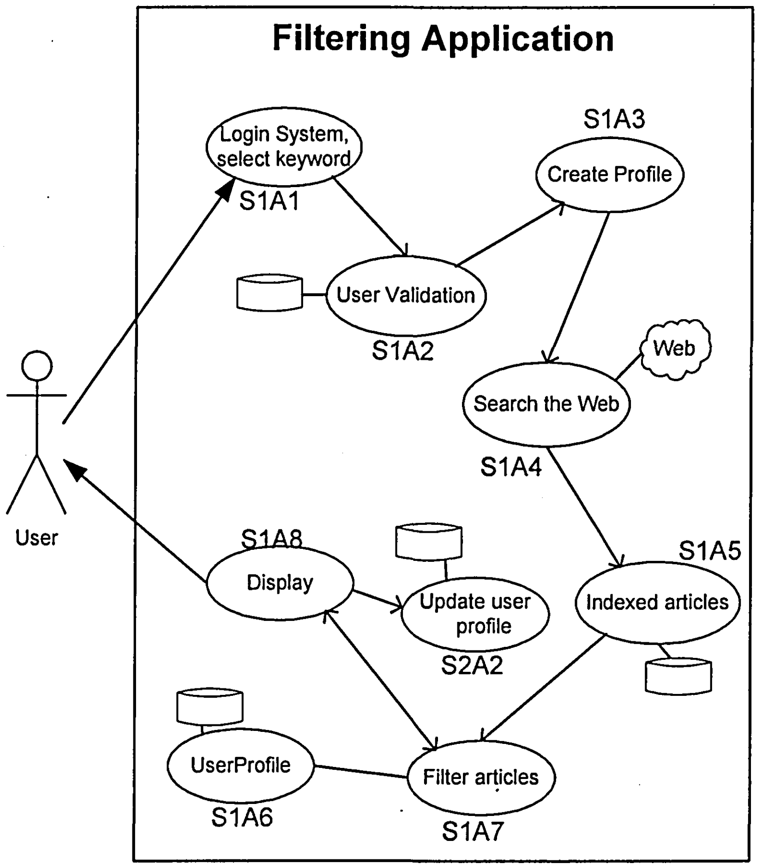


Figure 4.6: The Use Case for Filtering Application

As a result, the two *scenario use cases* shown in Figure 4.5 are transferred into Figure 4.6. The diagram shows the combination of functions between both event use cases. The use case of S2A1 is similar to S1A8. The rate of recurrence of a role use case is called the *pits* value. All of the roles have one pits recurrence value, while the S1A8 has two pits recurrence value. The pits value is used to capture the level of concurrent level of each role as it is performed. It useful as an indicator of rationality while assigning roles to agents. If the pits value of a role is high then the agent is complex and not assigned any other role. On the other hand, the type of role should be taken into consideration. For example, a role, which consume much time and processing should be assigned to an agent. Imbalance while assigning roles causes the agent system performance to be low because the agents stated are too busy and delay the flows of whole agent interactions. A single user may not affect the performance. However, the nature of the Filtering application consists of many users, each connecting to the system and affecting the performance of the system.

#### 4.1.1.4. Identify Agent Paradigm

There are various potential agent capabilities, which may be provided to agents as discussed in Section 2.1(agent properties). The identification of the agent properties is an important aspect of designing agent systems. This means, at an early stage of analysis, the potential of agent properties used for agent solution need to be identified. Not all agent paradigms are suitable for all kind of agent system solution. At this stage, the designers need to justify the appropriate attributes to present agent paradigm for agent solution.

Some problem domains may benefit from autonomy reactive or BDI agents. The BDI are associated with identification of the belief (the changes of the environment), desire (plan that associate with goal) and intention (actions), while autonomous and reactive what utilised the concurrence and action. Both views of agent (agent paradigm) may consist of two kinds of plan: individual plan and global system plan (share plan). For example, the BDI agent solution is suitable for use in the traffic monitoring system. This is because it consists of various agents; each agent observed certain traffic areas (keep changes) and provides with goals for each area of traffic. Another plan is to achieve the global traffic management for bigger areas.

The following discusses some methods used while justifying the agent paradigm for as an agent solution:

- Analysis of the system goal describes how the system achieves the goal of the system. The system goal is developed according to the user requirements. How the designer captured the system goal is important, because it influences the rest of the design. The system goal is developed based on complexity tasks, decision-making, roles in organisation, dependency tasks or knowledge. However, the system goal describe here is more towards on complexity tasks and distribution tasks.

- Analysis of the use case- Agent use case shows the roles associated with other roles and knowledge. The suitable agent properties are identified by analysing the dependency /independency of the roles while assigning the roles to agents.

In Filtering application, the analysis of the user requirements shows that the roles utilise the agent properties of *autonomy and reactivity* such as filtering, indexing and searching. These roles are complex to perform (long), which is an advantage if performed in parallel. The analysis of the scenario and user case shows no dependency knowledge among the roles that change of internal agent state of knowledge. Therefore, BDI agents or decision-making would be not the appropriate solution. There is no knowledge dependency rather than task dependency. In this case, the decomposition is based on task decomposition, similar to the sub-goal2.2 as a complex task assigned to an agent.

### *4.1.2. Agent System Level Design (ASLD)*

The *ASLD* aims to produce the architecture of the agent system solution. At this stage, the architecture represents the agents and their interactions. The *ASLD* is a recursive process identifying agents and interactions.

#### 4.1.2.1. Identifying Agent

Activities at this stage are to identify agents and their interactions. It is a recursive process until an optimum agent solution is achieved. The agent identification is influenced by the four agent characteristics: *agent behaviour, distribution, communication* and *openness.* The following describes how each agent characteristic influence on identifying agent.

*Identification of agent based on Agent behaviour*

The agent identification follows the following sequences of steps:

*The first* step uses the agent paradigm shows in Figure 3.6. In the case of the Filtering application system, the agent paradigm for Filtering application is based on autonomy and reactivity properties as the agent paradigm of agent solution (Figure 3.5).

*The second* step uses the agent use case to assign the roles to appropriate agents utilising the agent properties as identified in the first step. Identifying agents is a manual activity. The designer has to justify which roles are assigned to an appropriate agent. The system goal provides guidance on assigning the roles.

In order to provide guideline on how roles are assigned to an agent. This section shows some examples of common agents: interface/wrapper agent, intelligent agent, intermediate agent and agent specific tasks as a guideline. The interface or wrapper agent is an agent assigned as an interface to hardware, files or databases, while a wrapper agent is defined as an interface agent with a legacy system. An intelligent agent is an agent usually provided with intelligent behaviour. It usually applies a specific intelligent technique such as fuzzy logic. However, this method is limited to an intelligent the provision of intelligence to an agent rather that providing intelligence as a result of agent interactions. An intermediate agent is identified as a mediator between an agent and other agents or human or hardware or other entities, which support agent development. Agent specific task is an agent, which is assigned for a specific task such as functions. For example, decision making is a role and the tasks are functions used to allow the making decision role to be performed. This method shows that the use of roles for identifying agents ranges from high-level complex agents to a lower level agents.

*Identification of agent based on communication*

The used of interaction protocols helps in identifying agents. This can be captured from the user requirements. The collections of interaction protocols as shown in (FIPA+97b]) are the agent interaction patterns that can be used to identify agents, for example the identification biding role influence on identifying seller and buyer agent.

The agent community has defined the collections of these interaction protocols that are most common used in agent systems, for example contract-net, negotiation, bidding, etc. FIPA has defined more than a dozen types of interaction protocols. The interaction protocol presents the pattern of interactions between two agents, in order to achieve their goal. Examples of FIPA interaction protocol are FIPA-request, FIPA-send, FIPA-contract-net, FIPA-dutch-auction, FIPA-English-auction (see [FIPA00]) [7].There are several interaction-protocols patterns being developed [FIPA+97b].

*Identification of agent based on distribution*

The distribution criterion is a factor in identifying agents as well. The locational and conceptual distribution influence agent identification. The roles distribution is an example of conceptual distribution. In some cases, distribution becomes the first factor of identifying agents. For lower case example, in supply chain systems, the distribution of the supplier and consumer influences the identification of agents. Referring to the Filtering application, the user requirements and the user validation roles are assigned on the user side. The user validation is assigned to the user agent. The indexing articles and filtering articles roles represent a different role domain. Indexing articles is assigned to DocumentAgent, while filtering articles is assigned to FilterAgent. Both agents can be located in the same location or distributed depending on the user requirement.

*Identification of agent based on openness*

The openness criterion also influences identification of agents. The openness criterion enables reuse of an agent service. An agent provides a service. The service can be reused by another agent system. The openness criterion has influences on identifying agents. The service of indexing articles can be reused by other agent systems. The openness criterion only can be applied, by assuming that the agent environment provides an open agent environment architecture.

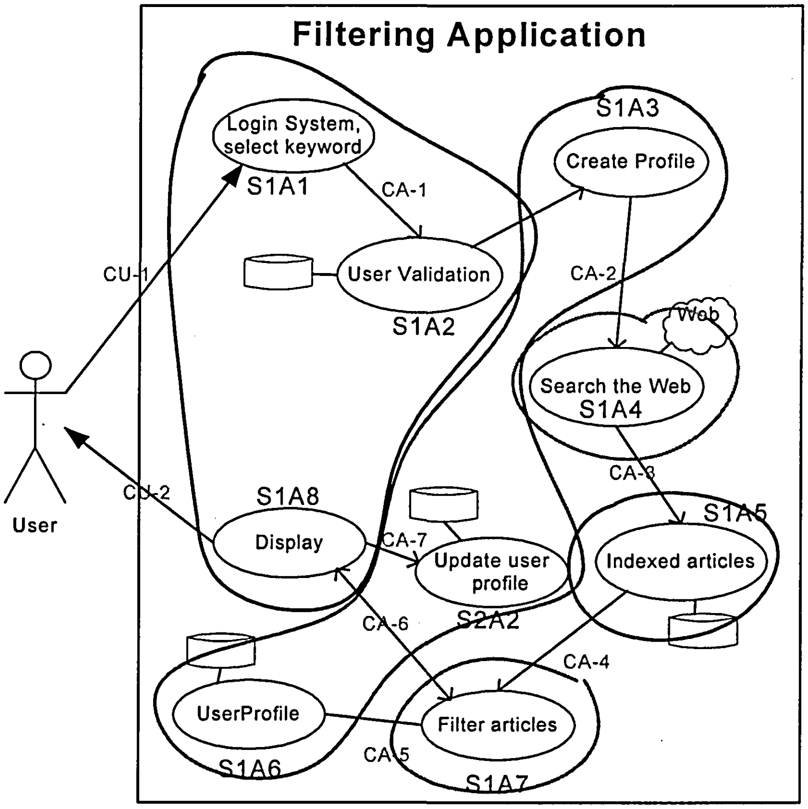


Figure 4.7: Filtering Application Agent Identification

Figure 4.7 show the analysis of roles is able to group the use cases of similar roles. The figure shows five agents are identified: *UserAgent, UserModelAgent, Search Agent, DocumentAgent*

[7] Not limited to these interaction pattern but still open into research.

and *FilterAgent.* The discussion of assessment, later, provides a guideline to produce an optimum agent system design.

As the result of agent identification, each agent is modelled using the abstraction of agent model shown in Figure 3.5. Each agent is assigned with a name, goal, roles and tasks. For example, the first area from the left in Figure 4.7, called UserAgent, is transferred into the agent model as shown in Figure 4.8. The goal is to provide a user interface between the user and the system. UserAgent has two roles: enquiry and feedback. The enquiry role entails the task of sending the enquiry to other agents. The Feedback role is to present the user interface of ranked articles and allow the user to provide feedback. It presents two tasks: display result and feedback on articles by assignment to another agent to update the user profile.

| |
|---|
| *Agent name*: UserAgent |
| *Goal:* Good userinterface |
| *Roles* : Query<br>:Feedback |
| *Query:* Send to query to other agent<br>*Feedback*: update user profile and new articles |

Figure 4.8: UserAgent Abstraction

#### 4.1.2.2. Identify Agent Interaction

The identification any pattern of agent interaction from the requirement is the source of identifying agent interaction. However, we propose a method to identify interaction if no existing interaction protocols are applied. The colour boundaries illustrate the use cases/roles assigned to agents. The agent interaction is identified by capturing the arrows across the colour boundary. Each agent interaction is named. The figure shows the agent interactions as marked with CA-# showing the interaction between agents and CU-# showing interaction with outside sources of events. The arrow across from S1A1 to S1A3 is identified as CA-1, while the S1A3 to S1A4 is identified as CA-2. A similar process is performed until all agent interactions are identified.

Referring to the Filtering application, the interaction between FilterAgent and UserModelAgent uses request protocol. The request protocol consists of the two communications, send and inform. This is because the FilterAgent wants the result from the interaction, rather than one-way interaction. FIPA agent communication specification provides several interaction protocols. The interaction protocol is selected depending on how each agent achieves its goal. For example in order to achieve the FilterAgent goal, the agent needs to request the UserModel

Agent about user profiles. Figure 3.9 shows the agent interaction between the UserAgent and the SearchAgent, and the FilterAgent and the UserModelAgent.

#### 4.1.2.3. Agent System Design Modelling

The result of identifying agent and agent interaction leads to identification of two types of modelling to present the high system design stage consisting of *Agent system architecture* and *Agent system plan.* Each activity modelling, presents a different perspective of modelling as described below.

*Agent system architecture*

The agent identification and agent interactions as shown in Figure 4.7 are transferred into agent system architecture as shown in Figure 4.9. The figure shows the agents and interactions. It also disseminates the conceptual and locational distribution using two types of line that represent two of distinct interactions: same or different location. Colour lines illustrate the interaction between agents in different locations, whereas interaction between agents in the same location is shown as a solid line.

The architecture is also able to portray which agents are associated with the external world as a file or the Web, with agents providing any intelligence such as reasoning, learning, etc., which agent's only focus on internal processing, and which agents deal with the legacy system. The architecture describes the means of modelling notation. The Figure shows that the User is responsible as an interface agent to the system. The arrow with a big head shows the source of event (see Appendix E1-5).

The advantage of this model is the ability to present an overall view of the system related to the real world. With such diagrammatic notation, it is easy for the user to understand the solution and accurately to capture the user requirements. The architecture is able to present the real solution of the agent system, and describe the domain of the system. Such a presentation of the system makes it is easy to validate the proposed system fulfilled the requirements. However, this model only shows the static model of agent interactions. A dynamic modelling view to show agent interactions is needed. The dynamic view aims to illustrate the agent system plan, which is discussed in next section.

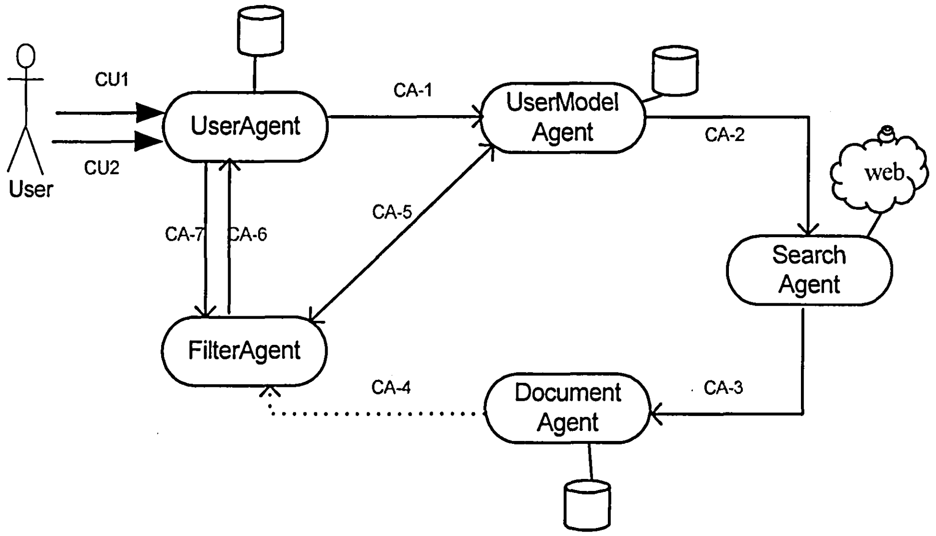


Figure 4.9: The Agent System Architecture

### 4.1.2.4. Agent System Plan

The agent system plan aims to present a dynamic modelling of agent interactions. It describes the events and agents that initiate agent interactions. The system plan model illustrates the sequence of agent interactions performed, based on the time, and the events that occur. It describes the scheduling of the agent system. For example, it identifies which agents act as initiators of the system. The system plan is not only used to schedule possible agent interactions, but can also be used:

- to show the reliability of the system,
- to present the intelligence of the system,
- to increase performance,
- to present a pattern of agent system interactions.

The system plan is used to schedule when and which agents interact with each other, which agents must be alive in order to perform the system, and which agent is the initiator and in what sequence the agents are activated.

The agent system plan modelling is portrayed using two types of models: *Plan sequence model* and *Plan activity model.* The plan sequence model presents the sequence of agent interactions based on time, while the plan activity model is able to the present the activity in each agent that leads to the agent interactions. Both of these models are useful to evaluate the system is fulfilled the requirement.

Plan Sequence Model

Figure 4.10 presents the plan sequence model for the Filtering application. The notation of the plan sequence model is an extension of the UML sequence diagram. The Plan sequence model uses a similar concept to the traditional object oriented sequence diagram, but the plan sequence model presents is not objects interactions rather than agent interactions.

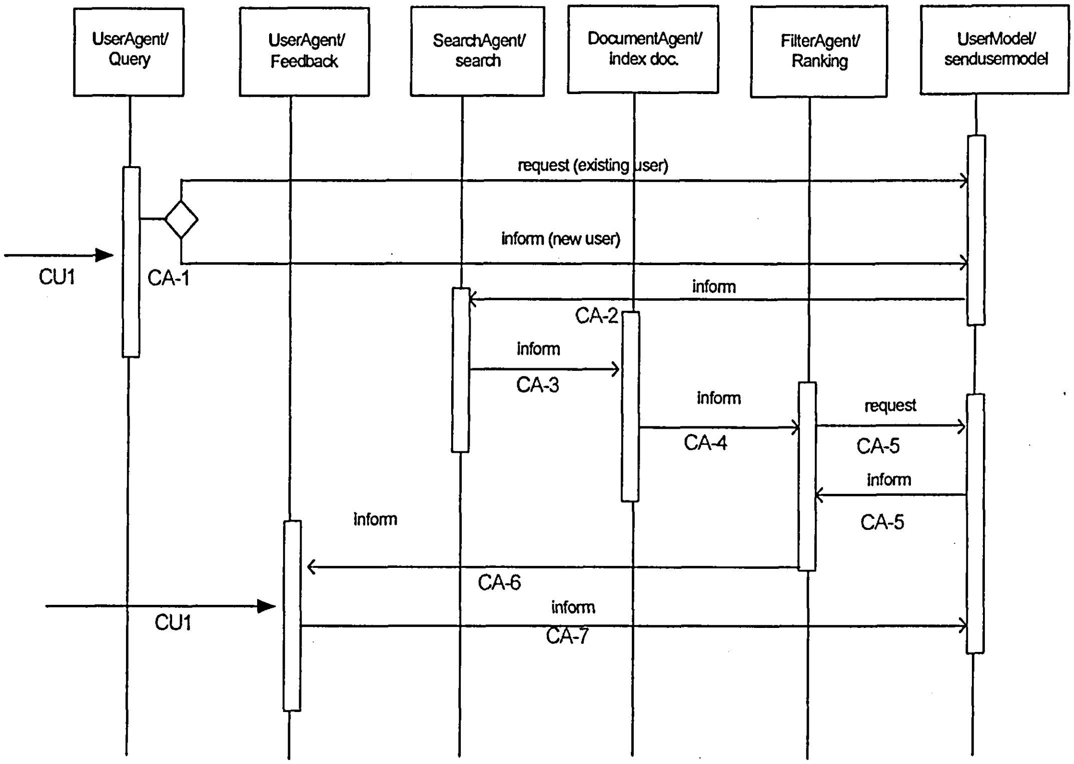


Figure 4.10: The Plan Sequence Model

The plan sequence model is developed based on the agent system architecture shown in Figure 4.7 and the agent abstraction shown in Figure 4.8. The agent system architecture describes the sequence of agent interactions and the agent abstraction describes the role of each agent. The UserAgent has two roles: query and feedback. The plan sequence model shows the agent interactions according to the agent role.

The figure shows that the UserAgent must be executed first, followed by the UserModelAgent. However, this system plan is used at the initial stages of the execution of the system. The Filtering Agent System shows the system plan as below: UserAgent-> UserModelAgent-> SearchAgent -> DocumentAgent-> FilterAgent.

*Plan Activity Model.*

The plan sequence model aims to overcome the limitation of the plan sequence model, which only focuses on the sequence of agent interaction but does not show the activity or decision that leads to actions. The modelling process of the plan activity model is similar to the traditional object-oriented activity diagram, but it presents the activity among the agents.

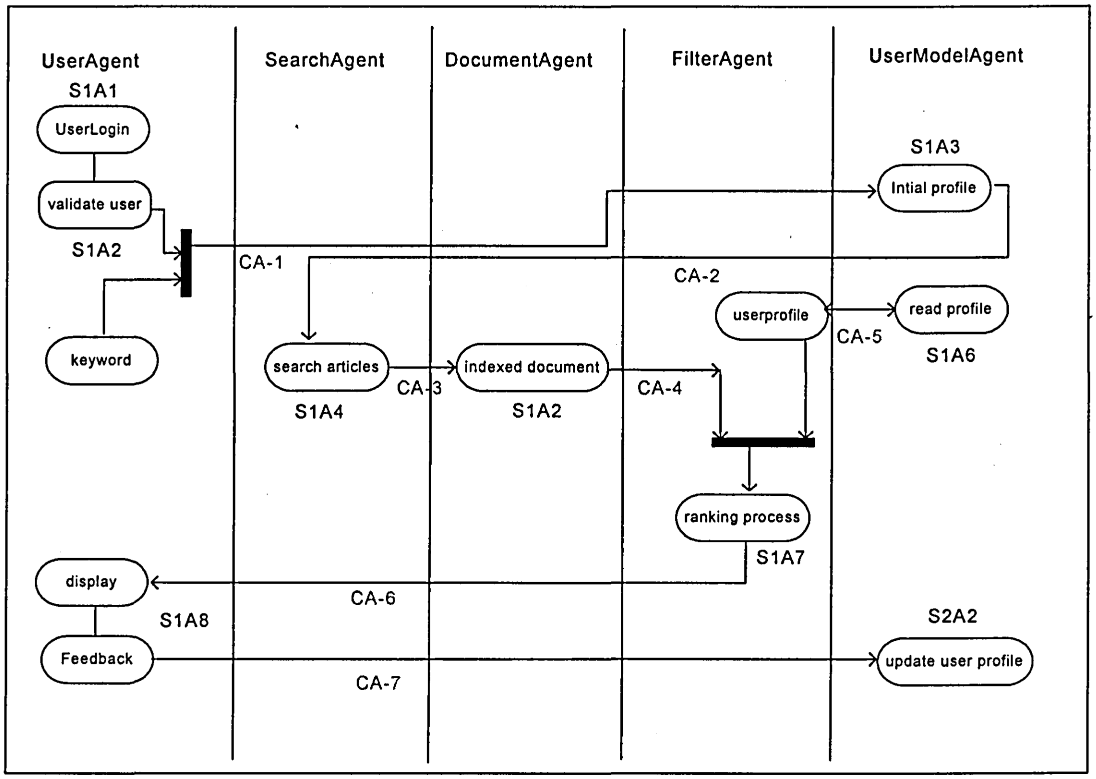


Figure 4.11: The Plan Activity Model

Figure 4.11 shows the plan activity model for the Filtering. The Filtering application only presents one event source; therefore, it only has one plan activity model. Each plan activity model presents each source event. The figure shows the transfer of the communication name and use case name into the plan activity model. The oval notations presented in the figure are the use cases shows in Figure 4.6.

As stated before, a system plan model is able to show the reliability, intelligence or performance of the system. The system plan above focuses on the initial stage of execution of the system, but does not focus on the reliability of the system. Reliability here means that the system will continue to operate even if some of the agents in the system are dead for some reason. For example, in the Filtering application, if the UserModelAgent is not working, the system still provides the result, but the filtering process will not use the user profile, but will use a standard user profile. The plan activity model is able to handle this situation.

The plan activity model is also able to indicate the intelligence of the system as the flow of the agent's activity. For example, in the Filtering application, the SearchAgent will not do searching when the network traffic is high or performance of the resource is utilised for other jobs.

Up to this stage, the design modelling is focused on agent and agent interactions, whilst in the next stage it is focused on individual agent design modelling.

### *4.1.3. Individual Agent Design(IAD)*

The IAD focuses on each internal agent design. It focuses on the internal design based on the information. The agent abstraction of agent identification, as illustrated in Figure 4.8, presents the agent goal and roles. The aim of this stage is to show in detail how each agent achieves, its own goals as the result of interaction with other agents. IAD consists of two activities: *agent model* and *agent plan design*

Agent Model Design (AMD)

The AMD aims to show detail information and task of each agent. The association of agent role with other task (internal task and internal interaction) are identified in plan sequence and plan activity model. Identification of this information is modelled as agent model as shown in Figure 4.12.

This agent model describes in more detail each role associated with actions. It portrays the communication name, with whom it communicates and the action that performs the interaction with other agents. The receiving event is marked as (i) and the action of role is marked as (O). The UserModel agent indicates the tasks for agent interaction and does not indicate any internal design task. The figure shows that the action as marked with (O) is an internal action or interaction action. The message type stated in the message type column indicates that the action is an interaction action. The data structure for agent interaction represents the content message of ACL. The information of the message is described as text or GIF. The structure of ACL is presented in Chapter 2.

| **UserAgent** | | | | |
|---|---|---|---|---|
| **Goal** | Provides interface for user interact to the system | | | |
| **Role** | Enquirer | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **Content** |
| 1. User interface | User | CU-1(i) | | |
| 2.Login system and perform query | UserModelAgent | CA-1(O) | *request* | *userid, password, keyword, status search* |
| 3. Display result | UserModelAgent | CA2.3(i) | *inform* | *ranked document result (userid, articles, ranked value)* |
| **Role** | Feedback | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| 1. Display result | Filter Agent | CA-7.2(i) | *inform* | *ranked document result (userid, articles, ranked value)* |
| 2.User Feedback | the User | CU-2(i) | | |
| 3. Feedback | User-Model Agent | CA-1(O) | *inform* | *Update user profile by using score value. (userid, articles, feedback value)* |

Figure 4.12: UserAgent Model

In certain situations, agents use different meanings of the semantic language. This is solved using an ontology.

```
Inform
        :Sender  UserModelAgent
        :receiver UserAgent
        :reply with      cmszouser
        :language        FIPA language
        :ontology        FIPA ontology
        :content  action(cmszo, userstatus)
```

Figure 4.13: Communication Data Structure

The Figure 4.13 below shows an example of definition of the data structure for one interaction task of the UserModelAgent. It shows an interaction with UserAgent using the speech act language command. The syntax of action (cmsuser,userstatus) describes the action of the UserModelAgent performed, whether to create a new userprofile or read the existing user profile.

*Agent Plan Design*

The agent model only shows the agent's internal tasks and the data structure use for agent interactions. In previous stages, it was possible to identify each agent's goal. However, the agent plan design aims to show the process by which each agent achieves its goal. We use the UML state diagram notation to model the *agent plan model.*

Figure 4.14 show the dynamic activity of each agent. The workflow of agent design artefacts in Figure 4.2 shows that the agent plan model is build from the agent system architecture, agent system plan sequence and agent system plan activity diagram. Figure 4.14 shows the UserAgentModel plan model. It shows all possible ways an agent receives messages or events, and how each message/event is associated with the internal agent tasks, leading to the agent's actions.

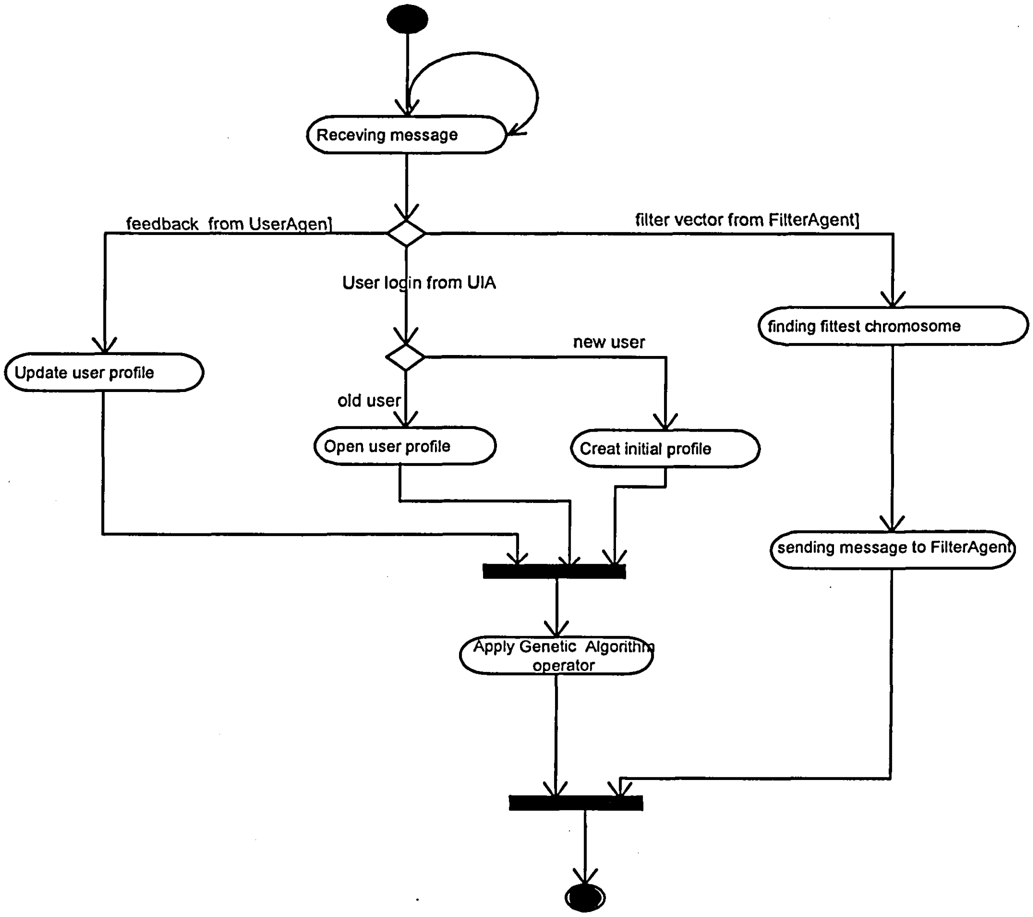


Figure 4.14: UserModelAgent Plan Model

The figure shows how the internal agent is always waiting for the event, to present autonomous behaviour. It shows that three types of events can occur. Two events are from the UserAgent and the other is from the FilterAgent. The first event is received through the user enquiry and the second event is through the user feedback. The figure shows the relationship between the events received that are associated with the internal agent. The UserModelAgent's internal tasks are to create a new user profile, read the user profile and update the user profile by applying the genetic algorithm, as described in the *Agent use cases.* The oval notation represented the internal agent task. The agent plan model shows the actions taken when the events occur that are associated with these internal tasks. The Figure 4.14 also shows the actions that influence interaction with other agents such as FilterAgent or UserAgent.

*Internal Agent Design*

The activity at this stage is to design all the internal tasks defined in the agent plan model. At this stage, the focus is on detailed design of how each agent achieves its own goal. It includes how to provide intelligence for each agent. Referring to the UserModelAgent, it has three internal tasks: update user profile, read user profile (find the fittest chromosome), create user profile. All these tasks applied the genetic algorithm. The attributes used to identify the user models is shown in Figure 4.15. A new user is provided with a constant value of five articles in default of the associated data. The task of updating user profile uses the same attributes as initiating the user model and the genetic algorithm attribute are defined to perform the genetic algorithm task while updating the user profile.

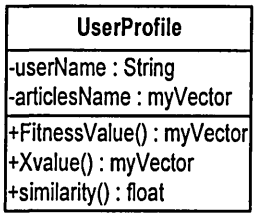


Figure 4.15: The Attributes to Define the User profile

At this stage, the data structure design is at object level. Figure 4.16 shows an example of class diagram of the FilterAgent using backprop and kohonen map learning techniques. The *DataSet* class defines the same data structure as of the backprop and kohonen map.

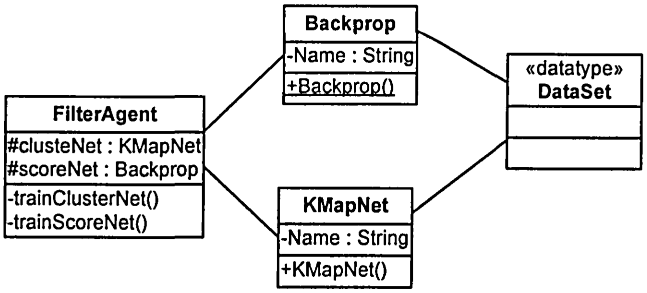


Figure 4.16: The Association between FilterAgent and Learning Techniques

Figure 4.16 shows how the UserModelAgent integrates with the learning behaviour. In these cases, the figure shows the FilterAgent used the components of Back propagation and Kohonen

Map Neural Network (class diagram). The internal data structure can be identified easily through the internal task of each agent.

### 4.1.4. *Refinement*

The refinement process is performed after identifying agents. The first version of the *agent system architecture* is assessed to produce an optimal agent system design. The assessment result influences the refinement of the agent system design. The recursive design process is applied until an optimum design agent system is produced. The refinement process is similar to *agent system design process*, i.e. involves identifying new agents or removing agents. The refinement of design agent may identify more agents and more interactions, or may remove some of the agents.

At this stage, we introduce the use of the conceptual distribution in the refinement process. The literature reviewed discussed in Chapter 3 revealed four types of conceptual distribution: organisation, location, decision making and logical. The conceptual distribution presents several perspectives of identifying agents. These conceptual distributions are used to optimise the agent design-balance between the complexity of the system and the performance of the system. These perspectives are uses in parallel. For example, according to the user requirement, the designer can use an organisation perspective to design agents, while at the same time it also should consider the aspects of location or logical as well. The use of an organisation perspective may produce only a few agents and the complexity of each agent may still be high. At this stage, the logical perspective can be utilised.

The first version of agent system design is based on the agent requirement (agent characteristic). A refinement process may be needed after the agent environment is developed. At this stage, a high-level agent abstraction is detailed into agent architecture, and the physical environment of the execution of agents has been developed. The development of the agent environment is able to identify the criteria of distribution, location or openness. It also identifies the mechanism of agent socialisation. Such features influence the refinement of *agent system design* in order to fulfil the assessment criteria describes later.

The functionality of locational distribution also influences the optimal design of an agent system. The designer should reduce the number of transactions among agents that are located at a long distance compared to those at short distance.

The openness criterion of the agent environment also influences agent refinement. An open agent environment presents reusable services. Therefore, during the design process, the designer should consider which aspects of services provided to agents could be reused. For example, the SearchAgent services provide a search engine or meta-search engine on the web that can be reused by another agent system.

This stage is able to identify exactly how many agents are involved, including the location, name and address, etc. of each agent. Meanwhile, the individual agent design is changed due to change in agent interactions.

*Assessment Attributes*

The optimal design agent is assessed based on three attributes: *number of agents, number of transactions* and *complexity of agents*. These three attributes should be balanced to produce an optimal design. The number of agents describes the total number of agents in an agent system, i.e. a multi-agent system. Parunak advises against having had too many agents. This is because *n* agents have $2^n$ possible number of interactions [Paru+97]. Too many transactions in a system will reduce the performance of the system. This is because multi-agent system design and implementation tends to be multi-threaded (both within agent and certainly within the society of agents). Wooldridge and Jenning remind us of the past lesson of concurrent and distributed systems which shows that an excessively multi-threaded system slows the performance of the system [Wool+98a].

The transactions of agents are identified based on how many times agents perform interaction. In order to optimise the agent system design, the value of rate of execution time is calculated based on the x pits of a role. The x number of role value should be not too different among agents. There are several issues can influence the performance of system, including the decision on identifying agent environment, as discussed in the next phase.

In order to utilise multi-threaded systems, the designer of the agent system should utilise the concurrency aspect of design as one of the obvious features of a multi-agent system. The concurrency of problem solving is able to increase the time of solution and utilise the resources. A poor multi-agent design is one where the amount of concurrent problem solving is relatively small or even in extreme cases non-existent. However concurrency should not be fully exploited. The concurrency aspect leads to optimisation in identifying the agent interactions, while too many agents increase the complexity of transactions among agents in an agent system. On the other hand, too few agents may result in high complexity of agents. Breaking down into several agents should reduce the complexity of the agent[8]. Identification of

[8] The organisation or hierarchy approach is a technique to reduce the complexity and reduce interactions by decomposed or grouped the dissimilarity/similarity of roles/tasks.

both types of complexity leads to the decision that the multi-agent system paradigm is not the optimal solution. Based on the above discussion, the design of agent system should balance between those three attributes.

## 4.2. AGENT ENVIRONMENT DEVELOPMENT(AED)

This phase is only needed if the organisation or designer decides to develop a bespoke agent environment. Designing an agent environment presents different method compared with designing agent system. Design agent system focuses on how agents are used to provide the solution of a system, whereas design agent environment focuses on what and how the functionality is provided so that it provides an environment for the execution of the system. The aim is to find out the components and their interactions that represent the functionality of the agent system.

Designing an agent environment is more akin to an art, rather than a precise science. An art of producing a good architecture depends to a large degree on experience. In involves a lot decision making, such as deciding the appropriate techniques to use to perform the agent environment's functionality. It also involves a decision of choosing appropriate technology to present the functionality. The decision focuses on non-functional requirements such as the criteria of flexibility, reliability and performance. Section 3.3.4.2 discusses the common functionality should provides to agent environment (result of derivation process).

Figure 4.12 shows that AED phase consists of three stages: *agent environment analysis, agent environment architectural design* and *components design.* Each stage is discussed in more detail in the next sub- section. This section is focuses on the development process, which illustrates the relationship of those stages.

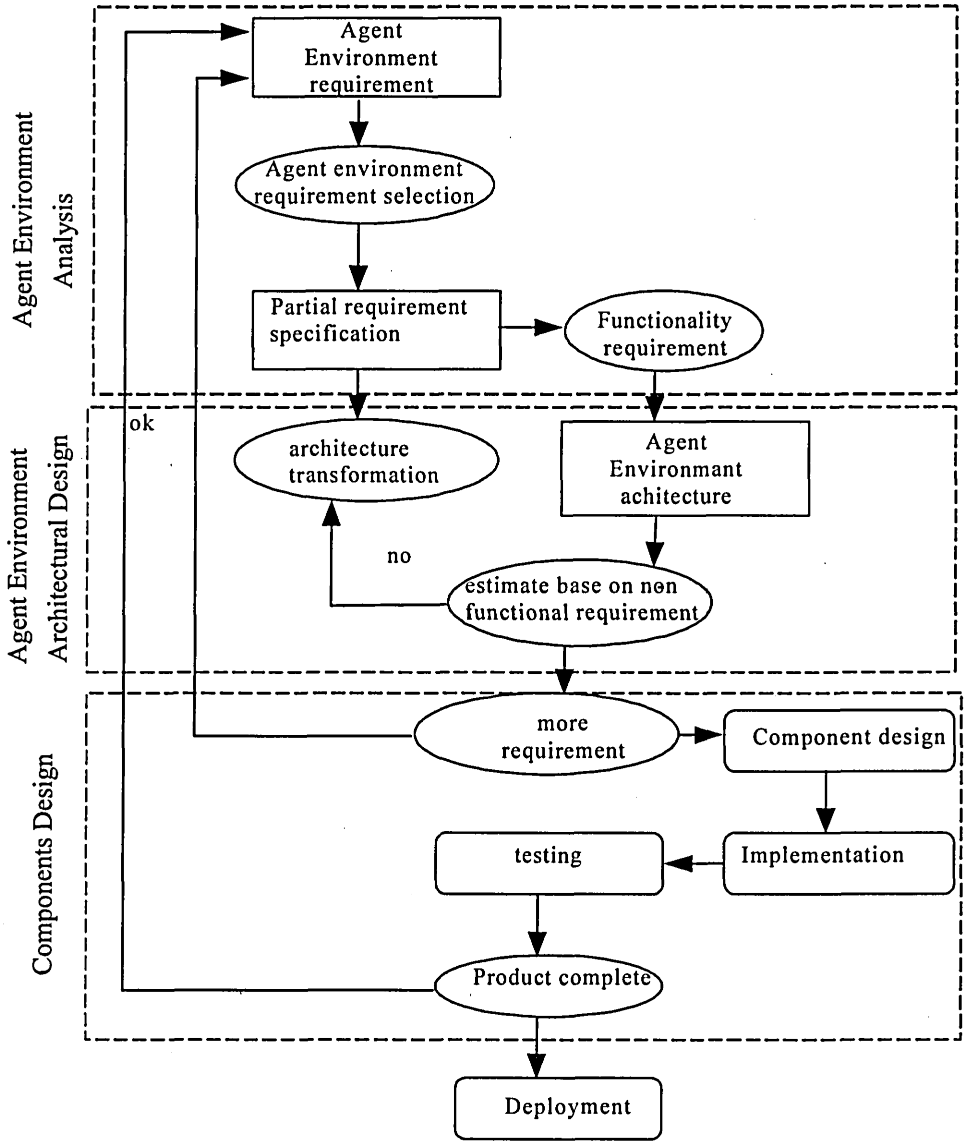

Agent Environment Analysis
Agent Environment Architectural Design
Components Design
Agent Environment requirement
Agent environment requirement selection
Partial requirement specification
Functionality requirement
ok
architecture transformation
Agent Environmant achitecture
no
estimate base on non functional requirement
more requirement
Component design
testing
Implementation
Product complete
Deployment

Figure 4.17:Agent Environment Development Process

The figure also presents a flow of activity and information of the whole development process (oval notation is the activity action, the rectangle notation describes the information are needed and the curved rectangle describes further detail activities), which is divided into three stages (dashed line). The development process presents a recursive process from selection and prioritisation of agent environment requirements until a complete product is produced. It illustrates a very high level of abstraction of AED phase.

The first loop shows the transformation of an agent environment architecture according to the first agent environment requirement. Based on this requirement, it is possible to identify the components and their relationships to develop agent environment architecture. For example, the user requirement stated the needs for a heterogeneous environment with the CORBA platform to be used as part of the agent environment architectural. This is an example of how a requirement leads to identification of common components and their relationships to produce the agent environment architecture. The second loop represents the choice of another agent environment requirement, which results in further components and relationships. At this stage, these components can be integrated with the previous agent environment architecture.

A similar process is performed until all the agent environment requirements have been taken into consideration. Every time the agent environment requirement is developed, the agent environment architecture is assessed and analysed based on the non-functional requirements. In order to produce a quality architecture design, architecture patterns or design patterns are used in the process, transforming the identified components and relationships into an agent environment architecture.

Each component is then designed in detail using the traditional method of development of object oriented components or reuse of existing components. The component design is able to show the relationship between objects/component or sub-components, which use design patterns to produce a quality component design, or it may reuse the existing components or sub-components. The complete design components are then implemented and tested until the entire components (includes integration of components) are implemented.

### *4.2.1. Agent Environment Analysis (AEA)*

Identifying an Agent Environment Requirement is the main activity of AEA. Figure 4.18 shows the process of identification of agent environment requirement, which aims to provide an input of the five views of agent environment design model as discussed in later stage (left rectangle notation, which discussed in section 3.3.4.2). The right rectangle notation shows the three common elements needed for development agent environment (scrutiny result), which leads the decision of agent environment requirement. The figure shows the artefacts for identifying the requirement and the flows of the artefacts to the next stage of development.

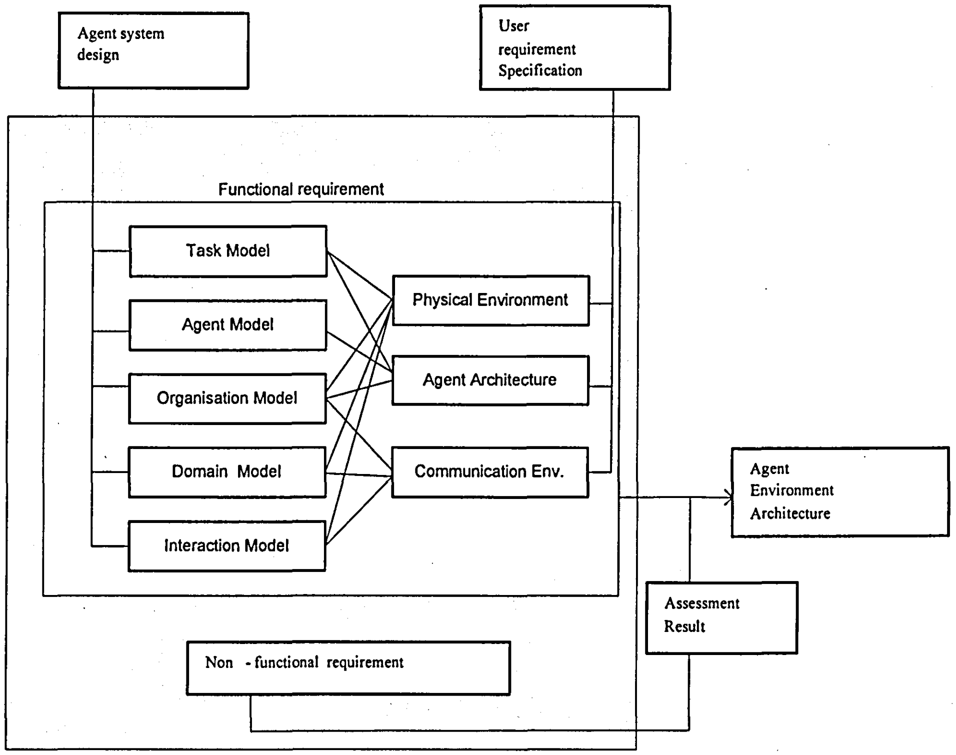


Figure 4.18: The Artefact of Identifying the Agent Environment Requirement

The agent environment's functional requirements are identified from two sources: from the result of the ASD phases and from the user requirement specification (stated by the user requirement), which generates two generic agent environment requirements:

- *Enterprise requirements*
- *Technology requirements*

The enterprise requirement focuses on the role of the agent environment and the purposes and policies that govern the system. It focuses on the semantics and structure of system and expresses the functional decomposition of a system into well-defined areas of organisation. The five types of models identified in previous chapters are the models captured from the ASD phase, while the technology requirements are captured from the user requirements or appropriate of current technology. The following shows examples of each enterprise models.

- *Organisation model* – the agent system architecture (i.e. Figure Appendix E-6) that describes the agents and interactions is an organisation model. It presents a pattern of agent socialisation that influences the decision on how to manage agents, called agent coordination: centralised or decentralised, and a group or multi-levels of agent groups. It also influences the identification of an appropriate agent coordination method. For example, the JAFMAS agent environment uses a petri-net method to present the agent coordination. This is the basis for the supply-chain management system. JADE is used in decentralised and multi-groups of agents.

- *Agent model* – analyses all the agent models to identify the suitable agent architecture [Mull99] (Refer to agent architecture discusses next).
- *Interaction Model-* captures all types of interaction between two agents (partly refer FIPA agent communication specification [FIPA+97b]).
- *Task Model* – identifies agent specific tasks used to perform the agent's role (associated with agent model). For example, to present the autonomy, belief, sensory, access knowledge, access environment, etc.
- *Domain Model- a*nalyse the information used for agent interactions. This identifies the heterogeneous information between agents. It is captured from the agent data structure model of the message content.

The three aspects of *technology requirement* that influence the design of the agent environment are *agent architecture, communication environment* and *physical environment*. Each of aspect is discusses as below.

*Agent Architecture*

Agent architecture represents the internal agent decision-making strategy that is suitable for the agent solution. It involves choosing an appropriate architecture type of agent paradigm solution, which been discussed in section 4.1.1.4. It describes how the agent achieves its own goals, including the infrastructure of each internal agent performed its roles.

Table 4.2 shows various example of agent architecture type that can be differentiate based on agent properties and their entities. The literatures review in Chapter 2 (Figure 2.2) shows the basic agent architecture entities are agent, environment, sensor and adaptor with belief, desire and intention.

| Agent Properties | Agent Environment Entities | |
|---|---|---|
| Pure Reactive<br>[Gene+87] | agent<br>environment | sensor<br>adaptor |
| Pure Stated Reactive<br>[Gene+87] | agent<br>environment<br>sensor | adaptor<br>state |
| Pure Percepts Reactive<br>[Gene+87] | agent<br>environment<br>sensor | see<br>adaptor<br>action |
| Pure State Reactive and autonomous<br>[Rusel+95] | agent<br>environment<br>sensor | adaptor<br>stated<br>action |
| BDI [Kend+95] | environment<br>sensor<br>adaptor | belief<br>desire<br>Intention |

Table 4.2: Agent Architecture Entities to Present Agent Behaviour

The following are additional agent characteristics that influence the identification of the internal agent architecture. These are part of agent capability, which may be provided to agents.

- Knowable -To what extent is the environment known to the agent?
- Predictable -To what extent can it be predicted by the agent?
- Controllable-To what extent can the agent modify the environment?
- Historical-Does future state depends on the entire history or only the current state?
- Real Time-Can the environments change while the agent is deliberating?

There is no specific method to develop a good agent architecture. Best practice has been gleaned from work on the existing agent architectures [Shaw+96]. In agent research, there are several existing architectures that have been developed, such as Reactive Architecture [Aare+96][Wool+98], FIPA architecture [FIPA+97a], BDI Architecture [Bratm+88], Homer Plan architecture [Vere+90] and Hybrid Architecture [Raji97] that can be reused or learnt from it. The architecture presents the entities and organisation of entities to model the agent. These architectures are built according to the user requirements. Some agent architectures can be used for the solution of any agent system [Kirn92], but some are very narrow and applicable only to a specific agent system. The scrutiny process performed in Chapter 3 captured that the JAFMAS environment is based on COOL architecture, while the JADE and JATLITE environments are based on reactive pure state architecture with autonomy.

*Communication Environment*

The communication environment focuses on three levels of environment: *high-level, middle level* and *lower level.* The two latter levels are associated with the physical environment. The middle level specifies the coordination strategy that allows agents to interact with each other and the lower level specifies the communication channel used to transmit agent communication. Both issues will be discussed in the physical environment section.

The higher level communication environment focuses on two aspects: the method of exchanging information according to type of information and the sequence of agent interactions to achieve their goal. In Section 2.1.2.2 discusses this issue. The exchanged information may be an idea, knowledge or information and represented as text, graphics or others.

Referring back to the existing agent environments, JATLITE provides several options method to exchange information. Information may be presented as graphics or a text message. The JAFMAS only focuses on the exchange of a text information, while JADE provides additional functionality i.e. semantic languages to express the ontology.

In terms of interaction patterns, the JATLITE environment only focuses on peer to peer communication without providing a specific agent-interaction pattern. Designers choose their own agent interaction strategy i.e. JAFMAS agent interactions used Petri-net, while JADE provides the basic peer to peer type of agent communication. JADE also provides various patterns of agent interaction, such as contract-net, dutch-bidding, etc. as part of the functionality provided in the agent environment.

The agent interaction model represented in the *ASLD* stage is used to identify what kind of agent interaction patterns are used to present the solution of the agent system. The Filtering application design shows that all the information exchanged would be data text, while the analysis of all agent interactions revealed that the Filtering application applied the *Send* and *Request* of ACL performative.

*Physical Environment*

The physical environment describes the environment in which the agent system is executed, encompassing the hardware, operating system or middleware used to implement the agent environment functionality, usually known as the agent platform. The agent platform consists of an operating system and the computer system's coordinating program, which is built on the instruction set for a processor or microprocessor, the hardware that performs logic operations and manages data movement in the computer. The operating system must be designed to work with the particular processor's set of instructions. As an example, the Microsoft Window 2000 is built to work with series of microprocessors from the Intel Corporation that share same set of instructions. The criterion of identification of agent physical environment is provide an open agent platform.

The development of physical environment consists of two layers: *agent coordination strategy* and *physical communication.* The 'broker' is one of the agent coordination strategies, representing the principle used to provide the agent coordination[9].

| **Physical Environment Components** | **Techniques** |
|---|---|
| Agent Management System | *-Execution and monitoring active agents.*<br>Identification<br>Directory Services<br>Registration<br>Negotiations<br>Query/Search<br>Mobility |
| Security of Agent Platform | *-Secure transfer of message and objects*<br>Secure protocols<br>Digital signature<br>Firewalls<br>Language based security<br>Data encryption |
| Agent Communication Channel | *-Communication Functions*<br>Protocols-document format<br>RPC- remote programming<br>RMI - remote invocation programming<br>Object serialisation |

Table 4.3: The Principle of Providing the Physical Environment.

The coordination strategy is related with existing technology on how to present the strategy. The principles show in Table 4.3 influences the choice of coordination strategy.[10] The scrutiny of the existing agent environments shows that the JATLITE environment design is based on a star network strategy, whereas the JAFMAS and JADE environments use several groups to present a decentralised strategy.

The physical communication describes the physical agent transportation. There are several communication technologies that may be presented as requirements of an agent system such as HTTP, WCTP (Wireless Communication Transportation Protocol), RIO 500 Communication protocol, SMTP and WCCP (Web Cache Communication Protocol) that influence the design of

[9] Based on the literature review, there are three coordination strategies that are commonly used: centralised coordination strategy, decentralised coordination strategy and multi-level co-operation model [Fisc95][Durde00].

[10] These principles are identified as the result of the scrutiny of existing agent frameworks, synthesis of the existing architecture[Busch+96] and design patterns[Gam+95], including the CORBA design pattern[Mow+97] and Java Enterprise Design pattern[Grand01].

physical communication. We make the assumptions that the lower layer of transportation follows the ISO communications standard.

Appendix C, Figure 3.12 shows JAFMAS uses UDP communication channel through sockets. The JATLITE physical environment provides three alternative communication channels: HTTP, FTP and SMTP, whereas the JADE physical environment provides more alternative communication channels: HTTP, SMTP, TCP/IP, MS-Series, which use a standard method to provide heterogeneous distribution system platform.

The four issues of non-functional requirement focus on aspects of quality such as performance, intelligence, openness and reliability [Bosc00][Shaw+96]. These issues are related to identifying the right choice of technology solution to produce an agent environment.

Figure 4.18 shows the relationship between the Enterprise requirements and the Technology requirements. The technology is the constraint on the design of the agent environment. The identification of agent environment requirement is also related to agent technology. The agent architecture is related to agent decision-making technique, techniques of agent communication and physical communication, which is related to the platform technology.

| **Properties** | **Elements of properties** |
|---|---|
| Design autonomy | Platform<br>Interaction Protocol Language<br>Internal agent architecture |
| Directory Services | Whitepages, yellowpages, greenpages |
| Mediation Services | Ontology based/ transaction |
| Security Services | Timestamps/Authentication |
| Remittance Service | Concurrent/Billing |
| Operating Support | Archiving/Redundancy/Restoration/Accounting |
| Message Protocol | KQML, FIPA |
| Communication Infrastructure | Share memory(Blackboard) or Message-based connected or Connection-less(email) or Point to point or Multicast or Broadcast or Push and Pull, Synchronous/Asynchronous |
| Communication Channel | HTTP, FTP, OLE, CORBA, DCOM |

Table 4.4: Agent Environment Characteristics

In Table 4.4 summarises the properties of technology that influence the design of an agent environment. The AED process in this section is based on the common agent environment requirements highlighted in the discussion of identification of agent environment requirements.

Table 4.5 below shows an example of attributes presented in a JADE environment, which provide the four agent characteristic (behaviour (reactive autonomous), distribution, communication and openness).

| Functional Requirement | Attributes | |
|---|---|---|
| Agent Architecture | Agent (goal, role, resource) | Reactive, Autonomous<br>Agent states: alive, dead, busy, waiting |
| Communication Environment | Coordination strategic use<br>FIPA ACL<br>Use FIPA exchange message<br>FIPA ontology<br>Provides several types of agent interaction patterns | Directory services<br>(SLO, SL1)<br>( 20 types) |
| Physical Environment | Physical Communication channel<br>Platform | HTTP, FTP, SMTP, MSeries, TCP/IP CORBA(support multi platform), Java platform, |

Table 4.5: The JADE Functional Requirement

At this stage, the agent environment should be clearly identified, and the choice of technology specified. The next stage is to show the process of developing an agent environment based on the identification of the functional requirements.

### *4.2.2. Agent Environment Architectural Design (AEAD)*

The agent environment design aims to produce an agent environment architecture. The agent environment architecture design is viewed from two angles: the horizontal design and vertical design architecture. The horizontal architecture design follows the layer architecture patterns as shown in (Figure D1-3). The vertical design identifies the abstract and concrete components captured from the enterprise and technology requirements. Figure 3.12 shows the structure of the components and their relationships. The abstract components can be inherited as a subclass, while the concrete components are the components to be instantiated. The vertical design consists of at least three main issues: *agent architecture, agent communication* and *agent management* as discussed in Chapter 3 (derivation process). The different requirements and techniques used for these three components influence the transformation of the agent environment architecture.

The development of agent architecture design has a strong relationship with the choice of technology used to perform agent environment functionality. The techniques or technologies are the constraint on the architecture design of the agent environment. The derivation process (Appendix D) shows the use of architecture patterns and design patterns of the current architecture of design agent environment.

The agent environment design focuses on identifying components or objects and their relationships to produce an agent environment architecture, which conforms to a certain archetype. The architecture design is represented in five models as discussed in Chapter 3. Each models represented the structure of components and relationships.

*Agent Environment Architecture Design Process*

Figure 4.19 shows the development process of agent environment architecture as a recursive process from high level architecture refined into concrete agent environment architecture by structuring and refining the component.

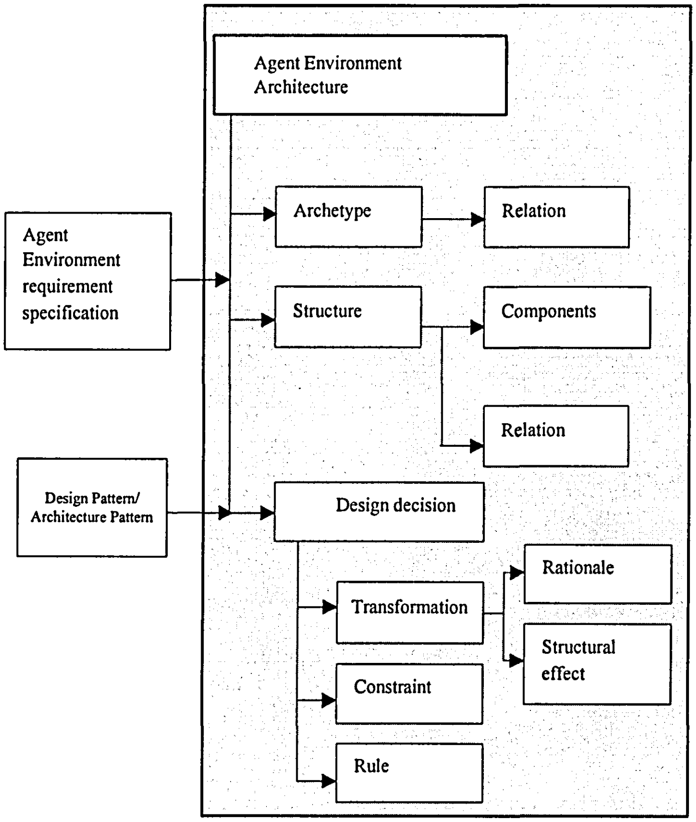


Figure 4.19: The Agent Environment Design Artefacts

The figure shows the artefacts of architecture design. The figure shows that the prioritised agent environment is a requirement able to identify a kind of architecture type, which the structure describes as components and relationships. The next agent environment requirement may produce another architecture type. The use of architecture patterns or design patterns guides the integration of various architecture types in providing a quality of an agent environment architecture type. The similar processes are performed until the entire of agent environment requirements have been take into consideration.

The scrutiny of existing agent frameworks in Chapter 3 showed examples of the abstract and concrete components/classes, and relationships between them. The concrete and the abstract classes are identified based on the roles of the agent environment. To decide how to structure the components, architecture patterns and design patterns are used. In this section, the patterns used in the development of the agent environment are discussed.

In this section, we show the development process of an agent environment based on the four agent characteristics: behaviour, distribution, communication and openness. The discussion focuses on how architecture and design patterns are used to develop agent environment for Filtering application.

The *first step* is to design the core agent based on the agent behaviours. In the Filtering application, this means having the behaviour of autonomy and reactivity.

The interaction model, agent model, internal architecture and agent platform influence the design of agent autonomy. Autonomy design focuses on synchronisation and concurrency. There are several patterns that address these issues: active object pattern [Lav+95] and ACE concurrency encapsulation patterns [Mats+93][Schm+95]. The core agent for JADE (Figure 4.5) and JATLITE (Figure Appendix C2-5) uses the reactive pattern as found in [Aarse+96][Kend+96] and the pro-active pattern combination of autonomy and real-time found in [Kend+00]. The composite pattern [Gam+95][Rieh97] is used to describe the concept of reuse of the components of an autonomous agent.

The *second step* is concerned with the agent communication characteristics. Mediator, Sensor and Adaptor Patterns are used for agent collaboration and interactions between agents. At this stage the agent is like an object; delegation can enhance reusability, while proliferating interconnection increases complexity and reusability because every time the agent needs to know every other object. The Mediator pattern [Gam+95], and Sensor and Adaptor [Copl92][Molin+96] are used for the aspect of agent interaction and collaboration. A Mediator is responsible for controlling and coordinating the interaction of a group of similar agents. This intermediary does not allow the agents to refer back to each other directly. It provides too many interactions and can be a drawback if a communication bottleneck occurs. The adapter pattern is applied as effectors of each agent, while the sensor pattern is applied as a variant to the adapter pattern. The combination of these patterns is used to provide collaboration and interaction between agents, (refer [Kend+96]); therefore, agents can apply structured messaging.

The *third step* uses the agent distribution characteristic to design agent environment. The analysis of the organisation model and agent model is used to determine whether the agent environment should be presented as a distributed location or not, and whether the agent environment is homogeneous or heterogeneous.

The homogeneous agent environment occurs when all agents are based on a similar agent platform, for example, all agents are Internet based, in which case the use of the Internet client-server architecture is more appropriate. The heterogeneous agent environment is applied when there are several types of agent platform in the agent system. The role of patterns as captured in Appendix D presents the distributed architecture patterns: homogeneous and heterogeneous, locational and logical distribution.

The proxy pattern [Gam+95] is used for the solution of decentralised agent management. Each agent would require an instance of a proxy for every interface that it supports and it would have to maintain information and knowledge of every agent that wanted to interact (see [Kend+97]). The proxy controls access to the real subject and also provides a distinct of interface. For example, the Fipa proxy would subscribe to Fipa interface and KQML proxy would subscribe to KQML interface.

If the agent collaboration occurs via coordination protocol, then the agent must be able to determine its behaviour according to the state of the coordination protocol (see [Kuwa95]) in which it is engaged.

In order that agents engage with several conversations simultaneously, Momento patterns [Gam+95] are used to externalise an object's internal state so that the object can be restored to this state later. A combination of proxy pattern and Momento patterns can support languages, which resolve conversation, i.e. store and recover their state (see [Kend+96]).

The combination of Mediator patterns with these patterns will provide a structure for agent cooperation. The proxies would be responsible for intercepting messages for the agent language while the Momento would store the conversation in which the agent is presently engaged (see in [Kend+96]).

The *fourth step* is to use the agent openness characteristic to design the agent environment. The concept of openness was discussed in Chapter 2. The openness characteristic applies to heterogeneous agent environments where agents have different agent platforms, and so need an open society that enables an N-to-N connection.

Figure D-1(Appendix D) shows the use of Broker patterns, which leads to CORBA patterns, like an open society that provides for communication and location transparency for objects/or/and agents to act as a client and server to any agent or object that is registered. The agent or its proxy can become a virtual member of any open society.

In broker patterns, agents will act as clients and servers for one another, so the broker is responsible for locating a server once a client has requested a service. Both client and server must register themselves with the broker (e.g. agent management in JADE). Both server and client employ proxies that respond to the interface known by the broker. The proxies may be located at different addresses. The openness presented in CORBA patterns provides proxies that subscribe to a certain interface. Thus, the client and server do not have to have direct knowledge of these interfaces. The use of bridges is for forwarding requests and responses to another broker who manages another society of clients and servers (for details see [Mow+97][Kend+96]).

The use of various kinds of directory services, mediation services, security services, remittance services and communication infrastructure as shown in the Table 4.5 will influence the structure design of the agent environment architecture.

### *4.2.3. Component Design (CD)*

The component design adapts the object oriented component design as object, classes or as a micro-architecture similar to [Garl+00] which focused on component design and [Kris96][Kris+96] which focused on object-oriented roles modelling which is a similar strand to component design. A component may be presented as a micro-architecture to perform a specific function. A component may developed from scratch[11] or reuse the existing components.

The research found the current practice of development component design uses design patterns and idioms. It can reuse the encapsulation of components so that they can be isolated enough to handle future changes. The Gamma pattern languages are primarily concerned with the development of effective design patterns of component level.

Components design consists of a group of cooperating objects whose interrelationships with other objects occur according to well defined and understood component solutions. The following are some design patterns are used while structuring the agent environment architectures:

[11] Follow the traditional object-oriented of component development proposed by [Garl+00] and [Kris96][Krist+96]

- Automated reasoning patterns - which use strategy patterns and interpreter pattern [Gam+95]. The belief, goal and plan are interpreted via such patterns.
- Composite patterns - which reuse any abstract components such as agent, behaviour, message and ontology [Gam+95][Rieh97].
- Patterns for databases access - used for information exchange using adaptor patterns [Copl+95] and the use of objects. Record is a pattern that facilitates exchange of information between relational databases [Farh96].
- Mobile patterns - is a special pattern when the agent has to move from one location to other location while interacting. There are other related patterns for mobile agents as shown in [Oshi+98].

### *4.2.4. Development of JADE Agent Environment*

The aim of this section is to show the development of the JADE agent environment by re-casting the JADE development in terms of the proposed of the method for AED as discussed in the previous section. We describe the development briefly based on the three stages of AED and the modelling, captured by scrutiny and reverse engineering with cross-reference between stages.

*The first* identifies JADE agent architecture. The basic JADE agent provides an autonomy and pure reactive stated behaviour (abstraction of agent model as portrayed in Figure 3.5). The individual JADE agent architecture consists of components capabilities, roles, actions, communication with other agents and internal communication and resource.

Physical environment- JADE environment supports the distributed heterogeneous agents, which use for agent communications such as SMTP, HTTP and TCP/IP and identify the method for managing agent interactions and managing the various channels of communication used by the agent.

Communication environment- peer-to peer interaction and support multi-cast using inter-mediator agent.

Interactions environment – support agent communication languages and ontology using semantic languages.

The *second* is to analyse the structure of the JADE environment, which shows the components and their relationships. From the analysis of the JADE requirement specification, there are three important components are identified: agent, communication and management components, which applied the CORBA patterns.

Using CORBA technology, the client and CORBA service management named as *Agent* and *FIPA-Services* respectively The Fipa Services consists of several other components such as Directory services, message-transport services, agent platform and others. The agent management is identified as the *Directory services.* Figure Appendix C3-2 shows the high-level of JADE environment architecture. The Directory facilitator component is an additional component to provide the agent environment with openness, so any agents registered with the directory facilitator are able to access any agents.

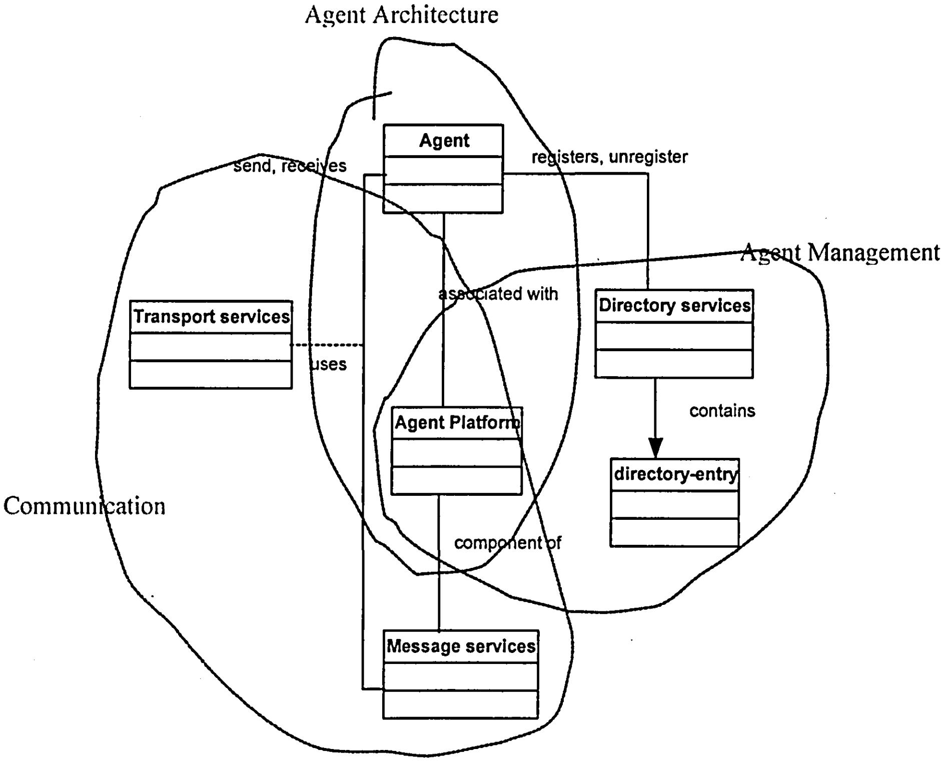


Figure 4.20 : The Structure of Relationship between the Components.

In order to develop the agent environment, the identified components are linked to each other using patterns that guide how these components are linked together in a context, for example, the relationship between the agent, agent-platform, directory-services, transport services, message transport-services and directory entry. Figure 4.20 shows the structure of the JADE environment that consist various components and their relationships.

The *third* decomposes each component by identifying the sub-components and their relationships. It describes the structure of the components. For example, agent components are presented as FIPA agent architecture. The agent architecture is based on FIPA architecture, which consists of the following components: *messaging, ACL* and *directory-entry.* The message consists of content message and ontology. The message service consists of the transport-message and type of transport.

The director-entry components can be decomposed into several other components, such as agent entry attributes, locator and entity-name, while locator is described based on the transport-description, transport type, transport address and transport properties. Each of the components is linked to the others in an archetype. Each component is decomposed into sub-components until the relationships among all components are presented. We are not going to describe all the diagramming of the components and sub-components but only the process. This is because it follows the object-oriented approach.

The *fourth* is to develop the organisation of the components to present the JADE environment. The use of design patterns and a framework development process makes it possible to identify the organisation structure of the components. Figure 4.21 shows an example of the JADE environment class diagram to illustrate the structure of the agent environment architecture.

The objects and classes are language-dependent in the case of class libraries and reuse frameworks, for example, when the class definitions are expressed in OMG IDL. In some conditions, standard object models are used, such as the object model defined in CORBA that defines the semantics of object interactions. The CORBAServices interface, for example defines how the basic object level interfaces for the management and control of objects consists of the Naming service object, Event services object, Life cycle object, Persistence service objects, Relationship service object, Transaction Service object, Externalisation service object, and object concurrency and locking.

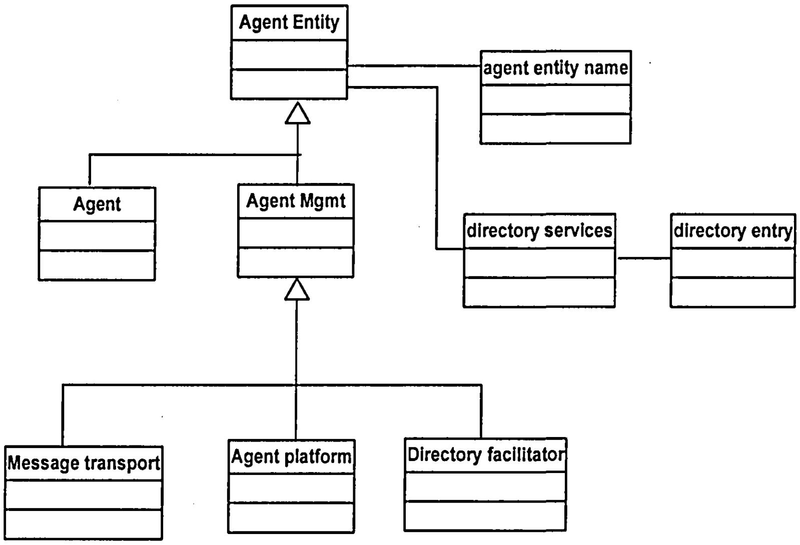


Figure 4.21: The JADE Organisation Architecture.

The concept of reuse should apply in both *development agent system* and *development agent environments*. The reuse of the existing agent environment can be identified based on the similarity of the agent environment functional requirements. Those requirements are used as a guide to the choice of suitable of existing agent environment from those available, rather than developing a new one. Chapter 2 has discussed several agents environment available for reuse, while the previous section discussed the artefacts of the agent environment functional requirements. The development of an agent system can use any of those agent environments if the requirements of the agent environment are similar to the core environment requirements.

In DAS phase, the reuse only applicable in two conditions: both agent systems perform in the similar agent environments or if different the agent environment should provides openness criterion. This reuse focuses on providing agents with services that may be used in other agents (refers section 2.4). So, the agent service is reused by other agent system, which needs the similar services.

Despite these requirements, other conditions that influence the decision to reuse any agent environment are good support, an open source code, etc., discussed in more detail in next section.

Our discussion in this section focuses on a high-level design in the process of developing agent environment. This is because we realise the development of agent environment part could use the framework application development process, which defines the components needed for the agent environment, so that later each component can be designed based on object oriented development. Each component is then implemented, executed and tested.

## 4.3. REUSE OF 'OFF-THE-SHELF' AGENT ENVIRONMENT (OFSAE)

OFSAE is an executable of agent environment built for reuse purpose. Most off-the-shelf agent environment provides the agent features as discussed in Chapter 2. Reuse the off-the-shelf agent environment reduces agent development time. As discussed in Chapter 2, there are various off-the-shelf agent environment been developed [Tools99][Toolkit03], which limited to a specific agent domain solution such as mobile agent versus multi-agent system or single agent. It also different on the architecture and techniques are used to represent the agent environment it self. There are various variables can differentiated them. In this case, choosing the right OFSAE is important. In this section, we proposed development process for reuse the OFSAE as guideline for agent developer.

### *4.3.1. Process to Reuse the Off-the-Shelf Agent Environment*

Figure 4.22 shows the flows of activities of reuse OFSAE, which consists of four steps, each discussed next. The agent environment analysis is similar process to agent environment analysis in development agent environment (as discussed in previous section). However, in here we provide common agent environment specification as guideline to produce the agent environment specification.

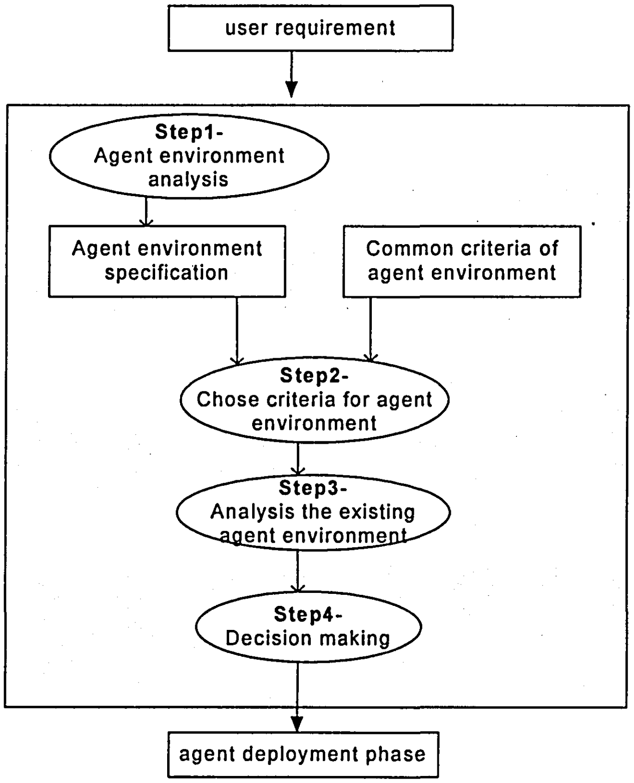

user requirement
Step1-
Agent environment analysis
Agent environment specification
Common criteria of agent environment
Step2-
Chose criteria for agent environment
Step3-
Analysis the existing agent environment
Step4-
Decision making
agent deployment phase

Figure 4.22: Reuse off-the-shelf Agent Environment

Step1- *Agent environment analysis (AEA)*

The analysis of the agent environment is similar to the AEA stage for development of a bespoke agent environment. The aim of analysis of the agent environment is to produce a concrete agent environment requirement specification for agent system solution. This step aims to produce the agent environment requirements, which divides into two categories: *enterprise* and *technology* requirements. The enterprise requirements state the requirements relevant to the features of an agent environment needed for agent execution and the technology requirement is the requirement relevant to the technology used to present the architecture, physical and communication environments to support the features of the agent environment. The process of identification of the agent environment is as discussed in section 4.2.1. The agent environment properties as shown in Table 4.5 and the agent environment requirement references as shown in Figure 3.12 as guidance to the feature that may be part of the requirement. The physical environment includes the platform as well, for example, whether it supports DCOM, CORBA, HTTP, and SMTP.

Step2- *Choose the criteria for selecting agent environment*

The aims of this step is to identify appropriate the agent environment criteria. The criteria are captured by comparing the agent environment requirement specification produced in previous step with the list of common criteria provided to agent environment. The common criteria for agent environment presented in here are captured from analysis more then fifteen existing agent environments as discussed in later section. The list chosen criteria for the agent environment are prioritised.

Step3- *Analysis the OFSAE*

The analysis of OFSAE is performed by analysing the existing OFSAE that most fulfil the prioritised criteria for the chosen agent environment produced in the previous step.

Step4- *Decision-making* - choose the agent environment.

The result of analysis in the previous steps should provide at least three existing OFSAE that most fulfil the prioritised criteria. At this stage, further analysis is performed by evaluating the performance, identifying features for enhancement of the system and testing the system. The designers have to justify their choice according to what best fulfils the prioritised criteria.

| **Product /Criteria** | **JADE**<br>**[JADE98]** | **JAFMAS**<br>**[JAF97]** | **ZUES**<br>**[ZUES00]** | **JATLITE**<br>**[JAT97]** | **Agent Builder**<br>**[BUIL00]** | **April**<br>**[APRIL00]** |
|---|---|---|---|---|---|---|
| AI features | 2 | 3 | 2 | 2 | 3 | 2 |
| Open architecture | 3 | 1 | 3 | 1 | 2 | 2 |
| Extension | 3 | 2 | 3 | 2 | 2 | 2 |
| ACL | FIPA | KQML | KQML/ FIPA | KQML | KQML | KQML |
| Support Distribution Location | 3 | 1 | 3 | 2 | 3 | 2 |
| Basic properties | Autonomy reactive | Autonomy reactive | Autonomy reactive | Autonomy reactive | BDI | Autonomy reactive |
| Support Other behaviour | Stationary<br>Reasoning<br>Mobility | Stationary<br>Reasoning | Stationary | Stationary | Stationary | Stationary |
| Availability | 3 | 3 | 3 | 3 | 2 | 2 |
| Applicable | 3 | 1 | 3 | 1 | 3 | 2 |
| Easy of to use | Yes | Yes | Yes | No | Yes | No |
| Deployment guidance | 3 | 2 | 2 | 1 | 2 | 1 |
| Good documentation | 3 | 1 | 2 | 2 | 2 | 1 |
| Language | Java | Java | Java | Java | Not | April lang. |
|  |  |  |  |  |  |  |

Table 4.6: Example of Comparison Criteria of the Existing Agent Frameworks

Table 4.6 shows an example of the outcome of evaluation of several features of agent environments. The agent environments listed above fulfil the criteria of multi-agent system frameworks. The analysis is based on the observation and testing of the environments.

The features of the agent environments listed above vary subject to the common features required of the agent environment itself.

The results of the five or six agent environments need to be further investigated in terms of the features by evaluating the product by installing and testing the system, in order to justify which agent environment is most appropriate to be used. This technique is the fastest technique for developing an agent system. It can reduce up to half of the whole agent development process.

### *4.3.2. Common Features of an Agent Environment*

Due to different perspectives of the development of the existing agent environments, a complete characterisation of the existing agent environment is a bit far to be feasible. There are various issues can be abstracted the similarities of the enterprise and technology perspectives and other additional issues are considered as we categorise into the three following issues: agent capabilities, technical and software development. These criteria were captured through observation of more the fifteen existing agent environments available in [Tools99][Toolkit03]. The following discusses further detail of each category.

**Agent Capabilities issues**

The agent capabilities describe the functionality provided to agents, which we classify into several following issues.

- Basic behaviour - as discussed in the previous chapter, there are several different properties that may provides to agents as we discussed in section 2.1 such as autonomy, situatedness, responsiveness and sociality. These properties may be mandatory or optional. The mandatory ones are the basic properties that must be provided to the agent by which we name the agent types. For example being autonomous and reactive or autonomous and responsive using BDI are mandatory attributes, so the agents are called reactive or BDI agents respectively.
- Agent Communication Language - provides features of using ACL, which provide the criterion of openness. There are two fully specified ACLs: FIPA and KQML. The agent environment may be restricted to a specific agent communication language or open to other agent communication languages such as KQML or FIPA communication language. However, observation suggests that the latest research uses FIPA ACL rather than KQML.

- Information exchange - is viewed at two levels. First is the application level, which describes the method of information exchange, whether by blackboard method or message passing. The second is the technique used for data transmission; for example, JADE provides choices of HTTP, TCP/IP or SMTP, similar to JATLITE, while JAFMAS uses USD socket.
- Coordination protocols - describes the method of agent coordination. Do they provide the coordination and negotiation protocol as proposed by [Jenn93] and [Krau97]? The availability of a library of various coordination protocols is useful for future enhancement.
- Human-agent interaction - how is intelligence functionality provided to interact with humans such as speech act recognition, natural language, etc.
- Mental state - some agents do not provide BDI as mandatory, but rather as a mental state. This is an additional agent property.
- Deliberative capabilities - the agent explicitly represents its objectives, reasons about them, and which provides a plan to achieve the goal.
- Adaptivity - the ability for agents to change strategy to adapt the evolving environment and learn from previous experience.
- AI features - An important feature of intelligent agents, which have the following characteristics: ability to perform autonomously, which presents a certain level of artificial intelligence. Integrating environment with AI techniques such as rule-base, fuzzy logic and neural networks has many advantages.

**Technical issues**

There are several technical features desirable in an existing agent environment as follows:

- Support distribution - The distribution can be conceptual only or a combination of both conceptual and locational distribution.
- Support mobility - Mobility needs a special kind of environment to support the ability of agents to move to different locations.
- Open architecture - The open agent environment means that agents in the agent environment are allowed to interact with other agents in a different agent environment. This supports heterogeneous agents. For example, JADE presents an open environment various other agent environments can communicate openly with JADE, such as FIPA-OS, ZUES. JAFMAS does not meet the openness criterion because has not yet been tested with other agent architecture.
- Support ontology - Ontology is part of ACL, some agent environments provide ACL features but do not support ontology. The ontology supports the openness of a domain sending message.

- Support execution platform - on what platform the environment is developed, whether restricted to Microsoft Windows or Windows NT or Unix only rather than both platforms.
- Concurrency - must posses its own thread of control. Thus, agents find a natural implementation in concurrently executing processes.
- Security - important issues when mobility and distribution come into play. It must be certain that sure the security aspect has been is prioritised and tested.
- Real time control - depends on the application, the need for real –time control and response may arise.
- Specific platform - the platforms where the agent environment can be installed should be considered; also languages, the resources required and the availability of the source code.

**Software development issues**

There are various software development issues influences the desirable of choosing the existing agent environment as follows:

- Licensing - The terms and conditions of licence is an important requirement. Some agent environments are free, others payable with various term and condition. Since the agent environment is part of the development of the agent system, an open source is an important criterion to be considered for extension purposes. An open source allows the designer or developer to add any functionality needed for maintenance of the system.
- Level of documentation - An agent environment cannot be reused without having good documentation of how to use it.
- Supportive - Support from the agent environment developer is also important when encountering a problem while deploying and using the agent environment. Some agent environments provide training and good support.
- Level of usage - Indicates the level of applicability of the agent environment. A good agent environment should be tested in several types of agent system to see the usage level. Some agent environments are developed without being tested. Looking at successful systems running using the agent environment can identify this.
- Ease of use - The agent environment should be easy to reuse. Examples of agent applications that have been constructed demonstrate the ease of use of the agent environment. On the other hand, documentation of use is also taken into consideration, for example provision of an agent skeleton. The availability of an agent ‘skeleton’ that is easy to plug in helps the agent developer in building correct applications faster and more easily.

- Reliability - A reliable agent environment provides features that easily integrate with other non-agent system environments.
- Languages - The agent environment should be executable. It should be developed using a specific programming language. Knowing the language used to implement agent the environment is also important for maintenance, constructing agents and enhancement.
- Used and known - the agent environment provide the most well-known technology.
- Utilities - Some agent environments provide facilities for debugging the multi-agent system, because the nature of the system is distributed and concurrent, a difficult and complex task. Utilities facilities can be used to ensure development is correct, reliable and robust system.
- Ease of adding other features - In some agent environments it is easy to add other agent features while in others the scope is very limited.
- Software engineering support - Some the agent environments support the software engineering perspective e.g. providing well-defined methodology, ontology analysis, prototyping.

## 4.4. AGENT SYSTEM DEPLOYMENT(ASDO)

At this stage, the agent environment has been developed or identified. The main objective of this phase is to integrate the ASD result with the components that have been developed or provided in the agent environment to produce an executable agent system. The ASDO phase comprises three subsequent stages:

- Deployment design,
- Implementation,
- Execution and Testing.

The deployment design is a part of an integration process of the individual agent design result with the components in agent environment to produce an executable agent system. The deployment design modelling uses the traditional object-oriented concepts such as inheritance and instantiation. UML modelling such as class diagram, sequence diagram [Fowl97] is used to present the deployment design modelling. The individual agent design modelling is transformed into an object oriented design modelling.

The implementation process is similar to the object-oriented implementation process. The design at deployment stages presents at object level, therefore at this stage it is able to transform the deployment design into implementation.

The testing process consists of three stages. First are the execution and testing of each individual agent and making sure each agent has fulfilled the requirements. When all the agents have tested, the second stage is testing the communication of each pair of agents until all communications are tested. The last stage is executed by all the agents together and test the system based on each event capture in scenario.

### *4.4.1. Deployment Design Process*

The agent overview as presented in Figure 3.5 is the abstraction to present the agent model (autonomous reactive). However, the deployment of each agent is tied to the structure of agents as described in the agent architecture, which is identified in the AED phase. Therefore, this section uses the case of FIPA agent architecture to show how the deployment design process is performed in JADE.

The ASDO modelling is an integration of the agent environment components with the individual agent design. The following are some of the reused components of the agent environment as we are going to use to show the integration process.

*CoreAgent* is a component or interface that defines all basic properties needed for all agents.

*CommunicationAct* is a component for communication acts.

*MessageTemplate is* a template for communication acts including specific ontology.

*AgentCapabilities* – are components that define the agent capabilities associated with agents that may be presented by certain kinds of agent, such as *simplebehaviour, cyclicbehaviour* and *proactivebehaviour.*

*Interaction Protocols*- there are many types of interaction protocols components such as FIFA-auction, FIPA-contract net (more in [FIPA+97b]).

*Communication Channel*- many kinds of communication channel may be used.

These components were captured from the scrutiny of several agent frameworks in Chapter 3. There are several other components to support other kinds of functionality. The ASDO modelling shown in this section is based on these common components, to illustrate the deployment process.

The ASDO modelling process consists of six steps as listed below:

*Step 1: Creating agent*

Each agent at *IAD* is transformed then deployed with its agent environment to produce an executable agent system. The method of creating agents depends on the design of the architecture of the agent environment. Creating agents means creating a class that inherits the core agent, as the class can present itself as an agent. Based on the literature, there are three kinds of agent classes and instances: single inheritance, multiple inheritance and dynamic inheritance that are used to define the type of hierarchy, support reusability and changeability and share common behaviour.

Examples of these different kinds of agent classes and instances are: *Agentclass, Agent class/role-1, role-2,* .. and *AgentInstance/role-1, role-2, :Agentclass*. The first denotes an agent class; the second is an agent class satisfying specified roles and the last one defines an agent instance satisfying specified roles. Based on the descriptions of agent, the agent class is abstracted as shown as Figure 4.23. The agent class shows an extension of UML notation.

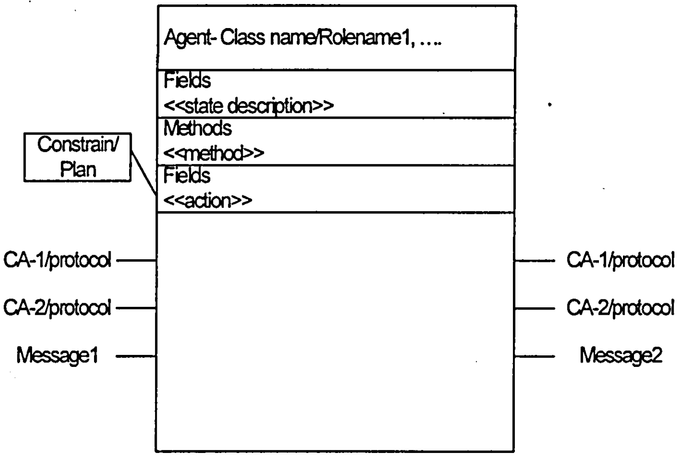


Figure 4.23: UML Class Diagram to Specify Agent Behaviour

Agent class name is created depending on the kinds of inheritance. The following is an example of creating an agent using a single inheritance by taking the example of the UserAgent. The CoreAgent is the interface to describe an agent. Different CoreAgent defines different types of agent. For the purposes of this modelling, we focus on similar types of agent. The Figure 4.24 shows the composite patterns to create agent.

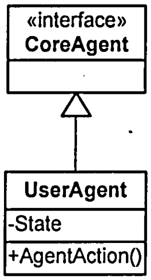


Figure 4.24: The Inheritance of Agent Abstract Class

The AgentAction describes an agent method. The agent action is associated with the agent roles. There are two types of agent actions. The first action implies direct communication and the second action implies interaction. Direct communication may change the agent's state, resource, or environment. The *agent plan model* describes the flow of agent actions. Figure 4.14 shows an example of the flow of agent action.

*Step 2*: Creates Agent class/Rolename

The second step is to assign the agent roles. The previous design modelling showed that the UserAgent has two roles: Enquiry and Feedback. Therefore two UserAgent role classes are created. The behaviour of agent roles depends on the behaviour of conversation between two agents by (interaction protocol or same other method). For example, a *Simplebehaviour* describes the capability of the agent roles to send a message to another agent without the other responding to the message. *Cyclicbehaviour* provides for the interactions between two agents until the session of the conversation is finished. Other examples of agent role behaviours are described in [FIPA+97b]. Each agent role is associated with actions. Referring back to the example, the two agent roles are shown as Figure 4.25. At this stage, any constraint, permission and agent states are illustrated.

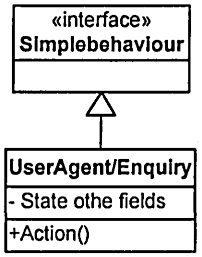


Figure 4.25: Assigning an Agent Role with Simplebehaviour

*Step 3*: Identify action for each agent role.

Agent role action defines the flow of tasks associated with the role. Other tasks are defined as a method or other object class.

*Step 4*: Create task class associated with Agent action and agent role action.

The tasks associated with agent action and agent role are created using the traditional object oriented approach.

Based on these models, an agent class diagram is identified. At this stage, the agent represents as a collection of objects. At this stage, the agent design is based on the object-oriented approach and uses UML design modelling. A collaboration diagram and sequence diagram are developed to represent the relationship between objects to present agent activity, when necessary. Each agent is then implemented and executed. Each agent is tested based on several events, captured as stated in the agent data model.

## 4.5. SUMMARY

This chapter has presented the whole agent development by showing the agent development process and modeling process. The artefact of agent development method presents the life-cycle of the agent development process as a recursive process of three following phases: a*gent system development, agent environment development* and *agent system deployment.* Each phase consists of various stages of development process. The *agent system development* phase consists of three stages: *analysis requirement, agent system level design* and *individual agent design.* The modelling process is presented in each stage of each phase. There are nine modelling processes representing the agent system development phases: problem domain model, system goal, use case model, scenarios, identifying agent paradigm, identifying agent and interactions, system plan model, individual agent model, internal plan and internal task.

The method provides two alternatives approaches for development agent environment: bespoke and off-the-shelf. The bespoke approach as we define as *agent environment development* phase consists of three stages: *agent environment analysis, agent environment architectural design* and *component design,* which contains with five analysis and design modeling: o*rganisation model, agent model, interaction model, task model and domain Model.*

The off-the-shelf approach contains four steps. This includes guidelines for choosing an existing agent environment for reuse. A set of agent environment characteristics used to classify the existing off-the-shelf agent environments are provided.

We believe that the proposal of the agent development process presents a systematic agent development. It consists of a step by step process in each phase and shows the transformation modelling from one phase to another phase. Each stage provides a deliverable of documents as the outcome of the modeling process.

In summary, the proposed of agent development method presents a clear picture of the whole process of development of an agent system from analysis to implementation and execution. Even though at the deployment stage, we restricted the modelling to the *JADE* agent environment, we are able to show a complete development process that designers can "fake".

The next chapter is to apply the proposed of agent development process in two case studies, to show how to use the methodology and to evaluate the applicability of the agent development process.

# CHAPTER 5

# 5 Development of Agent System- Case Studies

## 5.0. INTRODUCTION

The objective of this chapter is to demonstrate how to use the methodology presented in the previous chapters. Two case studies of the agent development system are used to evaluate the applicability of methodology. The first case study is forward engineering of the Filtering application[Male+00], named the Filtering agent system[Othm+03], the second case study is a Diabetic Consultation system. Both case studies are complex enough to show the applicable of the proposed of agent method. The use of the agent method for Filtering application is able to reduce the user workload due to the constant changes of information on the Web[Maes94], increase performance of the system and provide a flexibility to the system (users and information located anywhere), whilst the second case study is chosen because it presents the criterion of distributed intelligence in making decisions, reducing consultation service and increase the level of efficiency and accuracy of treatment to diabetic patient[Othm+02c][Cash00].

Both developments of the case studies presented here aim to show how to apply the agent method. Both developments show the milestone of agent development process by presenting a set of documents and modelling that should be delivered in each stage, and shows the transformation process from one stage to another stage. By contrast, the traditional development of agent systems when our research started was performed in an ad hoc fashion. It only provided an implementation code and possibly very general requirements. With such a set of deliverables, the system is not only easy to understand but also benefits of easy maintenance and enhancement.

The first section presents the development process of Filtering agent system, while the next section presents the development process for Diabetics Consultation System (DCS). As stated in our methodology, the development process is a recursive process. However, the deliverable of documentation is presented as a sequential process of phases and stages.

## 5.1. FILTERING AGENT SYSTEM (FAS)

The discussion in the previous chapter focused on the requirements of the Filtering application[Male+00]. The case study presented in this section is a forward engineering of the Filtering application, which provides additional requirements as described in the Filtering agent system requirement specification below.

The Filtering agent system development process delivered four type of documents: requirement specification, agent system design, identification agent environment and detailed design. The agent system design consists of three stages of deliverable documents: *analysis requirement, high level system design* and *agent level design*. Each document is explained in the following sub-sections.

### *5.1.1. Requirement Specification*

The first stage of our method is to analyse the user requirements. The following user requirement is the result of the requirement analysis. The requirement analysis in next section shows the process to produce the requirement specification. The analysis process shows the transaction process between requirement specification and analysis process. The requirements specification of the FAS are listed as follows.

1. The system is able to search articles from the web using specific keywords provided by the user. The article results are downloaded and ranked according the user profile value before being displayed to the user.
2. The system allows any user from anywhere to use the system. The system is a multi-user based system. The user needs to login to the system to be able to create or identify their user profile. The user profile stores the user name, article names and fitness values. An invalid user is designed as an anonymous user and an automatic user profile is created. The user is asked to provide detailed personal information.
3. The level of filtering when downloading the articles is ranked, based on keyword and category. Only the first 100 highest scoring articles are downloaded. Each article downloaded consists of the article's title, address and article content. Any search engine can be used in here (e.g. Google). The content of articles is given an initial weighted value. After all articles are downloaded the final weighted value is calculated. Each article is ranked according to the user profile fitness values. Then, the articles are displayed to the user according to the ranked value is calculated. The ranking process is based on each indexed document using the following similarity function [Othm+02b].
4. The system will capture the latest search profile that is used as the default profile when creating user profile.

5. The user is allowed to select the articles, to read the content and evaluate how far the articles of are interest to him/her. The most interesting articles are selected and the user profile updated. The feedback value provided will change the user profile using a genetic algorithm.
6. The user is allowed to specify the time when the search result is expected to be delivered: immediately, weekly or monthly. Based on the searching timetable, the system will search articles from the web to find new articles. When the user logs in to the system at any time, the new articles are displayed. But if the user is online, the new articles are displayed to the user, whereas the user receives the notification of finding new articles through email and the articles are displayed every time the user login the system.
7. The new articles are captured by comparing the previous indexed document, using the same keywords. The index process is uses the representation of a set of features derived from the document collection. The representation is based on a list of keywords captured from the subject of the document. However, before becoming features these words will typically have the case of their letters normalized, and certain types of words, such as prepositions, determiners, and pronouns removed from the feature set. The representation of documents involves the assignment of a numerical weight to all terms in a document collection.

Based on the requirement, the system presents the agent system criteria. The huge amount and the fast changes of information in the Web cause a high workload to users to find accurate information relevant to the user's need. Due to the huge data the FAS consists of various long and complex tasks that can be performed in parallel. The used agent (autonomous reactive) take advantages of this problem. Separation of long and complex processes into agents that autonomously find information and perform complex tasks such as indexing and filtering will reduce the time needed for the user to find information[Othm+03]. In addition, millions of multi-users can access the Internet and reuse similar information, and the long and complex process can be performed in parallel, which will increase the performance of system(access time).

In addition the information and users are distributed (not fixed location) as this is a criteria of the agent system. The information format is unstructured sending to the agents utilised by the ACL. The reuse of searching result and indexed document to capture patterns of other interested users reduces the processing time. Further more, the machine learning provided is able to learn user behaviour provides system with adaptive behaviour.

The user requirement specification stated above is a document provided by the user. Our method begins with analysis of the user requirement that may present in parallel in phase 1 and phase 2. The phase1 analysis captures requirements for the agent solution while the phase 2 analysis captures the requirement of using existing agent technology to perform the agent system solution. The requirements stated in the previous section do not state that any specific technology is required to perform the system. Therefore the agent environment requirement is identified by evaluating the appropriateness according to agent environment requirements identified after analysis of the result of agent system design.

Follow the method, the development process consists of three phases: agent system development, agent environment development and agent system deployment, all phases are discussed in the following subsequence sections, while the stages in each phase are discussed in the consequent subsections.

### *5.1.2. PHASE 1: Agent System Development*

The agent system development is the first phase of our agent development. As stated in the section on agent development method, agent system design consists of three stages: *requirement analysis, agent system level design* and *agent level design*. The agent system follows the flow of agent system design as shown in Figure 4.2 and the associated modelling notation as shown in Appendix E. Each modelling process is discussed in the methodology. The deliverables of each stage are discussed in the subsequent sections.

#### 5.1.2.1. Analysis. Requirement of Filtering Agent System

Following the analysis stages of the methodology, the analysis modelling process transfers the user requirement specification into analysis modelling. The analysis modelling captures the system goal, captures the scenario, develops agent use cases and identifies agent abstraction.

*Capturing the System goal*

*The first step* is to identify the goal of the system. The goal of the system is identified from the objectives of the user requirement. The objective of the system is to search articles from the web. The goal of a system is similar to the objective, but the system goal represents a certain criterion of the objective. The goal of a system is to *find articles from the web that are most near to the user's interest.* The criterion of approximation to the user's interest is what differentiates the objective of the system from the goal of the system.

The next step is to find the activity that leads to achievement of the goal of the system:

1. The requirement that fulfils the objective.

   The user provides keywords, then the system searches articles from the Web and the result is displayed back to the user (R1).

2. Find requirements that influenced, the achievement of the system goal.

   a. Filtering the articles based on the user profile (R2, R4)

   b. Provide a more precise user profile

i) constantly update the user profile near to the user interest (R5, R6)

ii) use genetic algorithm to create and update the user profile (R5)

c. Based on the timetable, the system keeps searching the articles from the web to find new articles (R7)

The methods a, b and c are captured in the requirements that influence the achievement of the system goal. Methods a and b(ii) are provided when designing the system, whereas methods b(i) and c are associated with the occurrence of the event.

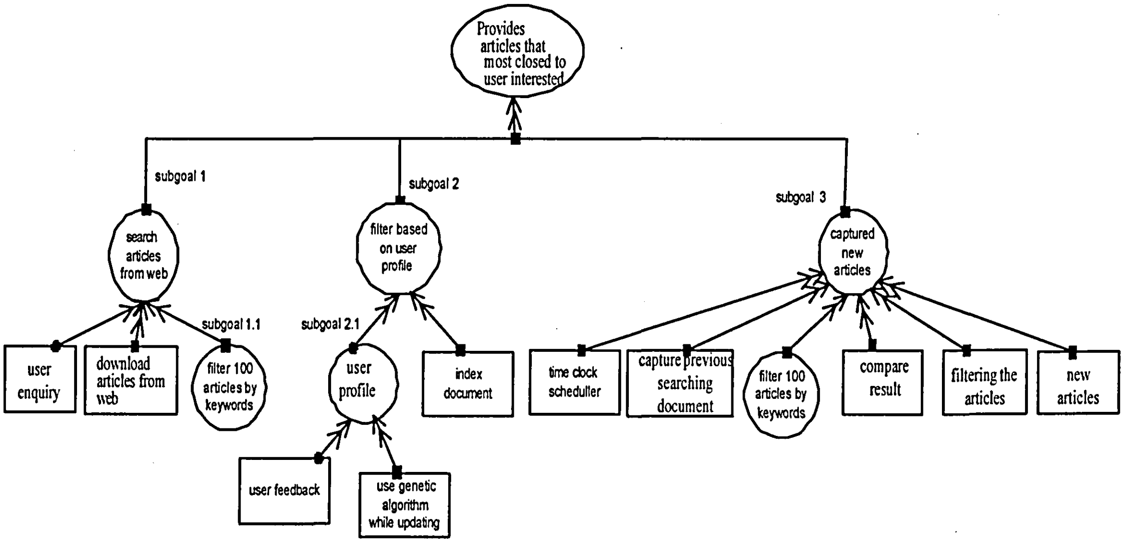

Provides articles that most closed to user interested
subgoal 1
subgoal 2
subgoal 3
search articles from web
filter based on user profile
captured new articles
subgoal 1.1
subgoal 2.1
user enquiry
download articles from web
filter 100 articles by keywords
user profile
index document
time clock scheduller
capture previous searching document
filter 100 articles by keywords
compare result
filtering the articles
new articles
user feedback
use genetic algorithm while updating

Figure 5.1: The Filtering System Goal

These three methods are stated in the user requirement. The research in this area is concerned with finding other techniques that influence the achievement of the system goal.

Using the system goal modelling described in the methodology, the FAS is illustrated using a hierarchy of system goals as shown in Figure 5.1.

Following the agent development method, the next stage is the agent use case stage, which consists of the following model: Filtering context diagram, event use case and Filtering agent.

*Capture Scenario*

*The second step* in the analysis of the agent system is to produce a context diagram of the FAS, which is captured based on events. Using context diagram modelling as proposed in the method, the Filtering agent problem domain is illustrated as in Figure 5.2.

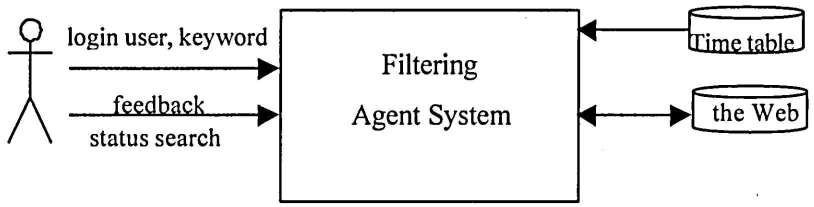


Figure 5.2: The Filtering Agents System Context Diagram.

The dynamic external environment of the system is articles in the Web(R1) and other environmental factors as resource are the user profile(R2), timetable(R7), the user new articles(R7), the user indexed document(R3) and userlogin database(R2).

| **Scenario 1** | | |
|---|---|---|
| Event | User Enquiry | |
| Source | User | |
| Activity | | |
| 1. | user makes enquiry(R1)<br>(userlogin(R2), password(R2), keyword(R1), status search(R7) | Role |
| 2. | Validate the user (R2), located not same as the enquiry,<br>(Userid, password)<br>UserloginDB | Action |
| 3. | Create userProfile for new user (R2) using appropriate default user profile<br>(Userid, profile features)<br>UserprofileDB | Task |
| 4. | Always display the new articles(R4) if any<br>userid (article's title, address, content), ranked value | Task |
| 5. | Set search timetable (R7)<br>(userid, keyword, date, time)<br>SearchtimetableDB | Role |
| 6. | Search from web (R1)<br>(userid, keyword)<br>The Web | Role |
| 7. | Download articles (R3), the 100 highest ranking articles<br>userid,(article's title, address, content) | Action |
| 8. | Indexed articles (R3)<br>userid, indexed document | Role |
| 9. | Capture new articles by comparing with previous indexed document(R8)<br>UserindexedDocDB | Task |
| 10. | Capture the latest user profile (R1)<br>UserprofileDB | Task |
| 11. | Filter indexed document using user profile (R1)<br>(indexed document, userprofile) | Role |
| 12. | Display ranked articles(R4), if user online (R2) or email<br>userid (article's title, address, content), ranked value | Action |
| 13. | Store the ranked articles (R7)<br>NewarticlesDB | Action |

Table 5.1: Event User Enquiry

| | | |
|---|---|---|
| **Scenario 11** | | |
| Event | User Feedback | |
| Source | User | |
| Activity | | |
| 1 | Choose articles and provide feedback on ranked values (R6), used the display article<br>according to user interested<br>userid (article's title, address, content, feedback value) | Role |
| 2. | Update userProfile (R5)<br>(Userid, profile features)<br>UserprofileDB | |
| 3. | Update new articles for articles not valued by the user<br>NewsarticlesDB | |

Table 5.2: Event User Feedback

| | | |
|---|---|---|
| **Scenario 111** | | |
| Event | Timetable | |
| Source | Computer time | |
| Activity | | |
| 1. | Capture the date to search<br>(userid, keyword, date, time)<br>TimetableDB | Role |
| 2. | Search from web (R1)<br>(userid, keyword)<br>The Web | |
| 3. | Download articles (R3), the 100 highest ranking articles<br>userid,(article's title, address, content) | Role |
| 4. | Index articles if found(R3)<br>userid, indexed document | Role |
| 5. | Compare with previous indexed document(R8)<br>UserindexedDocDB | |
| 6. | Capture user profile (R1)<br>UserprofileDB | |
| 7. | Filter indexed document using user profile (R1)<br>(indexed document, userprofile) | Role |
| 8. | Display ranked articles(R4) if user online (R2)<br>userid (article's title, address, content), ranked value | |
| 9. | Store the ranked articles (R7)<br>NewarticlesDB | |

Table 5.3: Event Time Scheduler

The analysis of the user requirement is able to identify three events: enquiry from the user (R1), user feedback (R2) and search timetable(R7) and match with the schedule searching time. Based on each event, the event scenario ṡ developed as shown below. The scenario modelling proposed in previous chapter is followed. Scenario 1, 11 and 111 are shown in Table 5.1, 5.2 and 5.3 respectively. Each activity is assigned the reference number of the relevant user requirement.

*Use Cases of the FAS*

The third step is to develop the event use cases based on each event scenario. The first use case is developed based on Scenariol as shown in Figure 5.3. Each activity in the scenario is transferred into the use case. All activities in Table 5.1 are transferred into the use case as S1A1...S1A12. A similar step is performed for the other two scenarios, shown in Figure 5.3, 5.4 and 5.5.

The Event Feedback use case consists of S2A1 to S2A3 and the Event Timetable use case consists of S3A1 to S3A9.

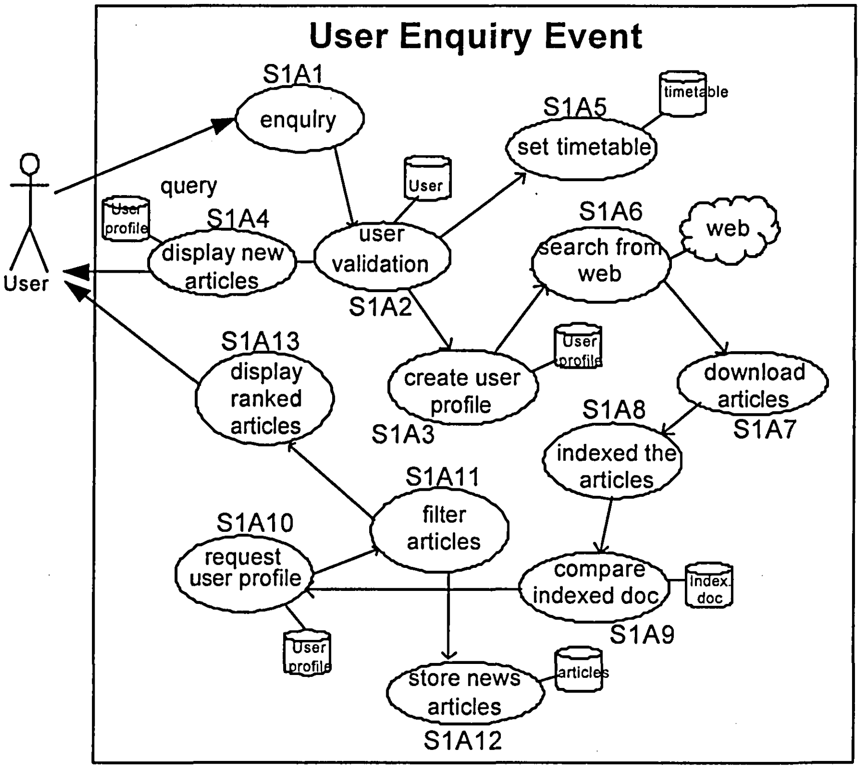


Figure 5.3: Event User Enquiry

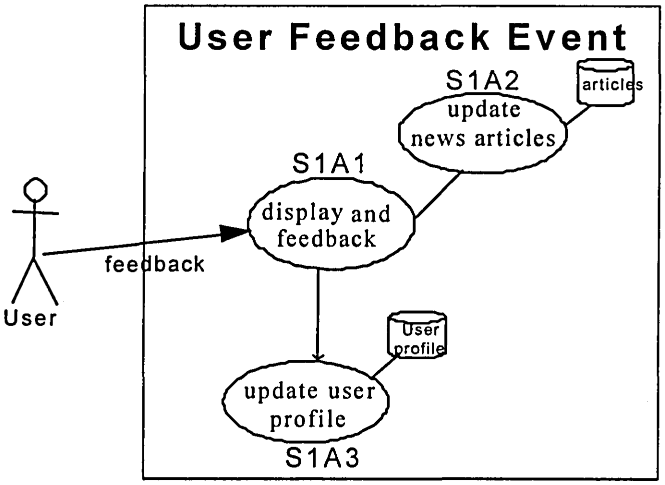


Figure 5.4: Event User Feedback

*The fourth step* is to develop the FAS use case. The agent use case is the combination of all event use cases, using the equation provided.

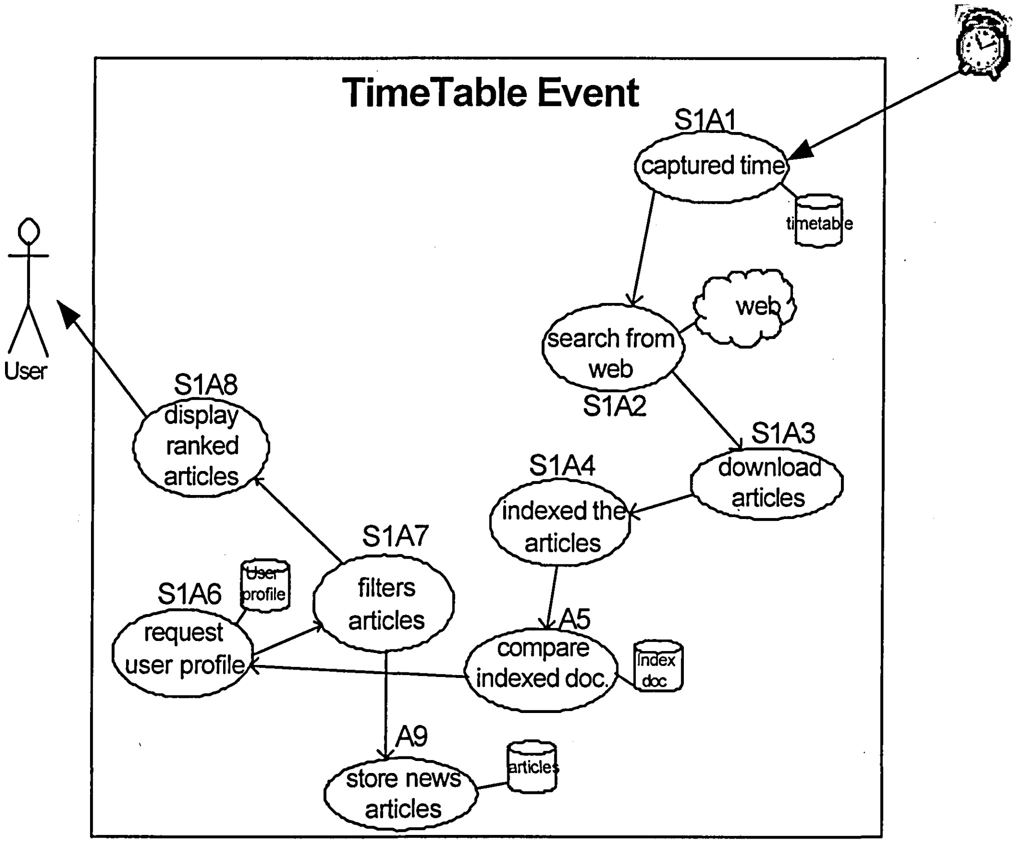


Figure 5.5 : Event TimeTable

There are seven use cases found are similar in the three events use cases: S1A6=S3A2, S1A7=S3A3, S1A8=S3A4, S1A9=S3A5, S1A10=S3A6, S1A13=S3A8 and S1A4=S3A8=S2A1. The first six use cases are the use cases from scenario 1 and scenario 111 and the last use case consists of all the three use cases. This means, the pits value of the first six use cases is two pits, while the value of the last use case is three pits.

The Filtering agent use case is developed as the result of all link found in those the use cases as shown in Figure 5.6.

The unique roles are captured by combining the similar roles into the unique role of use case. As the result the agent use case consists of the following elements: {S1A1, S1A2, S1A3, S1A5, S1A6, S1A7, S1A8, S1A9, S1A10, S1A11, S1A12, S1A13, S3A1, S2A3} and the pits values are [1,1,1,1,2,2,2,2,2,2,1,2,3,1} respectively. The pits value describes the rate of role are performed. The pits value of S1A1 is 1 because no similar use case is found, while the pits value of S3A1 is 3 for the role presented in the three scenario use cases.

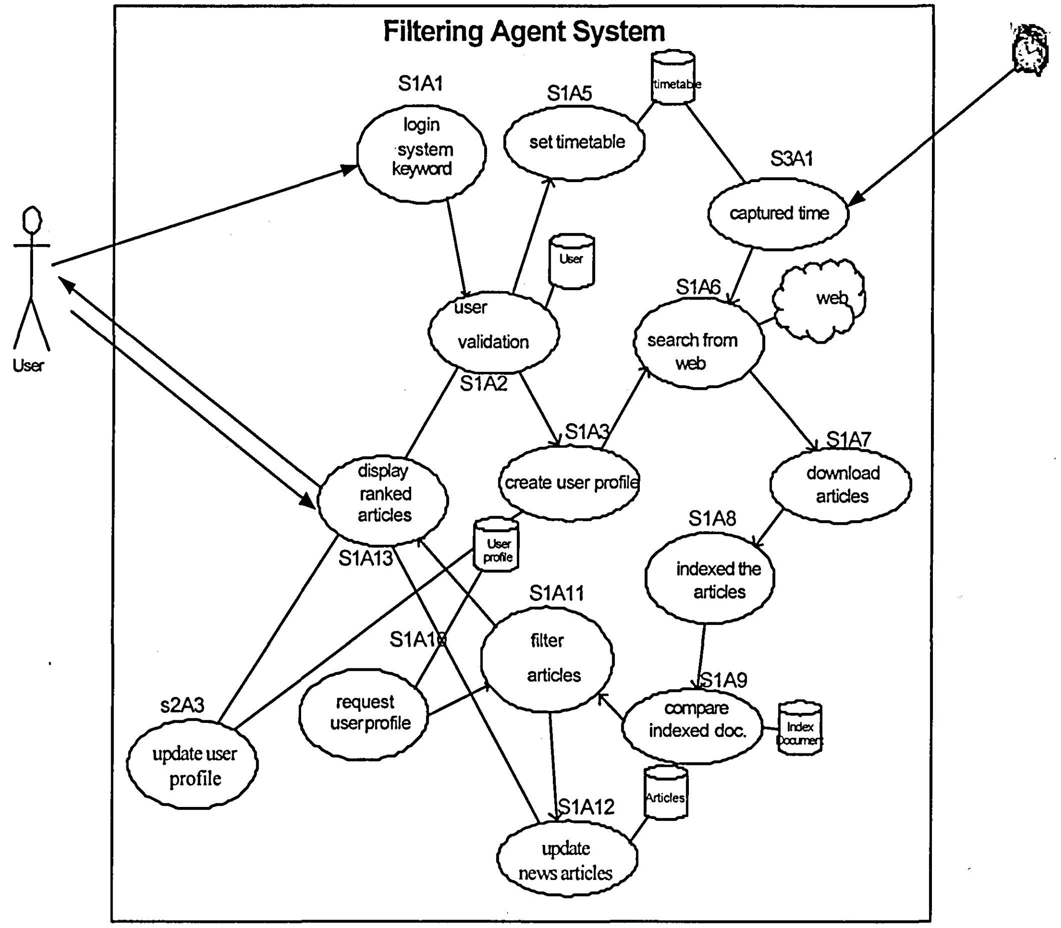


Figure 5.6: The Filtering Agent System Use Case.

*Identify Agent Paradigm*

The *fourth step* is to identify the agent paradigm. The characteristics of agent behaviour are identified:

- Adaptive behaviour- from goal no 2, the user profile of the system is constantly changed according to the user feedback.
- Learning behaviour - presented as requirement No. 5, whereby the filtering process is based on the user profile and the user profile is updated based on the user preferences, by applying genetic algorithm learning techniques.
- Pro-active behaviour, captured from goal No.3 and requirement No. 7, whereby the system keeps searching on behalf the user to find new articles.

The user requirement includes the criterion of distribution. The distributed location criterion is presented in requirement No. 2 that the user can access the system from anywhere, the search engine may be located anywhere and filtering functionality can be reused for other systems.

The openness criterion is important in that it allows the agent to notify the user of new articles found through email. The agent environment should provide this openness and the agent roles or services are open to use by any other agents from different environments. In this condition, the criteria of openness become a necessity for the agent environment of the FAS.

The analysis of the user requirements reveals that the FAS is not tied to any design perspective such as conceptual design of organisation or decision-making but rather based on distribution and complex task perspective ( reduce the complexity of system using agent).

A justification was made that the agent paradigm (abstraction for the solution of FAS is based on autonomous and reactive behaviour. So, the design is not focused on providing intelligence in all agents but presented as a result of interactions of the whole system such as the adaptive behaviour and pro-active behaviour. The criterion of intelligence such as learning behaviour is assigned only to the UserModelAgent.

Another method of identifying agent abstraction properties is examining the existing interaction protocol, but the analysis of the agent use case shows that the FAS only uses the FIPA-request and FIPA-informed.

The next stage of design is to identify agents, based on this agent abstraction and agent characteristics.

#### 5.1.2.2. Agent System Level Design

The *fifth step* is to produce the architecture of the FAS. The design process starts at this step. The previous steps focused on analysis modelling, while the design stage begins with identifying agents and their interactions as an iterative process, which are represented in specific types of agent models: agent system architecture and system plan. The system plan model is illustrated in terms of two types of plan model: plan sequence diagram and plan activity diagram.

*Identify Agent*

The first step in agent system design is identifying the agent. As stated in the methodology, there are four factors that influence identification of agents: behaviours, distribution, communication and openness. Identification is a recursive process, including justifying and assessment of agents in terms of the performance using the pit value. The intersection result of all use cases above results in the agent use case as shown in Figure 5.7 below. The colour line area shows the roles assigned to agents. Six agents are identified: UserAgent(user), DocumentAgent(document), FilteringAgent(process), ScheduleAgent (timetable), SearchAgent(the web) and UserModelAgent (user profile and new articles). The analysis of pits value does not effect the performance because the concurrent pit value is small.

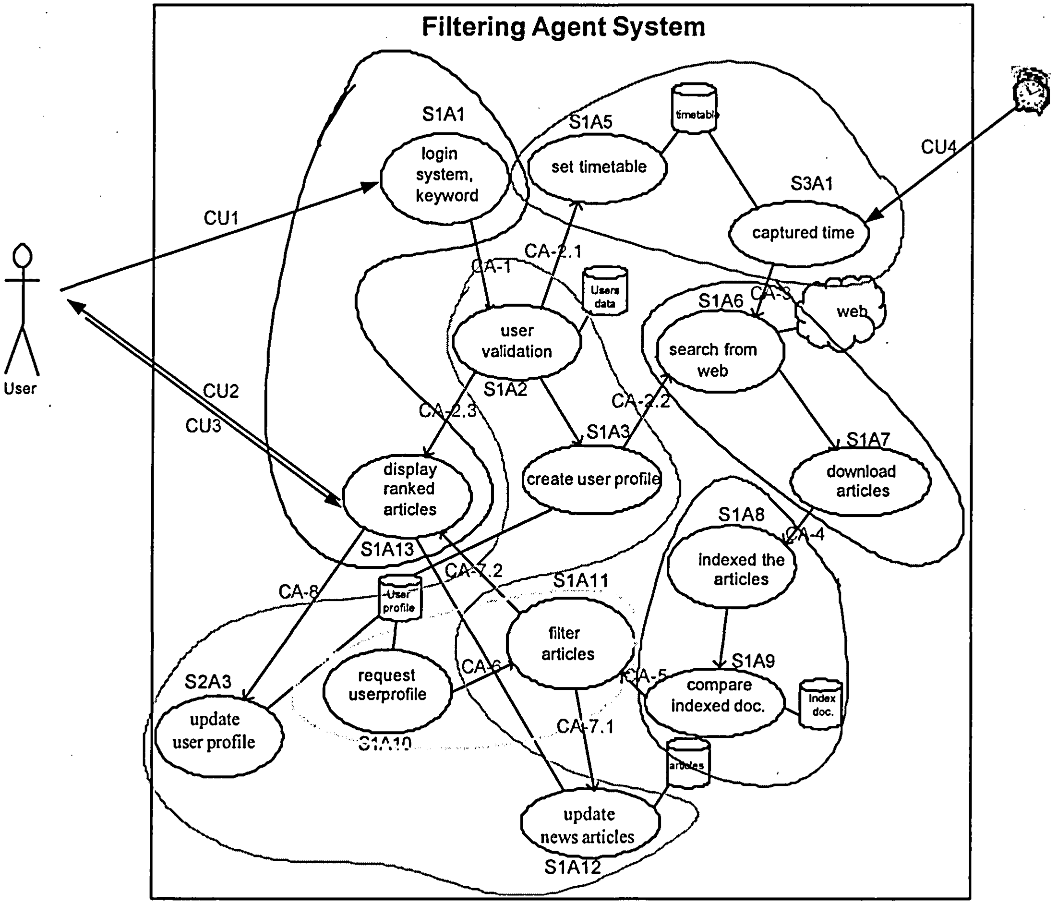


Figure 5.7: Assigning Roles to Agent

The use cases S1A1 and S1A13 present a similar type of role, as user interface. Therefore these two use cases are assigned to an agent named UserAgent. The S1A5 and S3A1 use cases are roles related to the timetable; these use cases are assigned to ScheduleAgent, which is responsible for arranging the timetable and indicating when the searching process should be performed. S1A6 and S1A7 are assigned to an agent named SearchAgent, the role of which is to download articles from the web according the keyword. S1A8 and S1A9 are related to indexing articles: therefore we combine them as an agent named DocumentAgent. The task of S1A11 is to filter the articles, but it needs the user profile for the filtering process, therefore S1A10 is combined with S1A11 as FilterAgent. The use cases S1A3, S2A3, S1A12, S1A10 and S1A2 are combined as an agent named UserModelAgent. This is because most of the use cases are related to manipulation of the user profile. Although S1A2 is not directly related to the user profile, it captures the userid and password, which are entities of the user profile database. The S1A10 use case is needed in FilterAgent but it is captured as a UserModelAgent responsibility, so that the use case is captured in the UserModelAgent and in the FilterAgent.

| |
|---|
| *Agent name*: UserAgent |
| *Goal:* most relevant articles |
| *Roles* : Query<br>:Feedback |
| *Query:* Send query to other agents<br>*Feedback*: Update user profile and newarticles |

| |
|---|
| *Agentname: ScheduleAgent* |
| *Goal: choose right time to search* |
| *Roles: Scheduletime*<br>*:Captured time* |
| *Assign timetable monthly*<br>*Capture clock time based on timetable.* |

| |
|---|
| *Agent name*: SearchAgent |
| *Goal:* Downloads first 100 of relevant articles |
| *Roles* : Searcher |
| Use several types of search engines |

| |
|---|
| *Agent name:*DocumentAgent |
| *Goal:* Accurate and fast Indexing |
| *Roles*: Indexer |
| Choose good index technique |

| |
|---|
| *Agent name*: FilterAgent |
| *Goal:* Accurate filtering |
| *Roles* :Filter |
| Use userprofile to filter the articles |

| |
|---|
| *Agentname: UserModelAgent* |
| *Goal: Provides services* |
| *Roles:* UserEnquiry<br>:UserFeedback<br>:FilterAgentServices |
| Use learning technique of genetic algorithm to maintain the user profile |

Table 5.4: Agent Abstraction

At this stage, it should be possible to identify each agent's goal and roles. The agent identification makes it possible to identify the agent interaction. The description of the agent goal, roles and brief of tasks as shown using agent abstraction to present agents is shown in Table 5.4. These agent abstractions are developed in detail at the agent level design stage. By using our agent framework, we can identify the relationship between agent, roles and tasks.

*Identify agent Interaction*

The second step of agent system design is to identify agent interaction. Agent interactions are identified using the similar agent use case. The agent identification is able to identify possible interaction between agents, interaction with the external environment or object agents and internal interaction. The agent identification shown in Figure 5.7 is also used to identify agent interactions. The agent use case diagram is able to identify the internal agent interaction and inter-agent interactions. The arrows crossing the colour boundary are identified as inter-agent interactions. Each arrow that crosses the colour boundary is named as shown in Figure 5.7. As a result, the FAS architecture is developed as shown in Figure 5.8. The architecture represents the agents and their interaction.

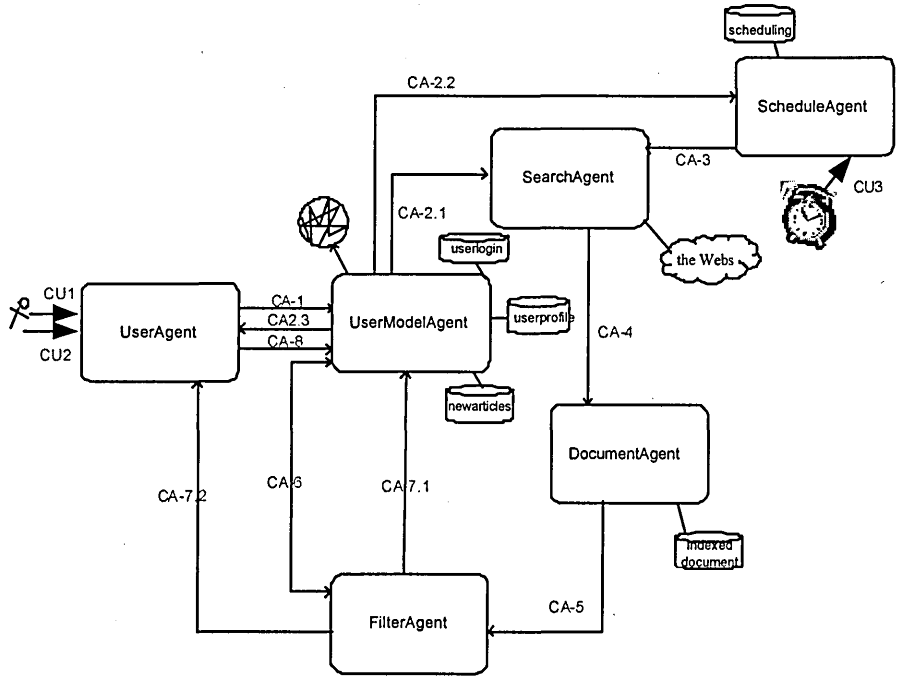


Figure 5.8: Agent System Architecture.

The User agent provides a user interface for the user to login into the system, provides keywords, and displays ranked articles to the user for feedback. The UserModelAgent is responsible to validate the user, create new user profiles, update user profiles, update new articles and inform ScheduleAgent to schedule the search timetable. The schedule time is assigned based on the username and keyword. ScheduleAgent maintains a schedule indicating when search the articles again. The scheduling time is assigned based on the user preferable time but again the ScheduleAgent has to act according to availability. SearchAgent is responsible to search articles from the web. The result of downloading articles is sent to DocumentAgent. DocumentAgent is responsible for indexing the articles and checking with the existing download articles to compare whether any new articles have been found. FilterAgent is responsible for ranking indexed articles according to the user profile.

The result of the ranked articles is send to UserAgent for display to the user. If UserAgent's status is waiting, then the result is displayed to the user; otherwise the result is stored in a new articles file by the UserModelAgent. Any new articles not ranked by the user are stored in the news articles files.

*Agent Interactions between Agents*

In the architecture shown above, the interactions identify what type of negotiation or communication patterns is used between two agents. The identification of interaction can be carried out at an earlier stage while analysing the requirements, but we did not find any obvious pattern of communication. Each agent interaction is discussed below:

CA-1 UserAgent sends information of the user login, password, keyword and status search, then in other condition the user provides feedback value for the new articles found using FIPA-request, FIPA-inform.

CA-2.1 UserModelAgent informs SearchAgent to do query based on keywords provided-FIPA-inform.

CA-2.2 UserModelAgent inform the ScheduleAgent to set schedule searching timetable-FIPA-inform.

CA-2.3 UserModelAgent sends User Agent the new articles found- FIPA-inform..

CA-3 ScheduleAgent instructs SearchAgent to search the articles- FIPA-inform.

CA-4 SearchAgent passes on the articles result to the DocumentAgent - FIPA-inform.

CA-5 DocumentAgent sends all articles, with weighted values, to FilterAgent- FIPA-inform.

CA-6 FilterAgent requests UserModelAgent for user profile informatio n- FIPA-request.

CA-7.1 FilterAgent sends new articles to NewArticlesAgent if the user status is not active-FIPA-inform.

CA-7.2 FilterAgent sends ranked articles to UserAgent- FIPA-inform.

CA-8 UserAgent sends feedback on articles values to the user model- FIPA-inform.

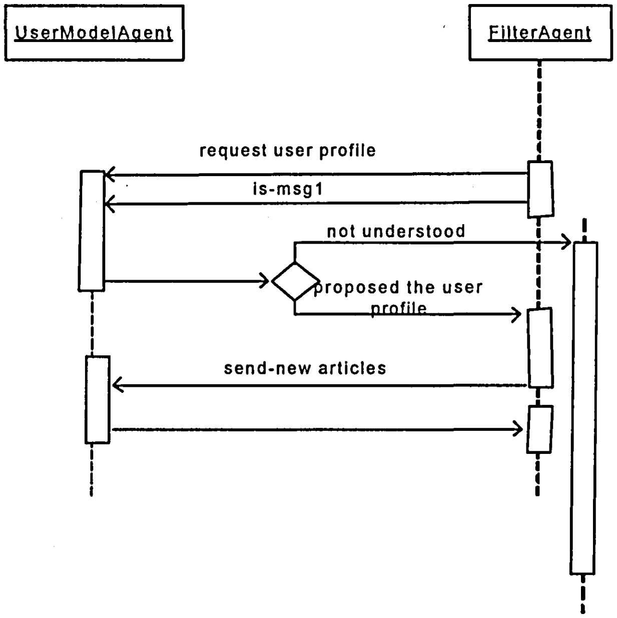


Figure 5.9: Agent Interactions

In most cases, the interaction between the agents is simple (inform or request). The interaction between UserModelAgent and FilterAgent is somewhat more complex. Therefore, we model the UserModelAgent and FilterAgent of agent interaction as shown in Figure 5.9.

The analysis of agent system architecture shows that most agents use inform and request interaction protocol for interaction. Therefore, other interactions model need not be documented.

*System Plan Model*

The *third step* of agent system design is to identify the plan of the agent system. It is modelled using activity diagrams and sequence diagrams.

According to the plan design modelling process, two activity diagrams are developed. Each event source is represented as one activity diagram. Each use case in the agent use case is transformed into a process in the activity diagram. Figure 5.10 shows the activities of the whole FAS, which represent the adaptive agent behaviour. Figure 5.11 shows the activity diagram of the FAS when the clock time hits the schedule search timetable.

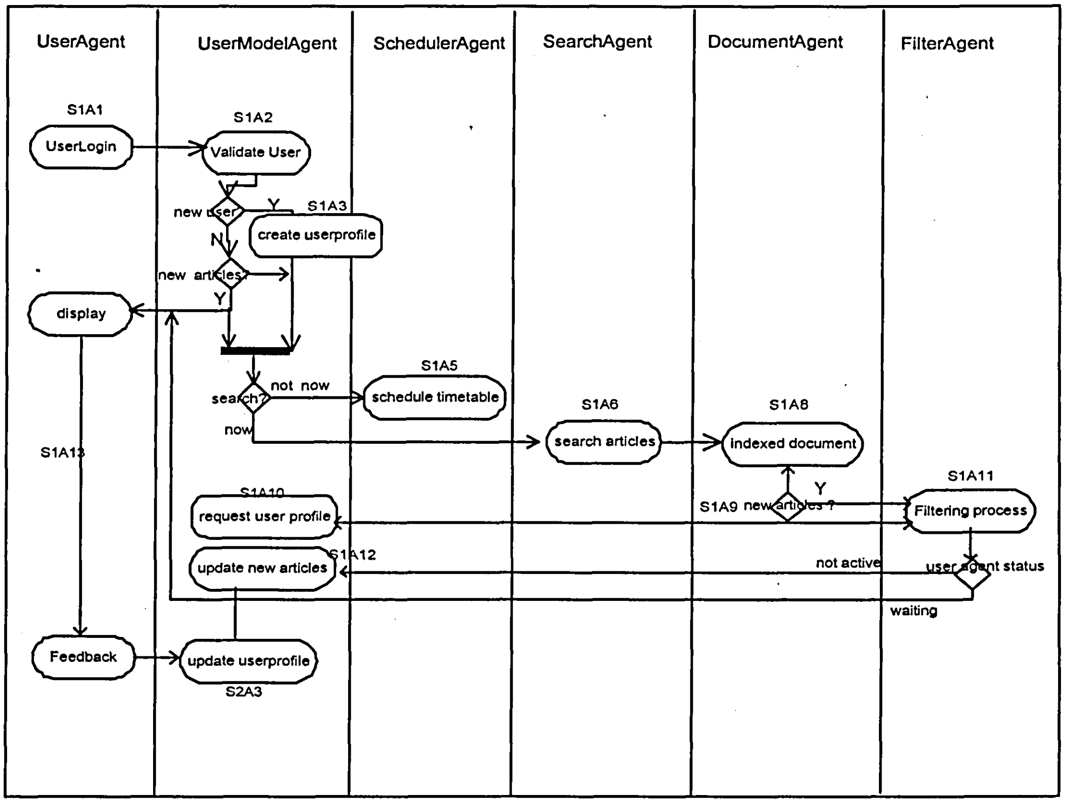


Figure 5.10: The Filtering Agent System Activity Diagram (the user initiator)

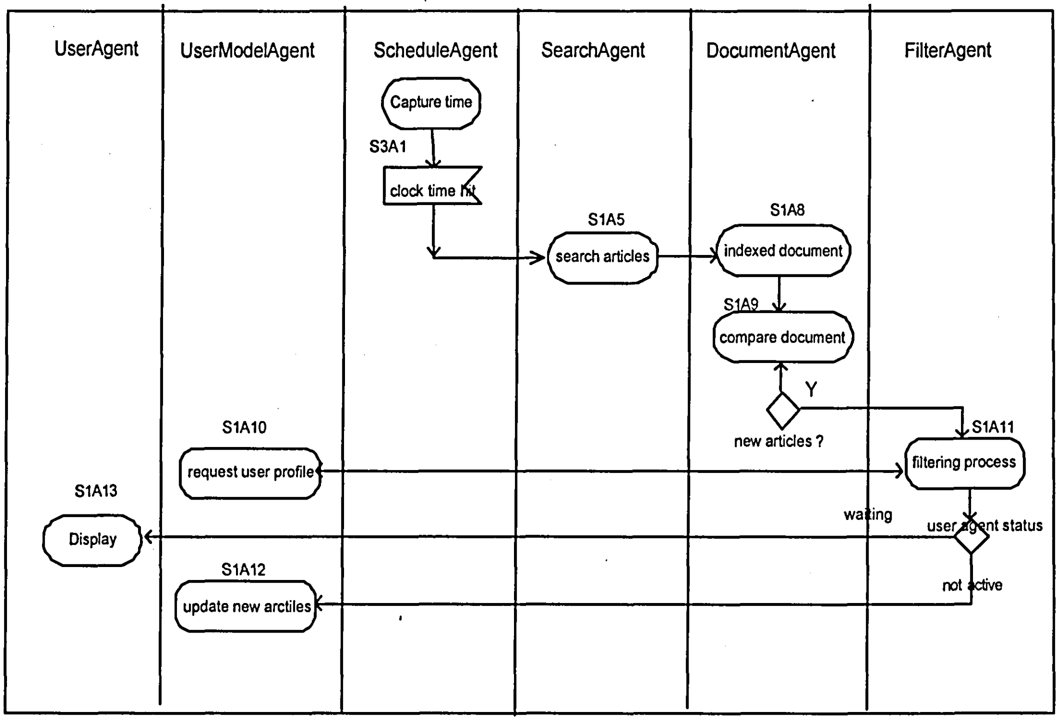


Figure 5.11: Activity Diagram from Clock Event (scheduler initatior)

Follow the plan modelling as discussed in the methodology, the global plan for the FAS is developed as shown in Figure 5.12 below. It shows that three events may occur: User login and request searching, user feedback and timetable clock. The three events are shown by the large-headed arrows. The global plan of the FAS interaction shows a pattern of an adaptive and proactive FAS.

When the clock hits the scheduled time, the following agents are involved in collaboration: ScheduleAgent, SearchAgent, DocumentAgent, FilterAgent and NewsArticles, UserModelAgent similar to when the user makes enquiry. However, the feedback role event only involves the UserModelAgent and UserAgent.

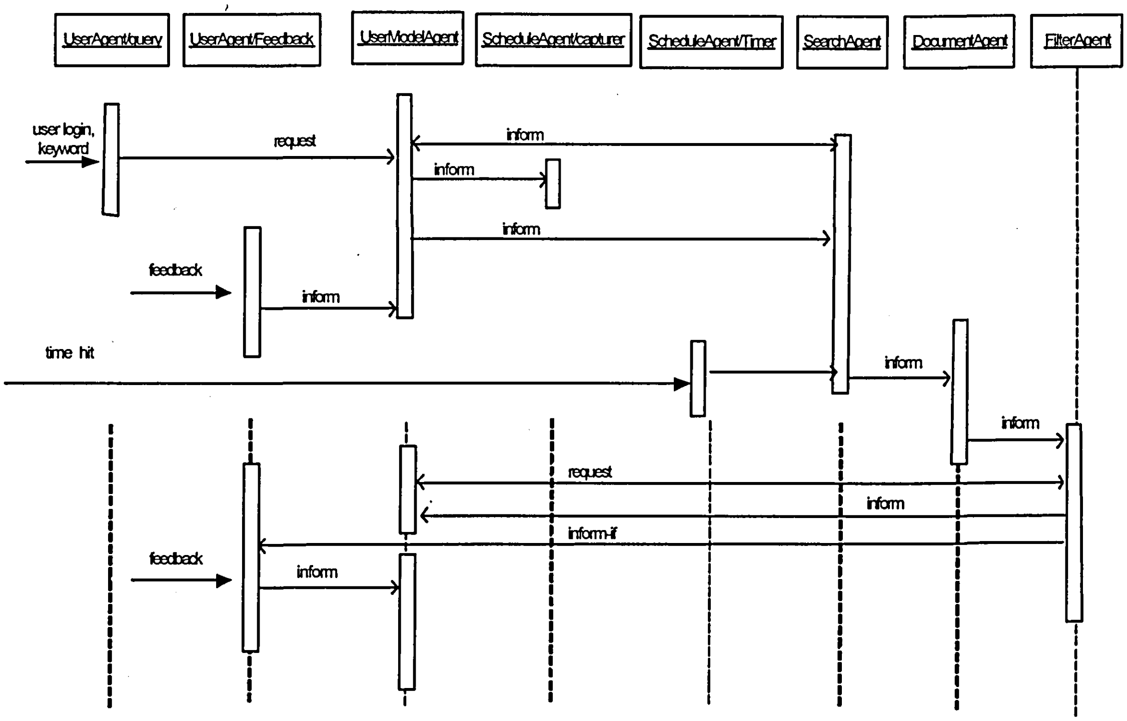


Figure 5.12: The Filtering Agent System Plan

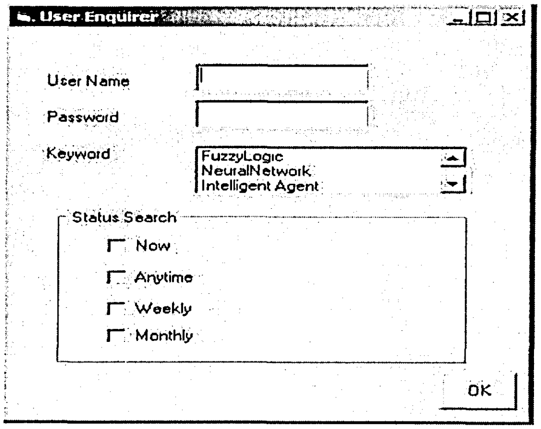
The system plan shows the agent interactions through the agent roles. At this stage the design is able to show the agent interaction of the whole agent system, while the next stage is focused on the design of each agent.

### 5.1.2.3. Individual Agent Design.

The *fourth step* is to design each agent. This stage of the design process is related to section 4.2.3 in the previous chapter. There are three agent design models to present agent level design: *agent model, agent state model* and *agent class diagram.* The agent class diagram is a design model produced after the development of the agent environment, when all the components used to develop agents have been developed. The above discussion presented the first two design models of each agent. The detailed tasks of each agent such as index function, genetic algorithm formula, etc. are not provided in the requirement. Here, we present the detailed process of the task, as we know the solution for each task as it implemented it in the system. At this stage, additional internal tasks may be required. Therefore, at this stage of modelling, a new task may be introduced which was not presented in previous modelling.

#### a. UserAgent

A UserAgent is responsible for providing a user interface for the FAS. It provides a user login interface that allows the user to log into the system, enter keywords to search articles and decide the status of searching, whether immediately, weekly or monthly. The second interface is to display the articles found, with ranking values according the user's interest. Based on the result, the user is allowed to update the user model by choosing the most interesting articles. The design of each user interface is shown in the Figure 6.13 and 6.14 respectively.


User Enquirer

User Name

Password

Keyword: FuzzyLogic<br>NeuralNetwork<br>Intelligent Agent

Status Search

□ Now

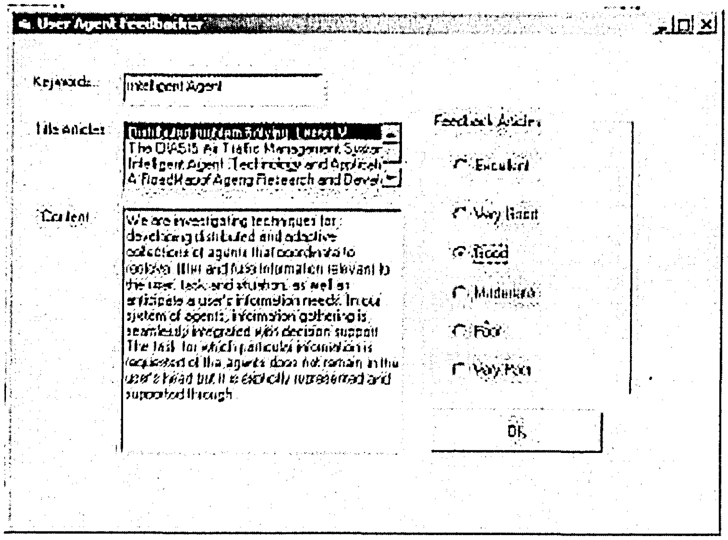
□ Anytime

□ Weekly

□ Monthly

OK

Figure 5.13 Enquirer User Interface


User Agent Feedbacker

Keywords: Intelligent Agent

Title Articles:
[illegible]
The OASIS Air Traffic Management Syster
Intelligent Agent: Technology and Applicati
A RoadMap of Agent Research and Devel

Content: We are investigating techniques for developing distributed and adaptive collections of agents that coordinate to retrieve, filter and fuse information relevant to the user, task and situation, as well as anticipate a user's information needs. In our system of agents, information gathering is seamlessly integrated with decision support. The task for which particular information is requested of the agents does not remain in the user's head but it is explicitly represented and supported through

Feedback Articles

○ Excellent

○ Very Good

● Good

○ Moderate

○ Poor

○ Very Poor

OK

Figure 5.14: Feedbacker User Interface

The agent model provides the description of the agent name, goal, roles and tasks: internal tasks and communication tasks, as described in our agent framework design. It includes the data description of both tasks.

UserAgent Model

| | | | | |
|---|---|---|---|---|
| **UserAgent** | | | | |
| **Goal** | Provides interface for user interact to the system | | | |
| **Role** | Enquirer | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **Content** |
| 1. User interface | User | CU-1(i) | | |
| 2.Login system and perform query | UserModelAgent | CA-1(O) | *request and inform* | *userid, password, keyword, status search* |
| 3. Display result | UserModelAgent | CA2.3(i) | *inform* | *rank document result (userid, articles, ranked value)* |
| | | | | |
| **Role** | Feedback | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| 1. Display result | Filter Agent | CA-7.2(i) | *inform* | *ranks document result (userid, articles, ranked value)* |
| 2.User Feedback | the User | CU-2(i) | | |
| 3. Feedback | User-Model Agent | CA-8(O) | *inform* | *Update user profile by using score value. (userid, articles, feedback value)* |

Figure 5.15: User Agent Model

The agent modelling data structure is a refinement of the FAS plan and activity diagram. It identifies possible event occurs, internal tasks and action taken, that may involve interaction with other agent. The UserAgent consists of two roles, first as enquirer and second as feedbacker. The user interface does not provide any internal decision, but is only an interface for the system. The user interface captures events coming from the user and initiates communication with other agents. The UserAgent data model is shown in Figure 5.15.

*UserAgent Plan Model*

The internal design agent shows the flow on receiving an event from other agents or from the external environment. The internal design describes the internal plan for achieving the agent goal. The agent plan model manages the flows of events received, associated with the action taken.

The internal agent design is developed based on the UserAgent data model and plan activity diagram in Figure 5.10 and 5.11. The UserAgent state model shows the flow of internal agent design, and also the agent decision process. The UserAgent state model is shown in Figure 5.16.

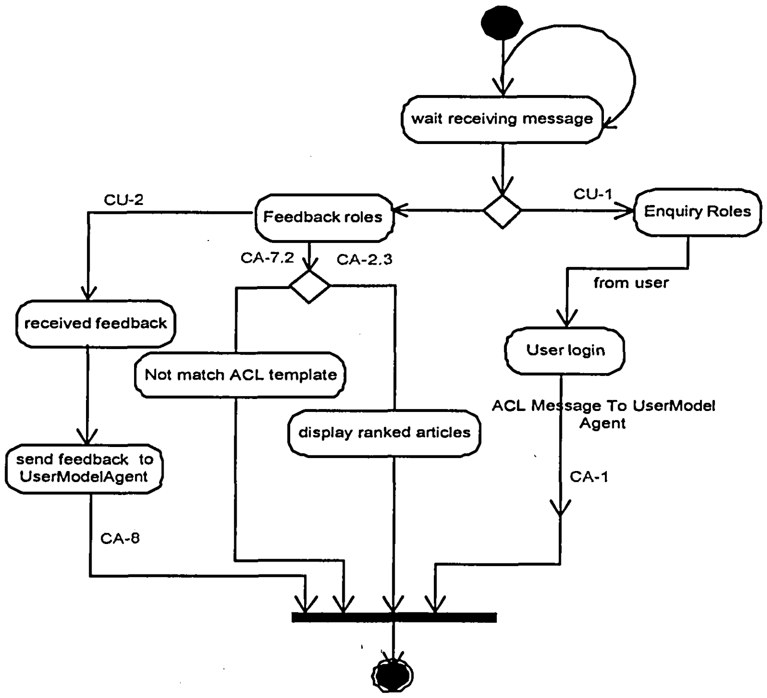


Figure 5.16: The UserAgent State Model Diagram

### b. UserModelAgent

UserModelAgent is the core of the system's adaptability. The UserModelAgent is responsible for:

- Validating the user. The UserModelAgent has knowledge of the userid and password, so when an event occurs from the user, it is able to react instantly to identify as a new user or existing user. If it is a new user, the UserModelAgent creates a user profile using the default genetic algorithm user properties references.
- Informing the ScheduleAgent to schedule a suitable time to search. The 'search status' provided by the user informs the UserModelAgent whether the user is waiting for the result (the agent's status search is now refer). The choice of any other 'search status ' means that the user wants the agent to search for articles at anytime, weekly or monthly.
- Sending new articles found to the UserAgent, every time the user login into the system and send email for the notification.

The user model uses the genetic algorithm (GA) learning techniques when creating the new user profile and updating the user profile. Thus, a user provides a linguistic value of feedback for every retrieval, which is then used to adjust the fitness of the chromosomes in the population. The amount by which the fitness of chromosomes is adjusted is determined from a fuzzy inference mechanism that relates the user relevance feedback and the similarity of the query and document vectors. The remit of this agent is to guide the user in the query formulation process and to store and manage the user's interest in the form of a user profile.

The analysis of the system plan and activity diagram are able to identify possible events occurs in the UserModelAgent. Four events can be identified. Each event is identified in terms of agent roles. For each role associated tasks are indicated.

Create initial user profile.

One aspect of the functionality of the UserModelAgent is the capability to build an initial user profile which stores a set of weighted keywords. In order to do that, the system exhibits a collaborative behaviour, whereby the user is enabled to build his/her profiles by sharing other profile in a similar search area.

Update User profile.

The main functionality of this agent is to adjust the representation of the user interest so it is consistent with the latest user feedback.

| **UserModel Agent** | | | | |
|---|---|---|---|---|
| **Goal** | Maintain the user model properties | | | |
| **Role** | UserEnquirer | | | |
| **Tasks** | **Collaboration** | **Communication** | | |
| | | **Name** | **Message type** | **Content** |
| 1. Validate the user | UserAgent | CA-1(i) | *Inform* | *User login name, or new user* |
| 2.Create initial user model (Apply genetic algorithm) | | | | |
| 3. Make enquiry to search | SearchAgent | CA-2.1(O) | *inform* | *userid, keyword* |
| 4. Acknowledge searching | ScheduleAgent | CA-2.2(O) | *inform* | *userid, keyword, status search* |
| 5. Display new articles | UserAgent | CA-2.3(O) | *inform* | *ranked document result (userid, articles, ranked value)* |
| **Role** | UserFeedaback | | | |
| 1. Receive Feedback | UserAgent | CA-8(i) | *inform* | *Update user profile by using score value* |
| 2. Update user model (apply GA) | | | | |
| 3. Update new articles | | | | |
| **Role** | FilterAgentservices | | | |
| 1. Receive query | FilterAgent | CA-6(i) | *Request* | *find fittest chromosome* |
| 2 .Finding fittest chromosome | | | | |
| 3. Send user model | FilterAgent | CA-6(O) | *Inform* | *Best chromosome vector* |
| 4. Receive Newsarticles | FilterAgent | CA-7.1(i) | *Inform* | *Vector of ranked new articles* |
| 6. Update new arctiles | | | | |

Figure 5.17: UserModel Agent Model

The data structure of the UserModelAgent agent model and the internal design is shown in Figure 5.17.

*Use ModelAgent- Plan Model*

Follow the agent method, the plan of the UserModelAgent is developed according to Figure 5.10 and 5.11. The UserModelAgent plan model is produced as shows in Figure 5.18. The following diagram shows the three internal UserModelAgent tasks.

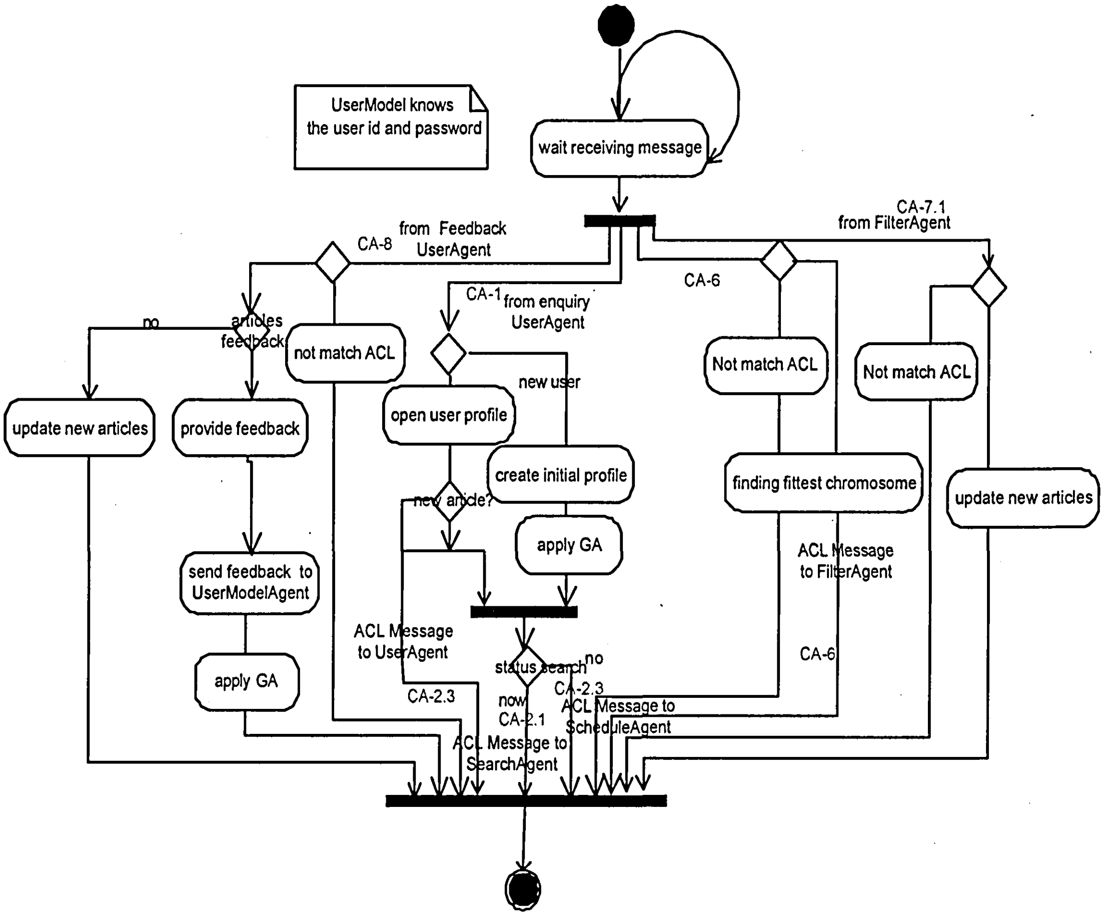


Figure 5.18: Internal State Diagram of UserModelAgent

The tasks use genetic algorithms to present the adaptive behaviour (the system should notice when the user changes his/her interests) and exploratory behaviour (the system should be able to explore new information in a particular domain which could potentially be of interest to the user) while reference feedback is for intelligent fuzzy logic behaviour (the system must serve the specific interest of the users and build user profiles in order to recommend the most relevant articles.)

### c. SearchAgent

The SearchAgent is responsible to search articles from the web. It has two roles, triggered by events from the user UserModelAgent and events from SchedulerAgent. However, both roles are associated with similar tasks. They are combined as one role. The three tasks are: receive the query, perform search from the web and send the article result to the DocumentAgent. The SearchAgent model is shown in Figure 5.19, while the SearchAgent Plan model is shown in Figure 5.20.

*SearchAgent Model*

| **Search Agent** | | | | |
|---|---|---|---|---|
| **Goal** | Always performs search articles from webs | | | |
| **Role** | Searcher | | | |
| **Task** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **Content** |
| 1. Receive keywords | UserAgent | CA-2.1(i) | *inform* | *userid, keywords* |
| | ScheduleAgent | CA-3(i) | *inform* | *userid,keywords* |
| 2. Perform Search | | | | |
| 3. Send Articles | DocumentAgent | CA-4(O) | *Inform* | *search result* |

Figure 5.19: SearchAgent Model

*SearchAgent Plan Model*

SearchAgent keeps waiting for events coming from the UserAgent or ScheduleAgent. If the user specifies the 'search status ' is now, the SearchAgent receives the command to search. Alternatively, the ScheduleAgent informs the SearchAgent to search articles according to the timetable.

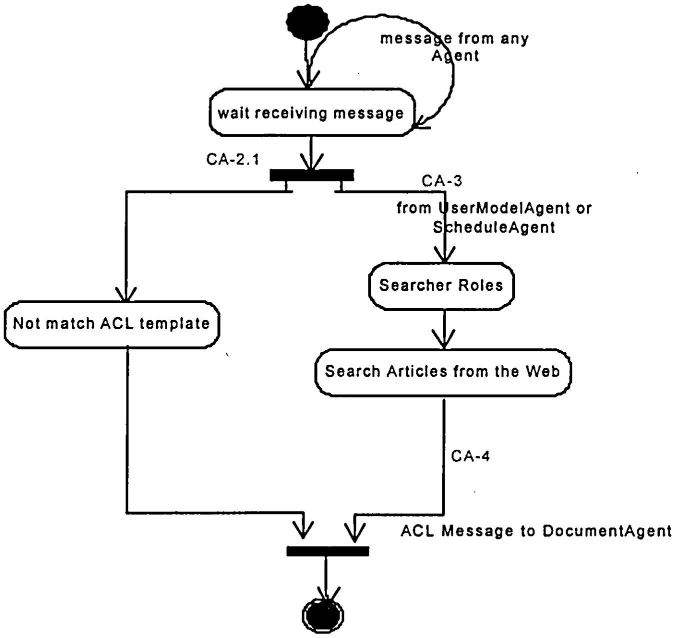


Figure 5.20: SearchAgent State Diagram

### d. DocumentAgent Model

The DocumentAgent aims to create a representation of each document. The internal DocumentAgent is captured from the analysis of Figure 5.10 and Figure 5.11. The Document agent has a similar task pattern to the Search Agent, which consists of three tasks, two of which can be identified: receiving a vector of articles, identifying new articles and sending a vector of indexed documents. The indexing process is the internal agent process, which translates the articles into index document representation.

Each index document is stored with the user ID and index document result. The indexed document is stored based on the userid and keyword. Every time it receives an article result from SearchAgent, the articles are indexed. The indexed document is compared with that from the previous search. If new articles are identified the index document is updated and the previous one deleted. If there is no previous index document, it is assumed that as new articles are found and they are sent to the FilterAgent.

The Document Agent model is shown in Figure 5.21 and the Document agent plan model show in Figure 5.22. The Figure 5.23 shows the internal process in indexing process task.

*Document Agent Model*

| | | | | |
|---|---|---|---|---|
| **Document Agent** | | | | |
| **Goal** | Create document index | | | |
| **Role** | Indexdocument | | | |
| **Task** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **Content** |
| 1. Received articles | Search Agent | CA-4(i) | *Inform* | *Vector of articles (title, content, address)* |
| 2. Indexing process | | | | |
| 3. Comparing with previous indexed document | | | | |
| 4. Send Document Index | FilterAgent | CA-3(O) | *Inform* | *Vector of indexed document* |

Figure 5.21 :Document Agent

Figure 5.22 shows that DocumentAgent knows the owner of the previous index document. So every time it receives articles from the SearchAgent, the document knows whether the articles represent a new search result. This is identified based on the userID and keywords. If it is a new search, the indexed document is sent to the FilterAgent, where it is compared with the user's previous indexed document. If there are new articles, the index document file is updated with the new one and the new articles are sent to the FilterAgent. The comparison process is based on article title, address and content.

*DocumentAgent Plan Model*

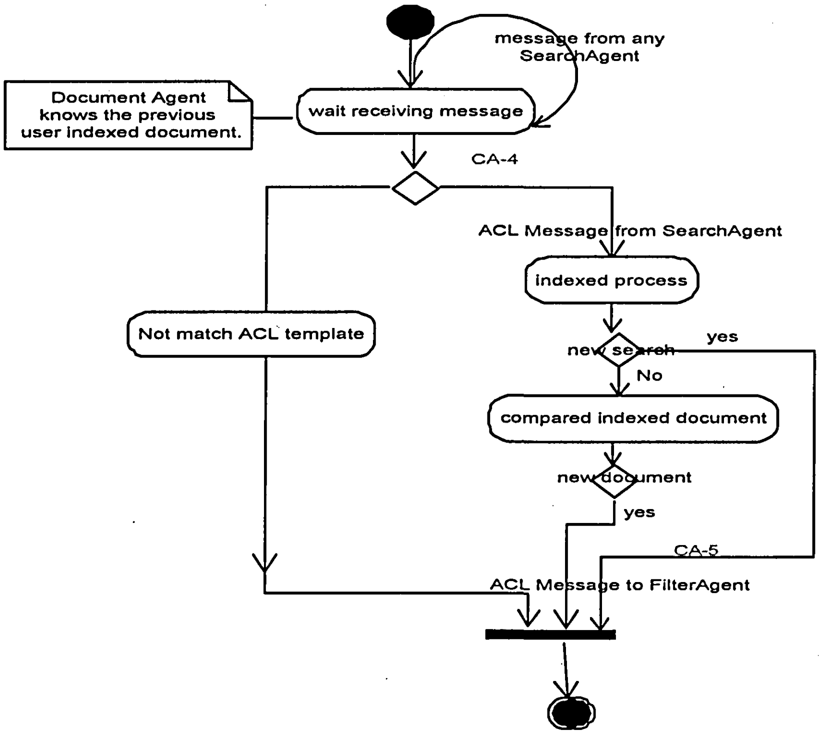

message from any SearchAgent
Document Agent knows the previous user indexed document.
wait receiving message
CA-4
ACL Message from SearchAgent
indexed process
Not match ACL template
new search
yes
No
compared indexed document
new document
yes
CA-5
ACL Message to FilterAgent

Figure 5.22:Document Agent State Diagram

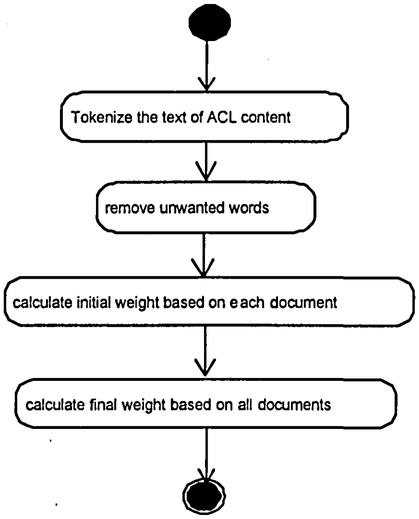

Tokenize the text of ACL content
remove unwanted words
calculate initial weight based on each document
calculate final weight based on all documents

Figure 5.23: Indexing Process

### e. FilterAgent Model.

The FilterAgent is responsible for ranking the articles using the index document. The articles are ranked based on the user profile, which indicates the user's interest preferences. Traditional search agents retrieve a set of potentially relevant documents from the Internet. This retrieval is efficient in terms of high recall rate, and fast response time, but at the cost of poor precision. Recall rate is the percentage of documents that are retrieved, while precision is the percentage of documents retrieved that are considered relevant. The Filter Agent increases the precision of ranking value by ranking the documents according to user preferences.

The FilterAgent is able to filter the articles based on user preferences, which should make them available to access, but the result may not be sent to the user if the user is not activated. In that case, the ranked articles are sent to the NewsArticlesAgent, whereas the articles are displayed to the UserAgent. When the user logs in, the new articles are sent to the user.

The FilterAgent only observes one type of event from the Document Agent. The main responsibility of Filter Agent is described in the Figure 5.24 and 5.25 showing the internal processing of the ranking process.

*FilterAgent Model*

| **Filter Agent** | | | | |
|---|---|---|---|---|
| **Goal** | Rank the result according indexing document upon the user model profile | | | |
| **Role** | Filtering | | | |
| **Tasks** | **Collaboration with agent** | **Communication** | | |
| | | **Name** | **Message Type** | **Content** |
| 1. Received index document | Document Agent | CA-5(i) | *Inform* | *vector of indexed document* |
| 2. Request the user profile | UserModel Agent | CA-6(O) | *request* | *userid* |
| 3. Result userprofile | UserModelAgent | CA-6(i) | *inform* | *best chromosome vector* |
| 4. Rank process | | | | |
| 5. Request the user status | UserAgent | CA-7.3(O) | *ask-if* | *user status: busy, dead, waiting* |
| 6. Update new articles file | UserModelAgent | CA-7.1(O) | *inform* | *rank document result* |
| 7. Send result | UserAgent | CA-7.2(O) | *inform* | *rank document result* |

Figure 5.24: FilterAgent Diagram

*FilterAgent Plan Model*

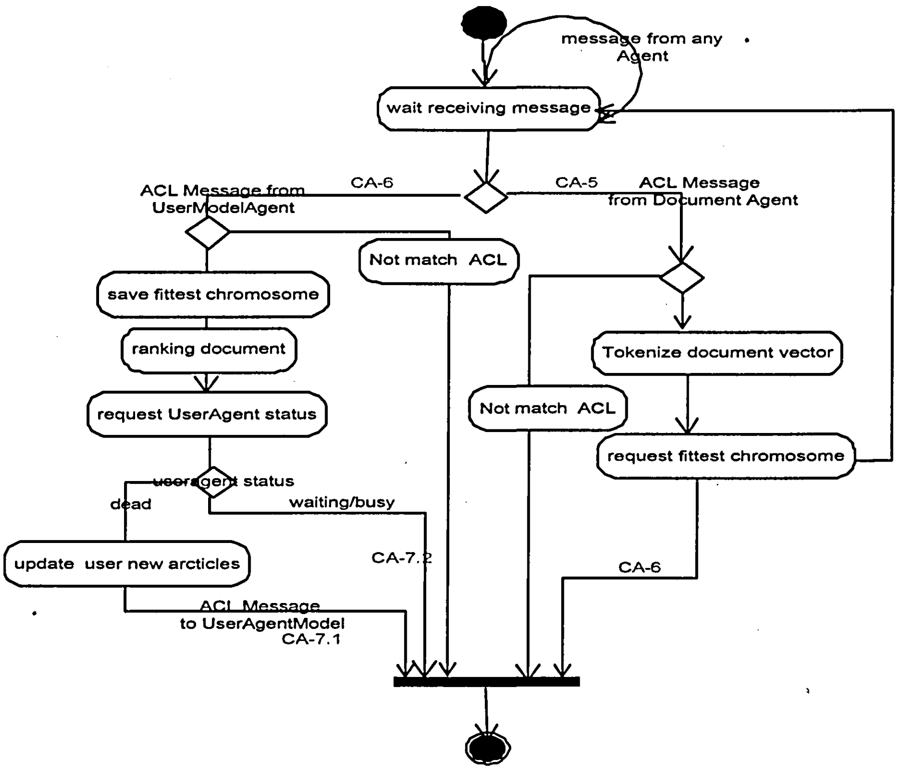


Figure 5.25: FilterAgent State Diagram

### f. ScheduleAgent Model

A similar process is applied to the design of internal ScheduleAgent. The ScheduleAgent receives two events. The first comes from the UserModelAgent. The event assigns a schedule timetable to search articles. The decision is based on the search status: anytime, weekly and monthly. The schedule timetable is maintained on a monthly basis. The second event is when the computer clock hits the schedule timetable. The ScheduleAgent informs the SearchAgent to search articles. The scheduler timetable stores the time to search with the relevant user ID and the keywords. The ScheduleAgent model and ScheduleAgent plan model are shown in Figure 5.26 and Figure 5.27 respectively.

The ScheduleAgent consists of two roles: In the first, it receives messages from the UserModelAgent. When the ScheduleAgent receives a message from UserModelAgent, the ScheduleAgent compares it with the schedule time table according to the date, user ID and keywords. If a schedule time is not set up, the ScheduleAgent chooses a random time to schedule, according to the balance of the free time available within a month.

The timetable is defined as a two dimensional vector. An example of the timetable is shown below.

| Date | Time | Keyword |
|---|---|---|
| 1/2/02 | 12:44 | IntelligentAgent |
| 1/2/02 | 12:50 | Fuzzy Logic |
| 2/2/02 | 14:20 | management information |
| 2/2/02 | 14:50 | recycling management |
| 2/2/02 | 3:30 | IntelligentAgent |

| UserId | Keyword |
|---|---|
| cmszo | IntelligentAgent |
| cmszo | Fuzzy Logic |
| cmsra | management information |
| cmsra | recycling management |
| cmssa | IntelligentAgent |

Table 5.5: The Schedule Timetable

The second role uses the time clock. If the schedule time table hits clock time, the ScheduleAgent sends to the SearchAgent to search articles from the Web, for example, at 12:44 on 1st. Feb. 2002. The SearchAgent searches the IntelligentAgent keyword with two users, cmszo and cmsra. The content message is (keyword, List(Userid), List(title, content, address)). We not present the SceduleAgent class diagram in this section, as the development process is similar to that of others.

*ScheduleAgent Model*

| | | | | |
|---|---|---|---|---|
| **ScheduleAgent** | | | | |
| **Goal** | Schedule the search timetable | | | |
| **Role** | Schedule time | | | |
| **Tasks** | **Collaboration with agent** | **Communication** | | |
| | | **Name** | **Message Type** | **Content** |
| 1. Received message | UserModelAgent | CA-2.2(i) | *Inform* | *userid, keyword, status search* |
| 2. Schedule time table | | | | |
| **Role** | Capture time | | | |
| **Roles:** time | | | | |
| 1. Hit the time | Internal timer and scheduler timetable | CU-3 | | |
| 2. Inform SearchAgent | SearchAgent | CA-3(O) | *Inform* | *userid, keyword.* |

Figure 5.26:ScheduleAgent Diagram

*ScheduleAgent Plan Model*

The ScheduleAgent keeps waiting for two sources of events: from the UserAgent and the clock, when it hits the search schedule timetable.

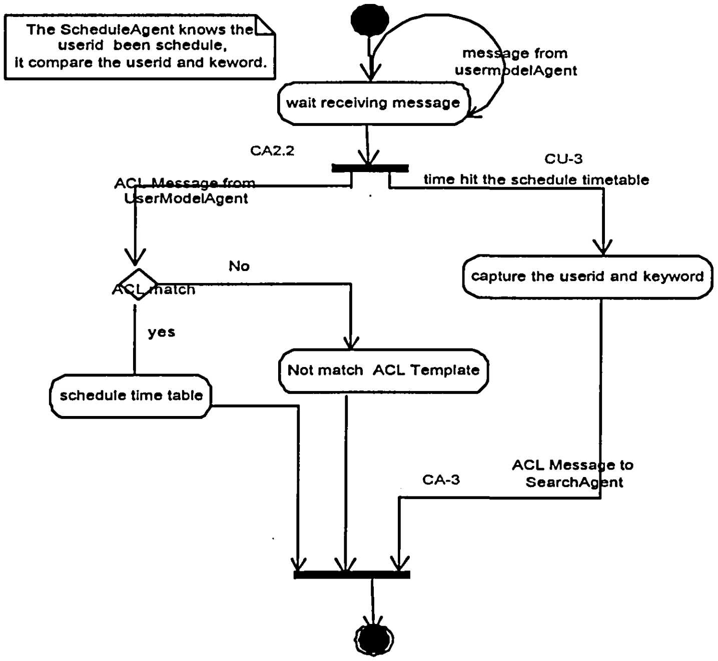

The ScheduleAgent knows the userid been schedule, It compare the userid and keword.
message from usermodelAgent
wait receiving message
CA2.2
CU-3
time hit the schedule timetable
ACL Message from UserModelAgent
No
ACL match
yes
capture the userid and keyword
Not match ACL Template
schedule time table
CA-3
ACL Message to SearchAgent

Figure 5.27: Internal Design of a ScheduleAgent.

At this stage we finished the agent system development phase, while in next section we show the agent environment development phase.

### *5.1.3. PHASE 2: Agent Environment Development*

Follow the development process of development agent environment as stated in Chapter 4, the development of the agent environment starts with identifying the agent environment's functional requirements. The second stage is to identify the architecture of agent environment design by identifying the components and relationships according to the functional requirement. The third is components design. Components vary in size and complexity. Reusing an off-the-shelf agent environment involves a similar process of identifying components but, in this case, the components consist of sub-components that provide the features needed.

The development process for reusing an the existing agent environment as stated in section 4.3, consists of four steps:

The first step is agent environment analysis to produce the concrete agent environment specification as shown in section 5.1.3.1.

The second step is to produce prioritised criteria for reuse of an existing agent environment using the common criteria of agent environments presented in section 4.3 and the agent environment specification stated in section 5.1.3.1. The prioritised criteria are presented in section 5.1.3.2.

The third step is analysis of existing agent environments that most fulfil the criteria stated in section 5.1.3.2; the result is shown section 5.1.3.3.
The fourth step is making a decision, as discussed in section 5.1.3.4.

### 5.1.3.1. Analysis Requirement of Agent Environment for Filtering Agent System

The functionality of the agent environment is extracted from the user requirement specification and agent system design. The agent system design result is identified as an enterprise requirement for development of an agent environment. The combination of the enterprise and technology requirement is used to produce the functional requirements for the development agent environment. The requirement is elaborated into three types of agent environment functional requirement as stated in the agent environment requirement in section 4.2.1. The requirements are identified based on analysis of the agent system design result and added to the standard features needed to be provided in an agent environment. The following is the justification according the design solution.

*Agent architecture functional requirement*

The agent architecture functional requirement discussed in this section follows the criteria discussed in section 4.2.1 to identify agent architecture criteria. The following are the issues related to identifying agent architecture that were captured in analysis of the agent system development phase.

1. The analysis of agent interaction as shown in Figure 5.19 represents the ACL interaction protocol used as the communication requirement of Filtering agent. Based on the agent system design result, the appropriate solution is to use any reactive architecture that is able to react with any event received internally or through other agents.
2. The agent architecture also needs to meet the criterion of autonomy. The agent system design utilises autonomous agents. This means each agent is provided with functionality to observe and capture any events that may occur. These basic features are utilised to provide pro-active agent. The agent action is based on the events captured. The agent level design stage identifies the possible events. These may come from within the internal agent, from other agents or from the external environment. Four types of events are captured: the agent management (agent status), the user, time clock (schedule time table) and other agents.
3. The agent architecture is able to identify the agent status as waiting, dead or busy. In order for agent interaction to be performed, the agent status should be not dead, except in the case of the FilterAgent sending the new articles to the UserAgent. If the UserAgent is dead the new articles are temporarily stored in a file and will be sent to the user when the user login the system next time. If the agent status is busy, the events are sent to the agent event queue.

*Communication environment requirement*

Based on discussion in the section on the communication environment in Chapter 4, there are two levels of identifying agent environment: application level and lower level as described below.

1. The analysis result shows that the types of information exchange among agents are identified. The analysis of agent interaction among all the agents shows the use of any illocution language as an interaction medium to exchange information between agents. The agent interaction analysis shows the FAS only uses the following performatives: inform and request.
2. In the agent system design, it was presented that the agent system has logical and locational distribution characteristics. UserAgent and FilterAgent are in different locations. FilterAgent and DocumentAgent are located at the same location. The decision is that all agents can communicate through TCP/IP.
3. Support the various type of message information exchanged. The analysis of the data structure of types of information exchanged shows all exchanged information is as standard text information.
4. Figure 6.9 shows the Filtering design system agent. The requirements of the system are those of distributed location, while providing a flexible environment to allocate the agents.

*Physical Environment Requirement.*

Referring to the Table 4.3, the requirements of the physical environment should consist of three issues: agent management, agent platform and agent communication channel. In order to execute the Filtering agent, the choice of the first two issues depends on the current technology of techniques to present agent management and agent platform. But the agent communication channel is identified according to the analysis of agent environment requirements. We can see that the requirement needs to agents to communicate using the standard TCP/IP and HTTP.

*Non-functional requirement.*

The non-functional requirement of the system is to have an agent environment that allows the agents to be allocated in any location.

The agent environment requirement as stated above is enough for us to justify reuse of an off-the-shelf environment rather than constructing one, for various reasons. The communication channel only uses TCP/IP and HTTP, and many current off-the-shelf environments support the above requirement. Similarly at the communication application level, only Inform and Request of ACL are used, and many off-the-shelf environments support this functionality. The physical environment is enough to use the current technology of creating the agent management and agent platform.

The reuse off-the-shelf agent environment method applied the reuse software abstraction (a separation of concerned) that has been preserved in the methodology. The next discussion shows the process of reusing the off-the shelf agent environment.

### 5.1.3.2. Criteria of Agent Environment for Filtering Agent System

Following the reuse process as stated in Section 4.3, a list of common criteria is drawn up and prioritised according to the agent environment requirement specification stated in the previous step. The following is a list of criteria used to analyse the current agent environments for the solution of the FAS.

Agent capabilities

- support basic properties of autonomy and reactivity
- support AI- provide intelligent features, fuzzy rules, genetic learning mechanism
- support FIPA ACL
- support at least FIPA request and FIPA inform interaction protocol
- adaptivity –change strategy

Technical issues

- distribution location
- support TCP/IP and HTTP
- concurrency
- open architecture

Software development issues

- level of usage -the agent environment has been applied for execution in many agent systems.
- language - Java
- supportive- the agent environment developer provides good support for installation and deployment
- easy to install- clear guidance to install the agent environment
- licence –free
- open to extend

### 5.1.3.3. Analysis Existing Agent Environment

Table 4.6 in the previous chapter showed an example of a comparison various 'off-the-shelf' with various selected features to justify to reuse an environment. However, for the purpose of the FAS solution, we use the above prioritised criteria to analyse 'off-the-shelf' agent environments. Follows the analysis process described in Chapter 4, we found eight agent environments that fulfilled the features. The following are the agent environments that fulfil most of the criteria, especially built for multi-agent-based system environments: JACK[JACK00], JAFMAS[Jaf97], JADE[JADE98], ZUES[ZEUS00], Agent Builder[BUIL00], COMTEC[COM00], APRIL[APRIL00]and Fipa-OS[FIPAOS00].

| **Product /Criteria** | **JADE** | **JAF-MAS** | **ZUES** | **JACK** | **Agent Buider** | **COM-TEC** | **APRIL** | **FIPA-OS** |
|---|---|---|---|---|---|---|---|---|
| Agent capabilities Issues | | | | | | | | |
| Basic properties | Reactive | reactive | Reactive | BDI | BDI | reactive | reactive | reactive |
| AI features | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 |
| ACL | FIPA | KQML | KQML/ FIPA | FIPA | KQML | FIPA | FIPA | FIPA |
| Support Distribution conceptual | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| Technical Issues | | | | | | | | |
| Openness | 3 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| Communication Channel | 3 | USP socket, TCP/IP (2) | TCP/IP (2) | TCP/IP (2) | TCP/IP (2) | TCP/IP (2) | TCP/IP (2) | TCP/IP (2) |
| Concurrent | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Open architecture | 3 | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| Domain application | 3 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| Software Development issues | | | | | | | | |
| Level usage | 4 | 2 | 3 | 4 | 4 | 3 | 2 | 3 |
| Language | Java | Java | Java | Java | Jess | Java | APRIL | Java |
| The Easiness to use | Yes | Yes | No | Yes | Yes | Yes | No | No |
| Supportive | 4 | 2 | 3 | 4 | 4 | 2 | 2 | 2 |
| Level Usage | 3 | 1 | 2 | 3 | 2 | 1 | 1 | 1 |
| Easy to extend | 4 | 3 | 3 | 2 | 1 | 3 | 3 | 3 |
| Availability | Open source | Open Source | Open Source | Open-fees | No | open source | open | open |
| Economic Issues | | | | | | | | |
| | 0 | 0 | 0 | 2 | 3 | 0 | 0 | 0 |

Table 5.6: The Comparison analysis for Filtering agent environment

Table 5.6 lists the most features relevant to requirements. The high value stated in table indicate the strength of the features provided. For the economic issues, the value of 0 means the agent platform is free and the assumption that the maintenance and consultation are free is also made.

### 5.1.3.4. Decision Making

Follows the step of decision making shown in the method, is to reduce the total of agent environment which meet the criteria. The Table 5.6 shows JAFMAS communication is limited because it only uses a UDP-socket for communication. AgentBuilder fulfils the requirement but does not provide open source for extension. COMTEC and FIPS-OS fulfil the requirement but does not provide detailed documentation and support and also have lower ratings for other criteria. ZEUS, JACK and JADE are the most convincing agent environments, but JACK is based on BDI architecture, while the FAS focused on reactive architecture. Comparing ZEUS and JADE, we justify from table 5.6 that JADE is more convincng than ZEUS in the following attributes: rate of usage, as is it been used in a similar domain of application (Personal assistant), easiness and supportive.

The agent architecture provides autonomous behaviour and ability to act; it also provides functionality for agents to communicate freely with other non-software agents. In addition, it is open source and provides open architecture that allows the flexible execution of agents.

In this section, we do not show the development process of the JADE environment, but rather use the JADE environment for agent development.

The next section shows the detailed design of the agent system. The detailed design is influenced by the organisation of each agent environment. The following stage shows the detailed design of each agent based on the JADE environment. The detailed design presented in the next section is based on the JADE components and structure organisation. It is at a level where it can be easily modified for implementation.

### *5.1.4. PHASE 3: Agent System Deployment*

The deployment process depends on agent architecture (part of agent environment), since the FAS is based on reactive architecture solution, similar to JADE agent ărchitecture. The deployment is according the JADE agent. The concept of deployment applies the object-oriented concept and uses UML notation. The deployment process gathers each individual agent design with appropriate JADE agent components.

At this stage, each agent defines as a class that inherits the core agent components to define an agent, use the components that define message communication, and others. The deployment represents the detailed design of each agent using the components developed in the agent environment stage. The agent level design stage is able to identify the objects used for the internal computing, maintaining agent knowledge and action influence on interaction with other agent. The deployment process as stated in the section 4.3 consists of the following four steps:

1. Creating agent.
2. Create a role object.
3. Identify action for each agent role.
4. Create task class associated with agent action and agent role action.

Each individual agent design performs these four steps to reach object levels of design such as agent class diagram.

The next stage is the implementation stage, which transforms the class diagram into implementation code using the object-oriented concept. The core implementation is at the action section, which transforms the individual agent plan model in the action section (refer Figure

4.23). The deployment of agent interaction is explained in the action section. In this section we only show the implementation code for SearchAgent to show an example of implementation code and a similar process is performed for all agents. It shows how SearchAgent receives communication from UserAgent and sends the result to DocumentAgent. Each agent is then executed and tested individually before being executed in the agent environment for further testing the execution of the agent system.The following shows the deployment of UserAgent with JADE components.

*UserAgent*

The deployment of each process transforms the agent data model as shown in Figure 4.12 into an agent class model. The refinement of individual level design shows that UserAgent is the name of an agent that consists of two roles: enquirer and feedback. Sometimes it becomes enquirer and sometimes it provides feedback.

UserAgent is created by inheriting the core agent name *Agent.* Both roles are responsible to interact with the user. For each role, its behaviour is identified. The enquirer role is identified as a 'Simplebehaviour', while feedback is identified as 'Cyclic behaviour'. Simplebehaviour and Cyclicbehaviour are the components of behaviour defined in the JADE environment that reuse present such behaviour to UserAgent.

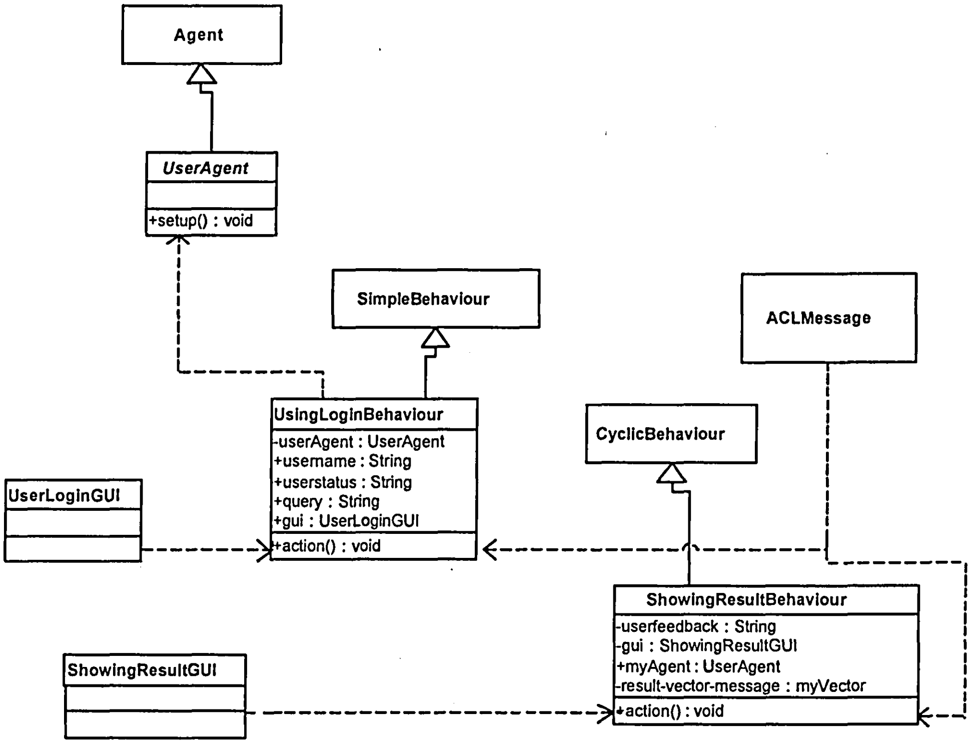


Figure 5.28: UserAgent Class Diagram.

From sequence diagram in Figure 5.12 it is possible to identify these agent role behaviours, based on the lifecycle of the role of the agent. The enquirer role consists of two tasks: first, login the system and enter the keywords and second, request the user profile. The feedback role is defined as cyclic behaviour because it waits for the result from the other agent and provides feedback.

The enquirer role is named as UserLoginBehaviour class and the feedback is named the ShowingResultBehaviour class. The class diagram of the UserAgent is shown in Figure 5.28.

The implementation in Java language of the first and second steps is as shown below:

```
public class UserAgent extends Agent {
/** UserAgent **/
public UserAgent() {
}
public void setup(){
    addBehaviour(new UserLoginBehaviour(this));
  } //setup
} //useragent class
```

*Deployment of the agent interactions.*

The deployment of agent interaction is transfers the agent interactions as stated in each agent model into ACL. The Agent State diagram is used, which integrates the agent communication with agent. The internal agent plan is located in the method *action* as shown in the agent class diagram.

This section shows an example of implementation code as described in *action()* method that follows the internal UserAgent on feeback result behaviour, which includes receiving the result from FilterAgent and sending it back to UserModelAgent. An example of implementation code is shown below:

```
Action() {
if((mt1.match(msg))) { //received
      try{
        transfer  the msg content to the ShowingResultGUI
        user provide feeback   }
        try {
        create message to send the  user feedback to UserModelAgent.
        ACLMessage msg2 = new ACLMessage(ACLMessage.INFORM);
        //create interactions
       msg2.addDest("UsermodelAgent");
       msg2.setContentObject(userfbinfo);}
```

A similar process is performed all other agents. In the next section, we show the development process of the DCS, which follows the proposed methodology as well.

## 5.2. DIABETICS CONSULTATION SYSTEM (DCS)

### *5.2.1. Requirement Specification*

The requirement specification of the DCS described here is an enhancement of the requirement of the Diabetics system as stated in [Neal+94][Neal+96], which consists of two systems, patient diabetes advisor and clinic diabetics advisor, which involve diabetes patient and doctor or physician.

The diabetes population in the world keeps increases every year. WHO reported that they project from 1995 to 2025 the number of worldwide adult population will increase 122% from 135 to 300 million people [Who98]. This means the cost of consultation, the appointment process, monitoring patient progress and the pharmacy operations of handling stock and finding better bargains from suppliers will tremendously increase. An efficient diabetic system is needed to reduce those tasks.

The scope of this system focuses on Type I diabetics, who need insulin injections based on various factors that influence the decision of dosage on insulin level. On the other hand, the agent paradigm could easily be extended with additional requirement and to deal with other types of diabetic. The following are the proposed requirement specifications for both patient and doctor.

1. The user will use the system every time insulin is taken and is able to access the system from anywhere. The patient should be authenticated every time the patient logs in to the system to enable the patient to access the detailed information including the GP name.
2. The expert system would advise/recommend the insulin dosage to be taken based on the requested parameters.
3. There are several combinations of parameter used by expert systems to advise [Neal+96]. However, this system focuses only on the following parameters: Glucose level in blood, Activity level and Diet type. The patient also needs to provide his/her health status, so the system can capture the patient's health status.
4. The patient consults the expert system every time insulin is taken by inputting all the parameters to the system and the treatment information is stored.
5. The patient needs to be reminded about when and how much insulin is to be taken by a text message on his mobile.
6. Periodic meetings with patient and doctor should be negotiated.
7. Email system should also be incorporated for ordering insulin product.

8. The system keeps triggering which patients need to be consulted by the doctor (periodical meeting or patient health status). The system proposes the patient schedule treatment and insulin dosage per treatment. However, the doctor has the right to change it.
9. The system will make sure that the pharmacy never runs out of insulin. It will automatically send email to order the insulin based on the stock and the demand for insulin according to the number of diabetic patients in treatment.
10. The system automatically negotiates time to meet when the patient health is continuously bad.

The requirement stated above presents the criteria of four agent system. The use of agents (autonomous reactive) is beneficial for the requirement of automatic negotiation for appointment making according to the scheduling, when capturing the unusual conditions in the patient. Automatic capturing of the patients status and automatic proposal for the solution is given to the doctor according the patient profile. The nature of the system is distributive, which there are various GPs located arbitrarily and users can access the system from anywhere. The communication applies in requirement no 6 (negotiate meeting appointment) and 9 (negotiate price).

Taking benefits of the agent paradigm, we believe, can cut the cost of the managing the diabetic patient and increase the efficiency for managing the patient by assigning the appropriate roles to agents.

These follows the proposed agent method, the deliverables of documentation development process consists of three phases similar to the development of FAS. Each deliverable of the development phase is presented in the following sections.

### 5.2.2. *PHASE 1: Agent System Development*

Following similar processes with FAS, the deliverables of development DCS are discussed in the subsequent sections.

#### 5.2.2.1. Analysis Requirement of Diabetics Consultation System

Following the analysis process as stated in the proposed methodology at section 4.1, the analysis process transfers the user requirement specification into analysis modelling. The analysis modelling such as system goal, event based scenario, agent use cases and agent abstraction is used for requirement elicitation.

*The Diabetic Consultation System Context Diagram.*

The *first step* of requirement analysis is to produce a context diagram of the DCS. The context diagram modelling process is described in section 4.1.1. Using context diagram modelling as proposed in the method, the DCS context diagram is illustrated as in Figure 5.29 below.

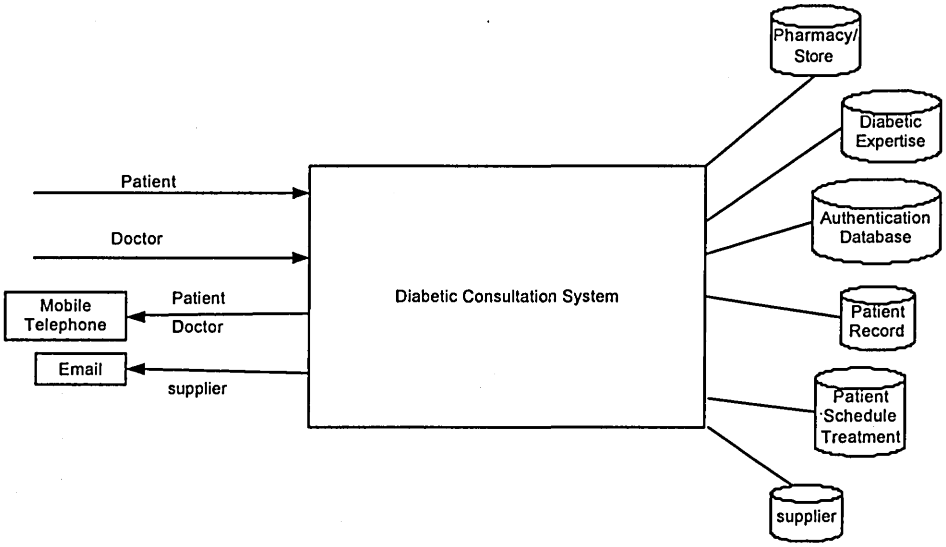


Figure 5.29: The Diabetics Consultation System- Context Diagram.

There are two actors that initiate events from the outside the system, i.e. patient and doctor. The patient and doctor will receive reminders from the system as text message in Mobile phone. The system sends email to the supplier. Six elements of knowledge are identified in relation to the requirement stated above: stock(parmacy store), diabetic expert, Authentication database(include patient information, i.e name, Gpname), patient treatment record, patient schedule treatment and insulin supplier. Other events occur from agents, which occur inside the system which we identify through out the development process.

*Capturing the System goal*

*The second step* is to identify the goal of the system. The goal of the system is identified from the objectives of the user requirement. The goal of a system is to *reduce consultation and increase efficiency of patient treatment.* There are five sub-goals captured from the user requirement. However, many additional requirements can be proposed to increase the achievement of the goal, but the boundary of the system is confined in the above requirement.

The research in this area is concerned with finding other techniques which influence the achievement of the system goal.

Using the system goal modelling described in section 4.1.1.1, the DCS goal is illustrated using a hierarchy of system goals as shown in Figure 5.30 below.

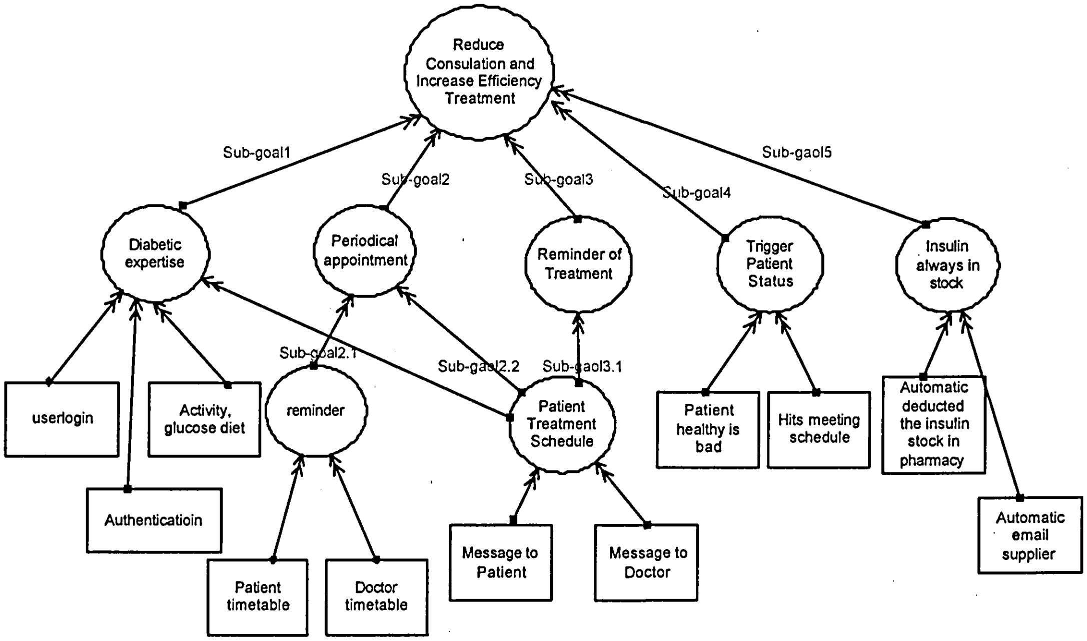


Figure 5.30: The Diabetic Consultation System Goal

The cross link at sub-goal4 shows that the subgoal4 is achieved using any two events occurs: captured status patient event or hits meeting schedule.

The goal is achieved by assigning with subsequent goals as follows:

1. Provides diabetic expertise(R3),
   a. Propose, the insulin level, as doctor or patient makes enquiry.
   b. Keep update information every time patient take insulin treatment at home.
2. Automatic periodical appointment treatment(R6,R10)
   a. Capture when doctor assigns the frequency of treatment whether daily, weekly or monthly and how much insulin dosage should be taken every treatment.
   b. Patient sets the available time for having consultation.
   c. Doctor sets available time for consultation.
3. Provides reminder consultation and treatment (R5)
   a. System reminds patient every time he should take the insulin treatment.
   b. System reminds the doctor of every consultation with patient.

4. Triggers patients that need treatment urgently or currently(R8)
   a. System captures the status patient
   b. System triggers the time for next meeting.
5. Make sure pharmacy does not run out insulin stock (R9)
   a. Automatically calculate the total insulin given to patient and deduced the insulin stock.
   b. Email to the suppliers when the stock is low and propose the right amount of insulin to order according to ratio of patient population in treatment and average insulin taken per patient.

Another modelling uses analysis elicitation to produce the event based scenario as shown below.

*Capture Scenario*

The *third step* is develop scenario. Figure 5.29 shows two actors playing roles in the system: patient and doctor. Following the scenario modelling process as discussed in section 4.1.1.2, two scenarios are produced: Scenario1 and Scenario11. Each scenario is shown in Table 5.7 and 5.8 respectively. Each activity is assigned the reference number of the relevant user requirement.

| **Scenario 1** | | |
|---|---|---|
| Event | **Enquiry** | |
| Source | **Patient** | |
| Activity | | |
| 1. | User Login (R2)<br>(Userid, Password) | Role<br>Subgoal 1 |
| 2. | Authenticate user (R2)<br>(Userid, Patient Name, GPname) | |
| 3. | Enquiry from Expert System (R3)<br>(activity, glucose, diet), health | |
| 4. | Expert System make decision R4<br>Fuzzy rules (activity, glucose, diet) | |
| 5. | Send result to patient (R3)<br>(insulin level) | |
| 6. | Update the treatment activity (R4)<br>(date, time, activity, diet, glucose, insulin dosage) | Role<br>Subgoal 2 and subgoal 3 |
| Event | Schedule | |
| 7. | Patient set his schedule meeting (R6)<br>(Date, time, free/not free) | Role<br>Subgoal 2.1 |
| 8. | Meeting agreement (R6)<br>Date, time | Subgoal 2 and subgoal 3 |
| 9. | Receiving text message (R5)<br>(patientno, message) | Subgoal 3.1 |

Table 5.7: Event User Enquiry

Based on those activities presented in the scenario associated with system goal, the role is identified. Using the role concept, it is possible to identify the role elements such as capabilities, resources and states for each activity as shown in the tables.

| **Scenario 2** | | |
|---|---|---|
| Event | **Consultation** | |
| Source | **Doctor** | |
| Activity | | |
| 1. | Trigger patient health(R8), from two source :(automatic trigger from the patient record, if health status is bad ( if more then 10 time for daily treatment, more then 2 time for weekly and monthly treatment) or periodical meeting or periodical time meeting | Role<br>Subgoal4 |
| 2. | Capture patient treatment history (R6)<br>(Date, time, patientid) | Role<br>Subgoal 4 |
| 3. | Enquiry from Expert System (R3)<br>Patient history(activity, glucose, diet, health ) | Role<br>Subgoal 4 |
| 4. | Expert System make decision R4<br>Fuzzy rules (activity, glucose, diet) | |
| 5. | Proposed patient treatment schedule and dosage to doctor (R8) (insulin level) | Role<br>Subgoal 3.1 and subgoal 2 |
| 6. | Agreement from doctor of the patient frequency treatments (R8),<br>patientid, (daily,weekly, monthly), frequency and patients insulin dosage (R8) | Role<br>Subgoal 3.1 |
| 7. | Store the patient schedule treatment (R8)<br>(patientid, vector(date, time), insulin dosage | Role<br>Subgoal 3 |
| 8. | Calculate the total insulin given to patient (R8)<br>(insulin-product name, patientid,total) | Subgoal 5 |
| 9. | Update insulin stock at pharmacy (R9)<br>Update the insulin stock | Subgoal 5 |
| 10. | Capture insulin stock and automatic send email to supplier when stock is less (R9) | Role<br>Subgoal 5 |
| Event | Schedule | |
| 11. | Doctor set his schedule meeting (R6)<br>(Date, time, free/not free) | Role<br>Subgoal 2.1 |
| 12. | Meeting agreement (R6) | Subgoal 2 |
| 13. | Receiving text message of consultation(R5)<br>(doctorid,date, time, GPname, message) | Subgoal 3.1 |

Table 5.8: Event Time Scheduler

### *Developed Use Cases for the Diabetic Consultation System*

*The fourth step* is to develop the event use cases based on each event scenario, following the event use case modelling process as described in section 4.1.1.3. The first use case is developed based on Scenariol as shown in Figure 5.31. Each activity in the scenario is transferred into the event use case. All activities in Table 5.7 are transferred into the nine use cases, named as S1A1 to S1A9. A similar step is performed to produce the Scenario II as shown in Figure 5.32. The Scenario II shows the thirteen use cases named it as S2A1 to S2A13.

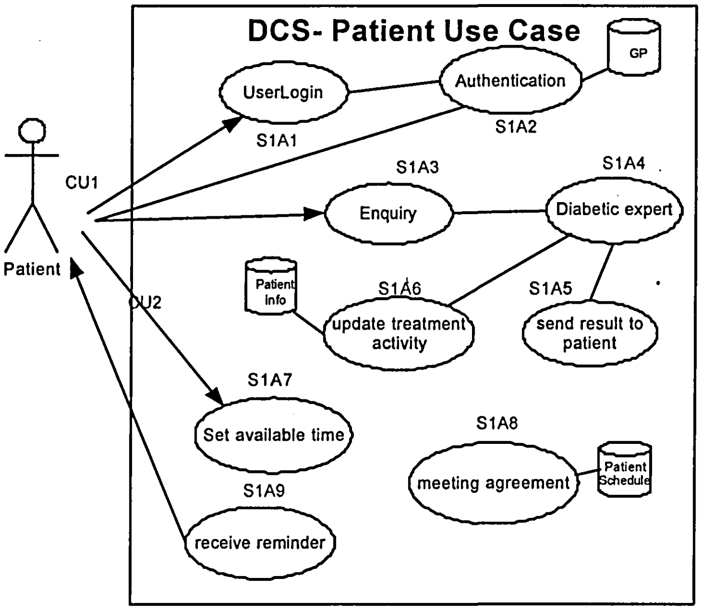


Figure 5.31: Patient Use Case

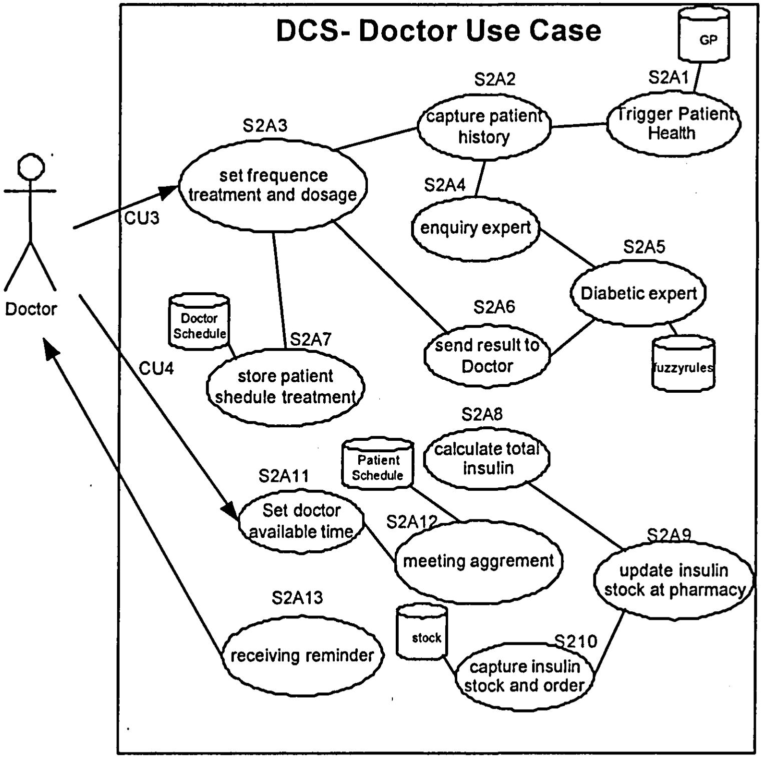


Figure 5.32:Doctor Use Case

Following the agent use case modelling process as stated in section 4.1.1.3 of the methodology, the agent system use case is developed by combining these two use cases and eliminating any redundancies of use cases. We found three similar set use cases in both patient and doctor use cases: S1A3= S2A4, S1A4=S2A5 and S1A8-S2A6. In these cases the pits value is almost similar for all use cases either 1 or 2.

As a result, the agent use case consists of nineteen uses cases as described the following elements: {S1A1, S1A2, S1A5, S1A6, S1A7, S1A9, S2A1, S2A2, S2A3, S2A4, S2A5, S2A6, S2A7, S2A8, S2A9, S2A10, S2A11, S2A12, S2A13}. The combination of the two use cases is represented in the use case for the DCS as shown in Figure 5.33 below.

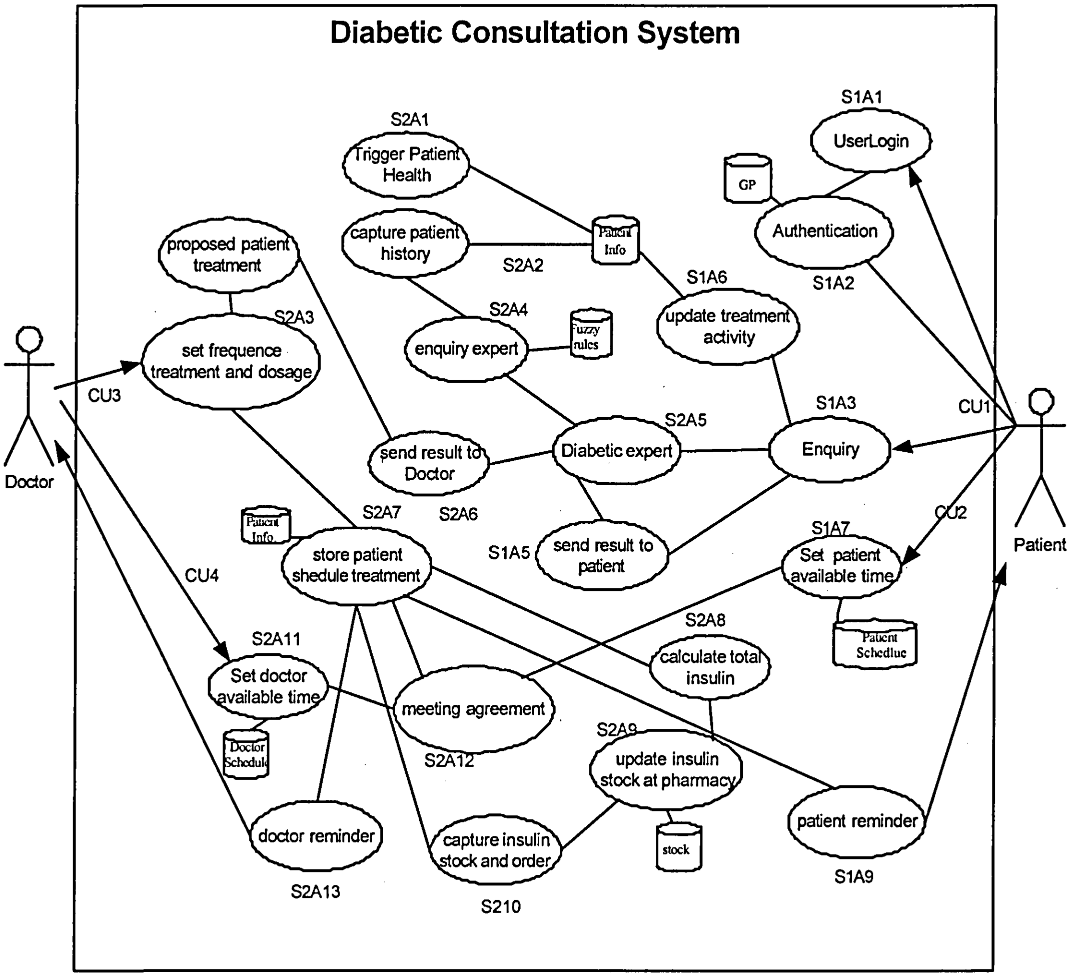


Figure 5.33: The Diabetic Consultation Use Cases

*Identify Agent Paradigm*

The *fifth step* is to identify the agent paradigm. The characteristics of agent behaviour are identified:

- intelligent behaviour (S2A5)-
  - when doctor (S2A6) and patient make enquiry (S1A3) give details of insulin dosage result according to activity, glucose and diet level (use case)

  - propose to the doctor the patient treatment including frequency and amount of insulin dosage intake according to the patient history (S2A2, S2A4, S2A5, S2A6, S2A3)
- Pro-active behaviour – there are several requirement that presents the pro-active behaviours that utilise the autonomy behaviour:
  - the system is pro-active to capture patients who need treatment urgently (S2A1)
  - pro-active to set periodical appointments between patients and doctors (S2A11, S2A12, S1A7)
  - Make sure insulin is always in stock and propose the amount of insulin to be ordered according to the growing rate of diabetic patient population (S2A8, S2A9,A2A10) and average insulin dosage intake.
  - Pro-active to remind the patient every time treatment is to be taken (S2A13).
  - Pro-active to remind the doctor of time and place of consultation (S1A9).

By analysis of the use cases presented in Figure 5.33, we can see that in order to perform the use cases a distributed and open environment is needed. So patients can access the system from anywhere and the doctor can access the system from any GP (General Practice). Therefore, the agent environment should provide openness, on which both clients may use different types of operating system. Similarly with the pharmacy stock.

To analyse the behaviour of the system is enough to use the agent paradigm of autonomous and reactive behaviour. The intelligent behaviour is provided by assigning the behaviour to particular agents rather than showing the behaviour as a result of the interactions of agents for the whole system.

Another method of identifying agent abstraction properties is examining the existing interaction protocol. The analysis of the agent use case also shows the role of automatic set meeting scheduler as an important feature of having autonomous and reactive behaviour to perform the actions.

The next stage of design is to identify agents, based on this agent paradigm and agent characteristics.

### 5.2.2.2. Agent System Level Design.

The *sixth step* is to produce the architecture of the Diabetic Consultation of agent based system. The design process starts at this step. The previous steps focused on analysis modelling. As stated in the methodology in section 4.1.2, the first design activity is to identify agents and their interactions as iterative process, which are represented in specific types of models: agent system architecture and system plan. The system plan model is illustrated in terms of two types of plan model: plan sequence diagram and plan activity diagram.

*Identifying Agent*

As stated in the methodology, identifying agents is the *first step* of designing an agent system, so roles can be assigned to agents. There are four factors that influence identification of agents: behaviour, distribution, communication and openness. Identification is a recursive process, including justifying and assessment of agents in terms of the performance using the pits value. The use case pits value is relatively small.

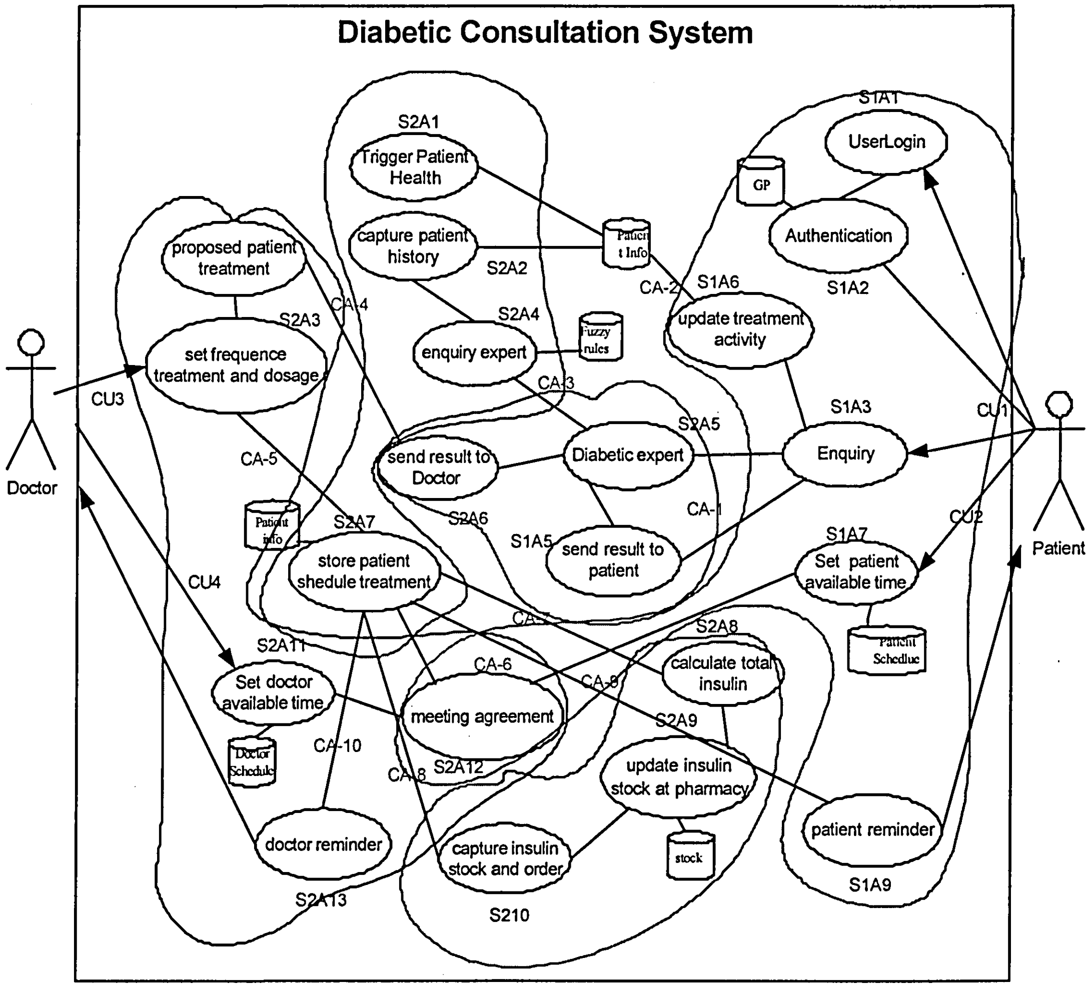


Figure 5.34: Assigning Roles to Agent

The roles presented as use cases are assigned to agents by considering the system goal and requirements. The agent paradigm and requirement are fulfilled based on the agent abstraction properties. Figure 5.34 shows roles are assigned to agents. The colour line area shows the appropriated roles assigned to agents. Five agents are identified: PatientAgent, DoctorAgent, ReminderAgent and Stock Agent. The analysis of pit does not affect the performance because the concurrent pit value is small.

At this stage, it should be possible to identify each agent's goal and roles. The agent identification is able to identify the agent interaction. The description of the agent goal, roles and summary of tasks as shown using agent abstraction to present agents is shown in Table 5.9. These agent abstractions are developed in detail at the agent level design stage. Using the agent design framework shown in Figure 3.11, we can identify the relationship between agent, roles and tasks.

| | |
|---|---|
| *Agent name*: PatientAgent | *Agent name*: Doctor Agent |
| *Goal:* Provide efficient treatment | *Goal:* Reduce consultation |
| *Roles* : user interface for enquiry<br>: user interface for patient set available time | *Roles* : list patient (health problem or time for treatment) include the proposed frequency treatment and dosage intake<br>:doctor confirmed the frequency treatment and dosage intake<br>:user interface for doctor set available time |
| Userlogin, Authentication, store treatment every treatment and receive reminder every time insulin intake | Provide Scheduling Meeting<br>Calculate insulin |

| | |
|---|---|
| *Agent name*: Expert Agent | *Agent name*: Reminder Agent |
| *Goal:* Accurate advisor | *Goal:* Reminder and capture patient |
| *Role* : Recommend Insulin to doctor and patient | *Roles* : Reminder to patient every time insulin treatment<br>Reminder to doctor every time consultation with patient<br>Captured patient health status and patient schedule treatment |
| Patient insulin intake depend to expert result<br>Proposed to doctor the patient treatment based on patient history and allowed to doctor change in value of treatment | *This agent autonomously check the patient record to trigger the unhealthy patient*<br>It proposed to the doctor of how frequent patient needs treatment and insulin dosage according to patient history |

| |
|---|
| *Agent name*: Stock Agent |
| *Goal:* Make sure insulin is always in stock |
| *Roles* : Keep checking the stock and email in case order is below the buffer. |
| Every time doctor assigned insulin to patient, the insulin stock is automatic update<br>It autonomously trigger the product insulin is nearly run out stock and order from the supplier |

Table 5.9: Agent Abstraction

*Identify agent Interaction.*

Following the proposed methodology as described in section 4.1.2.2, the second step of design the agent system is to identify agent interactions. Agent interactions are identified using similar agent use cases. The interactions between use cases are created to achieve the system goal and fulfil the user requirements. In certain conditions, the interaction will change to produce a better design by considering the four characteristics of agent systems. The agent identification is able to identify possible interaction between agents, interaction with the external environment or object agents and internal interaction. The arrows crossing the colour boundary are identified as inter-agent interactions. The arrow that crosses the colour boundary is named as shown in Figure 5.34. The agent use case is then transferred into an agent system architecture model as shown in Figure 6.35. The architecture represents the agents and their interactions as that represented the architecture of DCS.

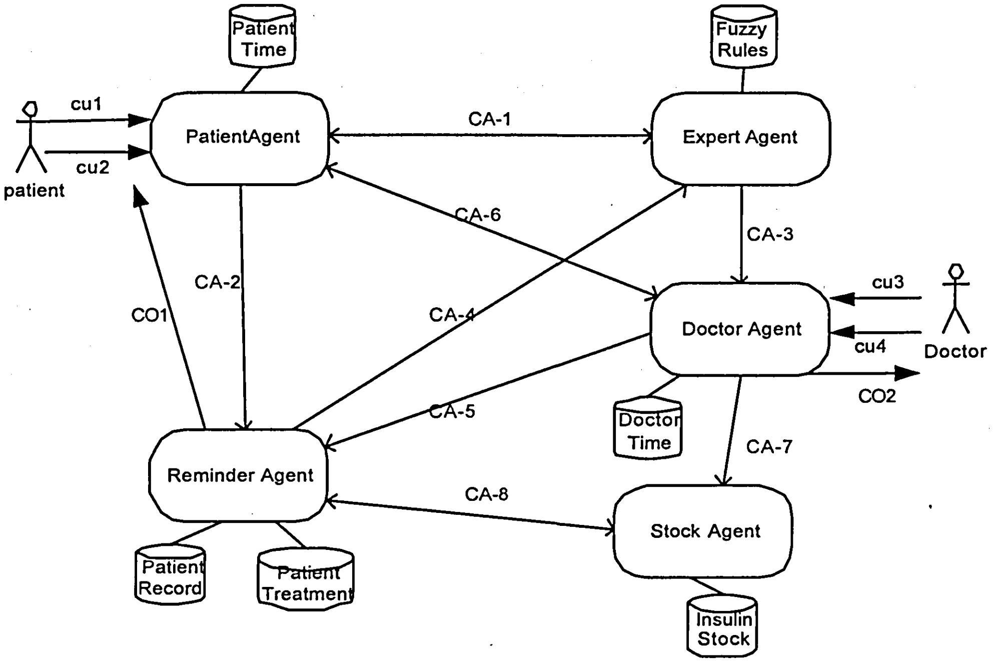


Figure 5.35: Agent System Architecture.

With such an architectural design, agents can be located anywhere. However, in the agent environment development section we will discuss in more detail how the requirements of each agent are able to produce the agent environment requirement for executing the system.

The decision of where agents are located also influences the identification of agent interaction, to produce an optimum interaction.

*Agent Interactions between Agents*

The agent system architecture above only shows that the communications between two agents occurs but does not describe what kind of interactions take place between the agents. The type of interactions is identified using the agent use case in Figure 5.35. The use cases which belong to several agents indicate that there is negotiation or collaboration among the agents. The colour area shows that the use case S1A12 belongs to PatientAgent and DoctorAgent. This means both agents have to negotiate to reach an agreement on the meeting date. This type of interaction is similar to the FIPA-contract-net interaction protocol.

The communication name CA-1 shows two ways of interaction. This pattern of interaction is the same as the FIPA-request interaction protocol. The rest of the interaction is an autonomous sending of message due to the pro-active behaviour provided to the agent. The following are the description and interaction modelling between two agents.

The communication names with prefix CU and CO indicate communication with the external object, on this case patient and doctor.

CU1 : the communication with the user through user interface for logging into the system and enquiry on treatment. The user interface design for each interaction is discussed in the section on user interface design.

CU2 : the communication from the user when the user sets the time for meeting.

CU3 : the communication from the doctor to validate the patient treatment schedule and insulin dosage to be taken each time.

CU4 : the communication from the doctor when setting an available time for meeting the patient.

CA-1 : UserAgent sends enquiry to the ExpertAgent on how much insulin should be taken based on patient's current activity, glucose and diet level, and the ExpertAgent proposes the result. This interaction is similar to the FIPA-request interaction protocol. The interaction occurs when patient make enquiry.

CA-2 : After treatment UserAgent updates the patient record treatment. This communication is as local communication with database. However, the CA-2 may communicate with ReminderAgent, if the ReminderAgent physically located at different location.

CA-3 : ExpertAgent keeps sending the proposed patient treatment to DoctorAgen, to help the doctor in making decision during consultation.

CA-4 : As the result of triggering the patient health and patient treatment consultation schedule, the ReminderAgent proposes the treatment information to DoctorAgent, It will send the patient information based on patient history and send to ExpertAgent to help on making a decision on insulin dosage. This interaction uses the FIPA-inform interaction protocol.

CA-5 : DoctorAgent sends the patient schedule treatment to ReminderAgent. Similar to CA-2, it may be local communication with databases or with ReminderAgent depending on where physically the two agents are located.

CA-6 : This communication utilises the autonomous behaviour of agents, which allows both agents to keeps communicating to set the appointment meeting between the patient and doctor. Reminder keeps capturing patients who need to meet the doctor. Based on the schedule of available time for both patient and doctor, both agents will set the meeting automatically. They will negotiate again if there is a change to the time of meeting on either side. This pattern of interaction is similar to the FIPA-contract net interaction protocol.

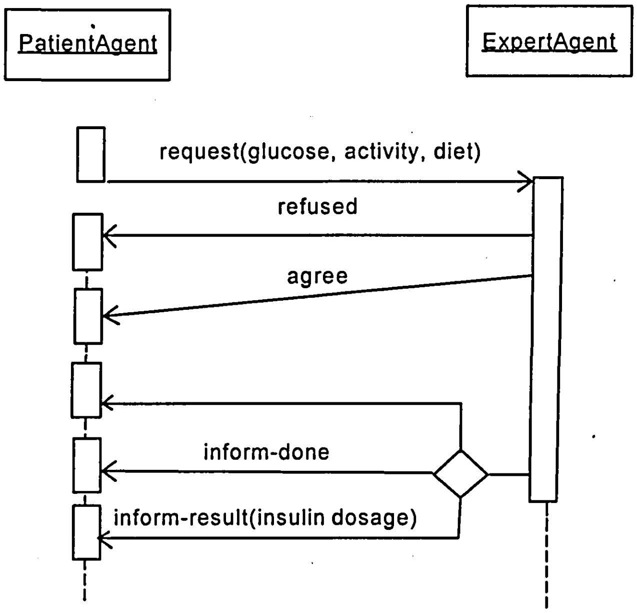
CA-7 : Every time a patient treatment schedule is decided, the information is sent to StockAgent, which calculates the total insulin and deducts it from the current stock.

CA-8 : We assume the StockAgent has the list of insulin product and contact email, so that it can order automatically when stock is low. The total amount of order is calculated based on the diabetic population and average insulin intake per person as captured by requesting from ReminderAgent.

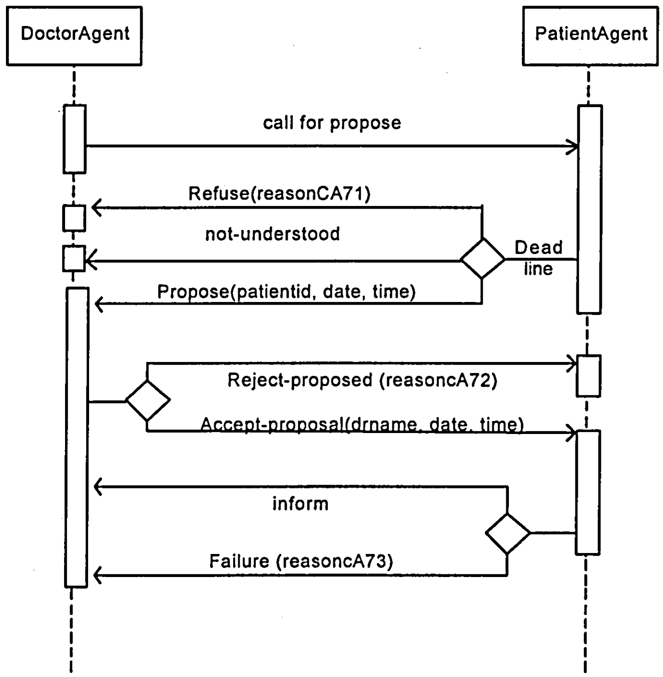
CO2 : ReminderAgent keeps sending messages to Patient when it is time to take insulin.

CO1 : The DoctorAgent keeps sending text messages to the doctor to remind him/her of the appointment with the patient.

In most cases, the interactions between the agents are simple (inform). The CA-1 and CA-8 present a request interaction protocol and CA-6 interaction presents the proposed-when interaction protocol. Both are shown in the model of interaction in Figure 5.36.


PatientAgent
ExpertAgent
request(glucose, activity, diet)
refused
agree
inform-done
inform-result(insulin dosage)

CA1: Request interaction protocol



CA-6: Proposed-when Interaction Protocol

Figure 5.36: Agent Interactions

*System Plan Model*

The *third step* of agent system design is to identify the plan of the agent system. Following the proposal methodology, there are two other types of modelling used to describe the dynamic agent interactions: activity diagram and sequence diagram. However, we argue it is enough to have the sequence diagram because the Diabetic Consultation does not present complicated agent interactions. Therefore the sequence diagram is enough to show the dynamic interaction of the system.

The system plan modelling process as stated in section 4.1.2.4 shows that each event is associated with each system plan model. However, here we show the combination of all the agent interactions as shown in Figure 5.37. The figure shows there are several source that initiate for agent interactions: patient(CU1 and CU2), doctor (CU3 and CU4), trigger patient health, schedule treatment, capture stock, patient and doctor reminder and meeting negotiation.

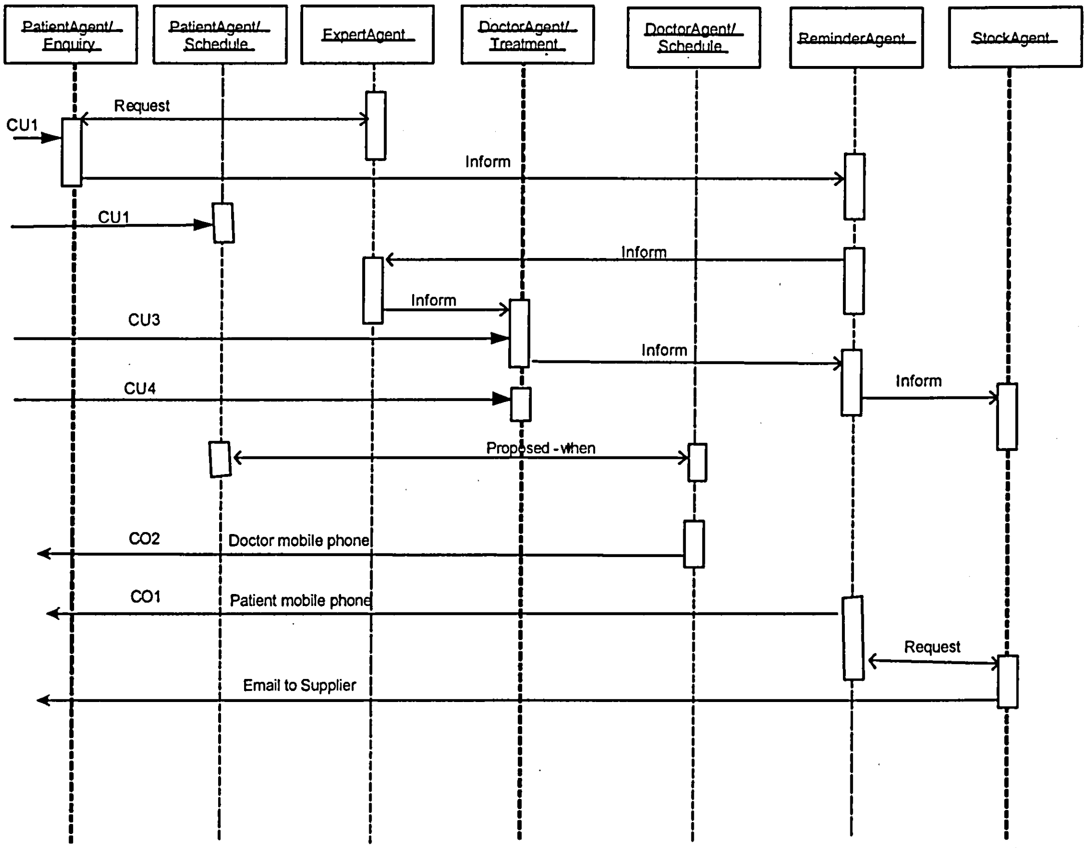


Figure 5.37: The Diabetic Consultant System Plan

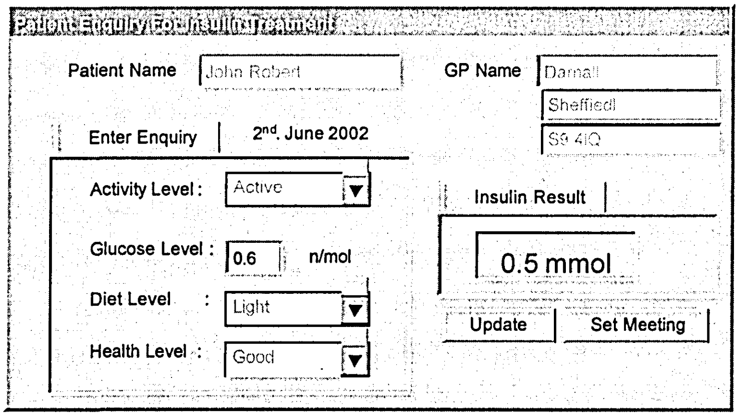
The figure shows the agent interactions are performed autonomously, which aids achievement of the system goal. The system keeps autonomously capturing patients that need treatment, keeps autonomously setting meetings, and keeps reminding patient and doctor. All these roles are performed by agents without interference with the patient and doctor.

### 5.2.2.3. Individual Agent Design

The *sixth step* is to design each agent. This proposed methodology in section 4.2.3 discusses the design process at this stage, which focuses on design of each agent. The methodology proposes two agent design models as appropriate for design modelling for each agent: *agent model* and *agent plan model.* Therefore, the individual agent design as presented below shows these two type modelling of each agent design.

a. PatientAgent

A PatientAgent is responsible for providing a user interface for the Patient that communicates with the system. Mainly there are two communications from patients, described as CU1 and CU2. The CU1 is associated with the user interface to patient login to the system and user interface enquiry, which later presents the insulin result. The second user interface is from CU2. It provides an interface to a timetable displaying the daily, weekly and monthly date and time, and when the patient is available to meet. The patient can also check the timetable to see if the doctor has set an appointment to meet. The patient also can change the timetable if he does not like the appointment time. Therefore another role for the PatientAgent is to agree the meeting date on behalf of the patient. Both patient user interface designs are shown in Figure 5.38 and 5.39 respectively.



Patient Enquiry For Insulin Treatment

Patient Name: John Robert

GP Name: Darnall<br>Sheffiedl<br>S9 4IQ

Enter Enquiry: 2nd. June 2002

Activity Level : Active ▼

Glucose Level : 0.6 n/mol

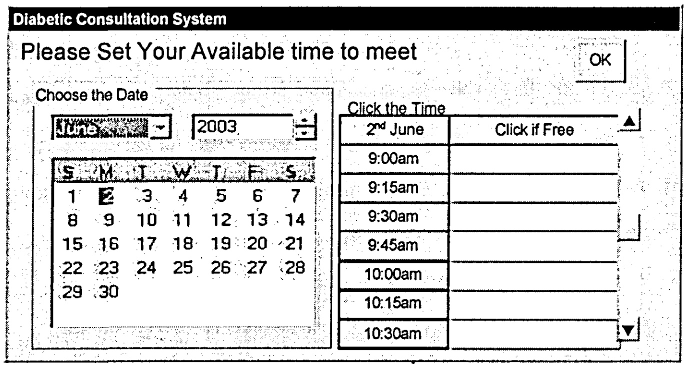
Diet Level : Light ▼

Health Level : Good ▼

Insulin Result: 0.5 mmol

Figure 5.38 Patient Enquiry

Another type of model for individual agent design is an agent model. Details of the agent abstraction model are shown in Table 4.8. The agent model provides a description of the agent name, goal, roles and tasks: internal tasks and communication tasks, as described in our agent framework design as discussed in Section 3.4.3. The detail of resource is captured from the parameters as set in the content of the message. As stated in the methodology, the agent model is developed from agent use case, system plan and scenario, in this case referring to Figure 5.34 and 5.35 and Table 5.6 and 5.7 respectively.



Diabetic Consultation System

Please Set Your Available time to meet

Choose the Date: June 2003

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | | | | | |

Click the Time

| 2nd June | Click if Free |
|---|---|
| 9:00am | |
| 9:15am | |
| 9:30am | |
| 9:45am | |
| 10:00am | |
| 10:15am | |
| 10:30am | |

Figure 5.39: Patient Set timetable

*Patient Agent Model*

The agent model consists of two roles: enquiry and set timetable, as we captured from the agent abstraction in Table 4.8. Each role associated with the user interface is shown in Figure 5.38 and 5.39 respectively. The user interface captures events coming from the user and initiates communication with other agents. Another event received by the patient is to receive a text message, where the user interface is the mobile itself. It is not the PatientAgent role but another assigned to the ReminderAgent role.

| | | | | |
|---|---|---|---|---|
| **PatientAgent** | | | | |
| **Goal** | Provide Better and Faster Treatment | | | |
| **Role** | Enquiry | | | |
| **Tasks** | **Collaboration** | **Comm..Name** | | |
| | | | **Message type** | **Content** |
| 1. User login interface | Patient | CU1 | | |
| 2.Authentication | Patient Agent | | | *Patient ID, Password* |
| 3. Enquiry | Expert Agent | CA-1(O) | *Request* | *Glucose Level, Activity, Diet, Health* |
| 4. Receive Recommended Insulin dosage | Expert Agent | CA-1(i) | *Inform* | *Insulin Dosage* |
| 5. Store Treatment | Reminder Agent | CA-2(O) | *Inform* | *Patient ID, GP name, Glucose level, activity level, diet, health level, insulin dosage), date, time* |
| **Role** | Set Meeting | | | |
| 6. Patient set schedule | Patient | CU2 | | *Patient ID, date, time, status* |
| 7. Agree meeting | Doctor Agent | CA-6(i/O) | *Contract-Net* | *Patiendid, DrID,date,time* |
| 8. Receive message | Reminder Agent | CO1 | | *Date, Time, message* |

Figure 5.40 : PatientAgent Data Model

*PatientAgent Plan Model*

Another type of model for individual agent design is the agent plan model. It describes the internal design for each agent. Following the agent plan modelling process as described in section 4.1.3, the PatientAgent plan model is produced.

The internal design agent shows the flow on receiving an event from other agents or from the external environment. The internal design describes the internal plan for achieving the agent goal. The agent plan model manages the flows of events received, associated with the action taken.

The PatientAgent plan model is developed based on the PatientAgent model and system plan model presented in Figure 5.37 and 5.40. The PatientAgent plan model is shown in Figure 5.41.

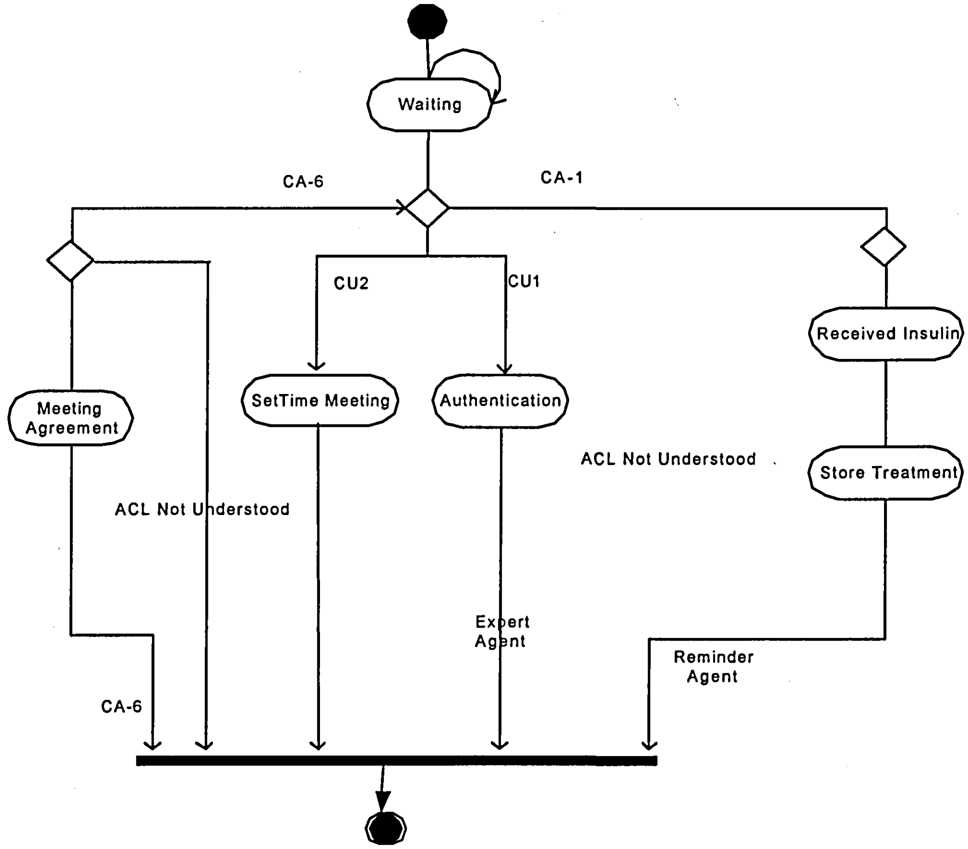
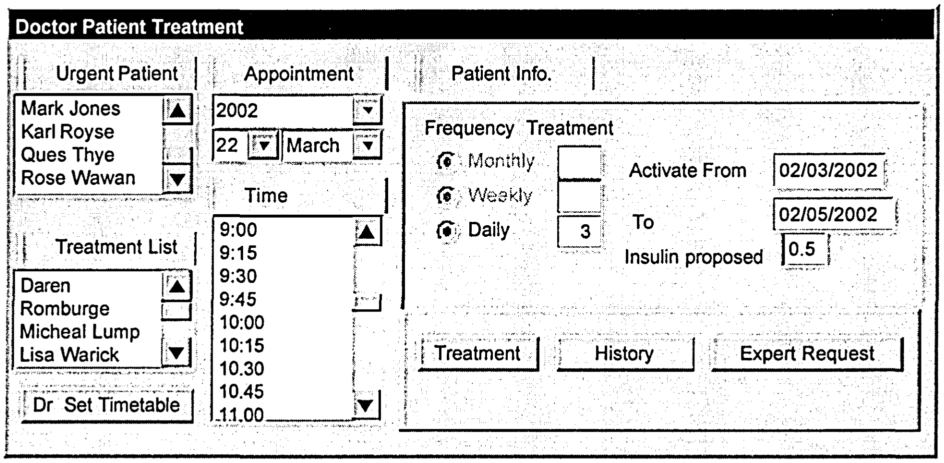


Figure 5.41: The PatientAgent Plan model

b. DoctorAgent

The responsibility of DoctorAgent is quite similar to PatientAgent; it provides a user interface for the doctor. The system plan model and agent system architecture shows that DoctorAgent consists of two roles as shown as CU3 and CU4. The first is doctor receives proposal of patient treatment sent by ReminderAgent, where the doctor's responsibility is to confirm that the patient information is correct. The user interface for patient treatment is design in similar way to the user interface for patient treatment(Figure 5.42). The user interface of the first role is shown in Figure 5.43.


Doctor Patient Treatment

Urgent Patient: Mark Jones, Karl Royse, Ques Thye, Rose Wawan

Treatment List: Daren, Romburge, Micheal Lump, Lisa Warick

Appointment: 2002, 22 March

Time: 9:00, 9:15, 9:30, 9:45, 10:00, 10:15, 10.30, 10.45, 11.00

○ Monthly
○ Weekly
● Daily 3

Activate From 02/03/2002

To 02/05/2002

Insulin proposed 0.5

Figure 5.42: Patient Treatment of Doctor User Interface

*DoctorAgent Model*

| **Doctor Agent** | | | | |
|---|---|---|---|---|
| **Goal** | Reduce consultation and efficient treatment | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **content** |
| **Role** | Patient Treatment | | | |
| 1. List of proposed patient to treatment (based on patient history and patient treatment record | Expert Agent | CA3(i) | *Inform* | *Vector (frequency (Daily, weekly, monthly), date from and end, insulin dosage, Patient ID)* |
| 2. Display the patient proposed treatment and confirmed | Doctor | CU3(i) | | |
| 3. Set the patient frequency insulin treatment and dosage | Reminder Agent | CA5(O) | *Inform* | *Frequency (Daily, weekly, monthly), date from and end, insulin dosage, Patient ID.* |
| **Role** | Meeting Scheduling | | | |
| 4. Doctor set available time | Doctor | CU4 | | *Date, time, status free* |
| 5. Negotiate meeting date | User Agent | CA-6(i/O) | *Contract Net* | *Date, time, patientid* |
| 6. List of meeting date with patient | DoctorAgent | | | *Vector(Patientid, date, time)* |
| 7.Reminder Doctor for consultation through mobile | DoctorAgent | CO2 | | *Patientid, date, time* |

Figure 5.43: DoctorAgent Model

The second responsibility, to set the doctor's available time to meet, is similar to the Patient's user interface for setting meetings as shown in Figure 5.43. Another role of DoctorAgent to is negotiate with PatientAgent to set an appointment that suits both patient and doctor. DoctorAgent and PatientAgent will negotiate to set meeting. Besides that, the DoctorAgent keeps sending reminders of the appointment date and time for by sending a text message through mobile phone. Following a the similar process to PatientAgent, the DoctorAgent model is developed as shown in Figure 5.43 and the PatientAgent plan model is shown in Figure 5.44.

*DoctorAgent Plan Model*

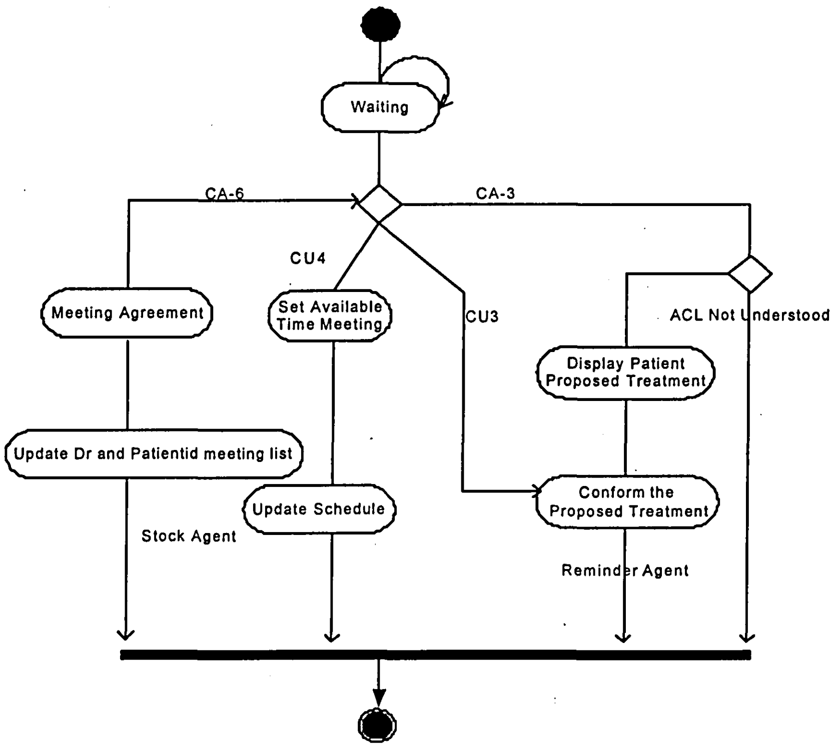

Waiting
CA-6
CA-3
CU4
Meeting Agreement
Set Available Time Meeting
CU3
ACL Not Understood
Display Patient Proposed Treatment
Update Dr and Patientid meeting list
Update Schedule
Conform the Proposed Treatment
Stock Agent
Reminder Agent

Figure 5.44: DoctorAgent State Diagram

c. ExpertAgent.

The Diabetic Consultant system plan shows that the role of ExpertAgent is to help the doctor and patient to identify the right amount of the insulin to be taken every time patient is needed, which depends on the following parameters: the patient's activity level, glucose level in blood and patient diet level. The patient does not need to ask the doctor how much insulin should be taken every time he/she takes insulin. The current practise is that doctor gives the patient a scheduled time-table of insulin intake, such as 3 times a day or week, etc. and the patient calculates the amount of insulin intake based on glucose level only, which is not accurate. In order to have an accurate insulin amount, the patient should to ask the consultant every time he/she takes insulin. This increases the consultant's time in line with the percentage of diabetes population.

With this expertise, the patient will be provided with the right amount of insulin intake without consultation from the doctor. From the doctor's perspective, the ExpertAgent is used to help the doctor to make decisions.

The DCS architecture shows that the ExpertAgent keeps receiving enquiry from ReminderAgent of the patient that need treatment and ExpertAgent proposes the right amount of insulin to the doctor. Therefore, it helps the doctor to monitor the patient without observing the patient's status and history. The ReminderAgent works together with ExpertAgent and proposes the accurate information on patient treatment to doctor, which is capturéd according to the pattern of patient history, and recorded every time insulin is taken, such as insulin dose, glucose level, activity level, diet and health level. The exact dosage needed to varies among patients. By this method, fuzzy rules for each patient can be created based on the pattern of patient history.

Following the previous modelling process, the ExpertAgent model is shows in Figure 5.45 and the ExpertAgent internal plan model is shown in Figure 5.46.

*ExpertAgent Model*

| **Expert Agent** | | | | |
|---|---|---|---|---|
| **Goal** : Provide Accurate Diabetic Treatment | | | | |
| **Role** : Propose Insulin Result | | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **Content** |
| 1. Receive Patients Request | Patient Agent | CA1 | *Request* | *Glucose level, activity level, Diet.* |
| 2. Send Recommendation | Patient Agent | CA1 | *Inform* | *Insulin Dosage* |
| 3. Enquiry | Reminder Agent | CA4 | *Inform* | *Glucose Level, Activity, Diet, Patient ID* |
| 4. Proposed Patient Treatment | Doctor Agent | CA3 | *Inform* | *Patient ID, Frequency, Insulin level, Duration.* |

Figure 5.45: ExpertAgent Model

*ExpertAgent Plan Model*

ExpertAgent keeps waiting for events coming from the PatientAgent or ReminderAgent. The role is similar but different information is sent as described in the content language.

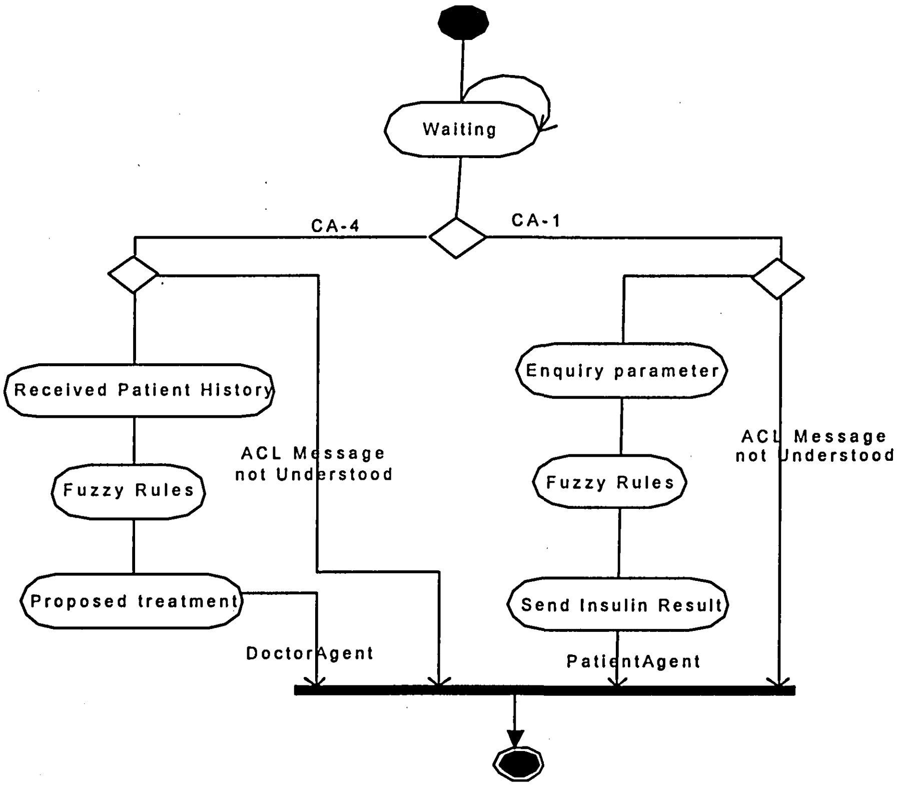


Figure 5.46: ExpertAgent State Diagram

The fuzzy rules for making decisions on how much insulin dosage are needed (based on the user enquiry parameters). It is a rule-based design. The three categories of parameters described as below.

- activity level (sleep, very light, light, strenuous, very strenuous)
- glucose level (very low, low, ok , high and very high )
- diet level is ( none, very light, light, heavy, very heavy)

The detail fuzzy rules design is presented in [Othm+02c]. We are not focused on designing the fuzzy rules because it follows the knowledge based approach.

d. ReminderAgent.

A similar modelling process is performed for ReminderAgent. The two appropriate models for ReminderAgent are shown in Figure 5.47 and 5.48.

The ReminderAgent stores the PatientTreatment and Patient Treatment Schedule. The Patient Treatment captures patient health history and captures the pattern of treatment suitable for the patient. The Patient Treatment Schedule sets the timetable describing when patient should take insulin treatment. This timetable is automatically generated when the doctor sets the patient's treatment schedule.

For tasks of 3 to 5 ReminderAgent displays autonomous behaviour. It keeps capturing the patient's health data and treatment time. If it captures these two conditions, ReminderAgent will take action. Figure 5.48 shows the tasks of capturing patient status and treatment date. The patient information is then sent to ExpertAgent, e.g. when the patient categories as bad condition,

*ReminderAgent Model*

The appointment date starts to be negotiated, when the treatment schedule is nearly finished depending on the duration of the schedule. If it is on a 3 month basis, the patient starts to negotiate the meeting time a month before finishing the current schedule. If on monthly basis, it should be not more than a week before the schedule treatment is finished.

| **ReminderAgent** | | | | |
|---|---|---|---|---|
| **Goal** | Reminder to Patient and Doctor | | | |
| **Role** | Reminder | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **Content** |
| 1. Store Patient Treatment | Patient Agent | CA-2(i) | *Inform* | *Pateinet ID, GP name, glucose level, activity level, diet level, health level, insulin dosage)* |
| 2. Store Patient Treatment Schedule and Dosage | Doctor Agent | CA-5(i) | *Inform* | *Patient ID,( Start and Finish Date, Time), Frequency, insulin level* |
| 3. Remind Patient for Insulin with Text Message on his Mobile | | C01 | | *Date, Time, Insulin Dosage Quantity* |
| 4. Keep captured the patients treatment date | ExpertAgent | CA-4(O) | *Inform* | *PatientID,Gpname, Average( glucose, activity, diet, health)* |
| 5. Capture the patient health status is bad | ExpertAgent | CA-4(O) | *Inform* | *PatientID,Gpname, Average( glucose, activity, diet, health)* |

Figure 5.47:ReminderAgent Model

With this method, doctors do not need to monitor all the patients but only monitor the patients who really need it. In fact, the doctor can monitor the patient and postpone the next periodical consultation, if the doctor sees the progress of the patient's health is good and the doctor may decide not to make a consultation but to assign the patient's treatment schedule without meeting the patient, and the patient does not need to know the changes. The patient only relies on the reminder from his mobile.

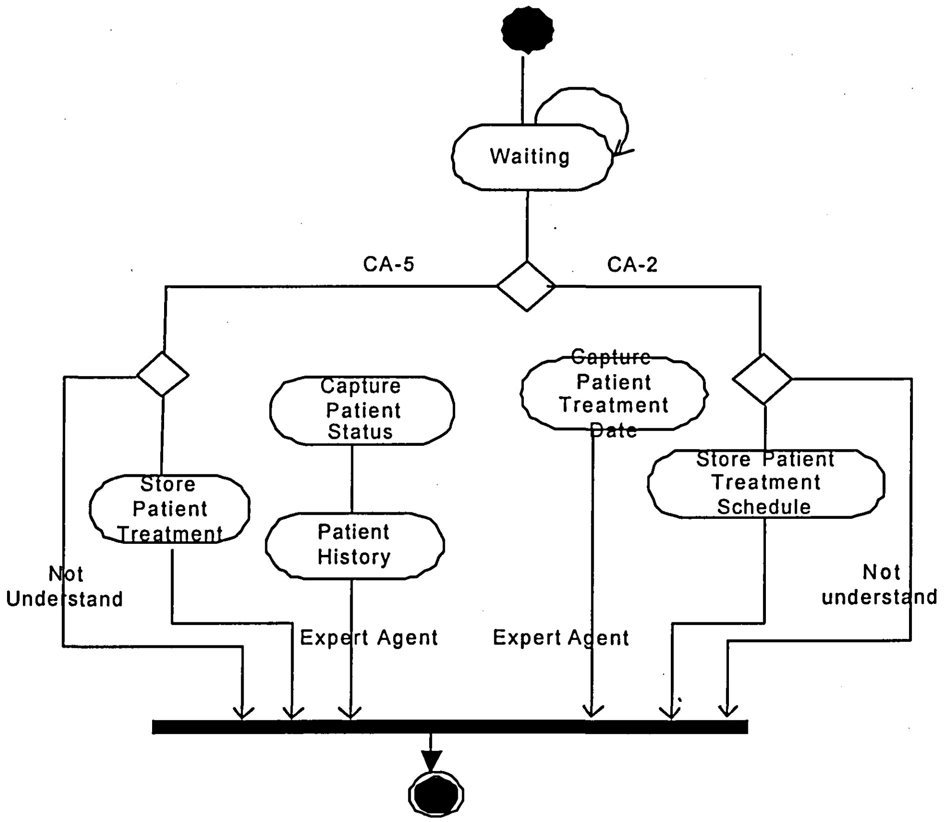


Figure 5.48: Reminder Agent State Diagram

e. StockAgent.

A similar design modelling process is performed by the Stock Agent. The two models that describe the internal agent design are shown in Figure 5.49 and 5.50. Every time the doctor assigns the patient a treatment schedule on how frequently insulin is to be taken per day/ week/ month, what insulin dosage is proposed and the duration of the treatment, it sends the information to the Stock Agent and it calculates the total amount of insulin needed for the patient and when the patient takes the insulin from the pharmacy, the current amount of insulin product in stock is automatically updated.

If the amount of stock is low, Stock Agent can automatically order the insulin product. In order to order the right amount, the StockAgent requests ReminderAgent on the percentage of diabetes population and average insulin needed per person. Therefore, the StockAgent can order the right amount of insulin.

*StockAgent Model*

| **Stock Agent** | | | | |
|---|---|---|---|---|
| **Goal** | To make sure the stock is in store | | | |
| **Tasks** | **Collaboration** | **Comm. Name** | | |
| | | | **Message type** | **Content** |
| 1. Received Patient Treatment Schedule | Doctor Agent | CA-7(i) | *Inform* | *Patient ID, Frequency, Insulin level, Duration* |
| 2. Calculate Total Insulin ( Frequency * Daily * Weekly * | | | | |
| 3. Deduct from Stock (Based on order). | | | | |
| 4. Keep capture the level of stock of a product | | | | |
| 5. Request Patient Population and average insulin dosage | ReminderAgent | CA-8(O) | *Request* | *Productinsulin,* |
| 6. Received result | ReminderAgent | CA-8(i) | *Inform* | *Totalpatient, Average insulin* |
| 7. Send email to supplier | *Suppliername,supplieremail address, producinsulin name and total order.* | | | |

Figure 5.49: StockAgent Model

*StockAgent Plan Model*

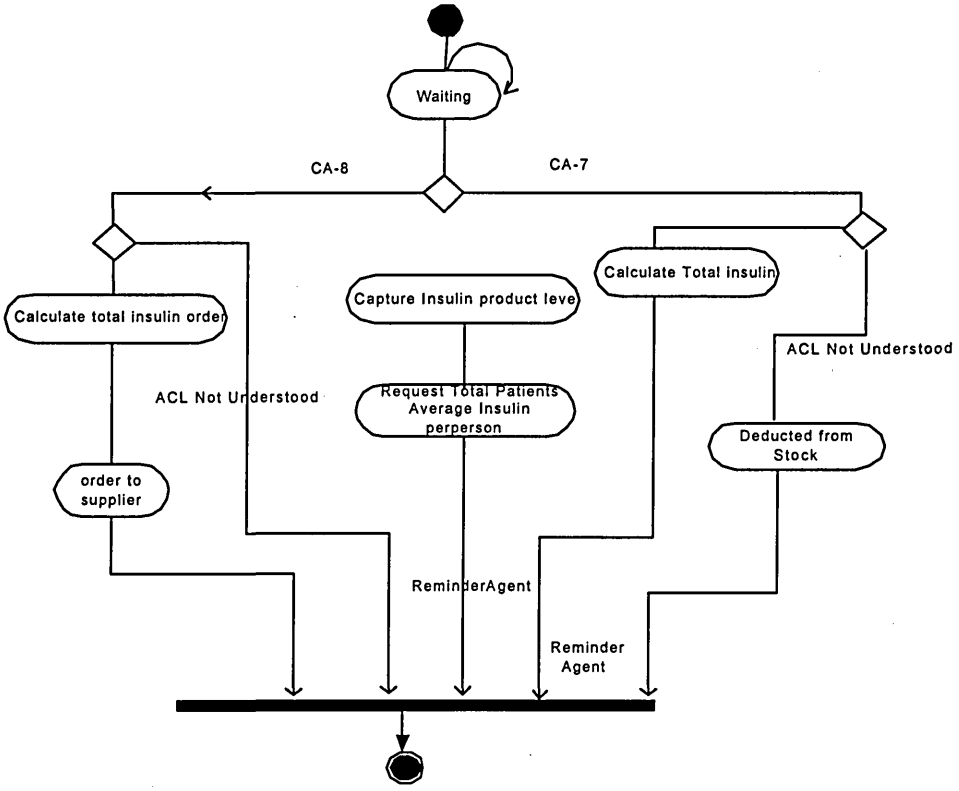


Figure 5.50: StockAgent State Diagram

At this stage, it shows the agent environment development phase.

### *5.2.3. PHASE 2: Agent Environment Development*

Following the proposed agent method shown in Chapter 4, this section discusses the second phase of the development of agent system, i.e. agent environment development, which is a recursive development process of three stages: analysis agent environment, architectural design and component design.

Before analysis of the agent environment, the agent environment requirement specification should be identified. Figure 5.51 shows the high-level architecture of the execution of DCS used to capture part of the requirement.

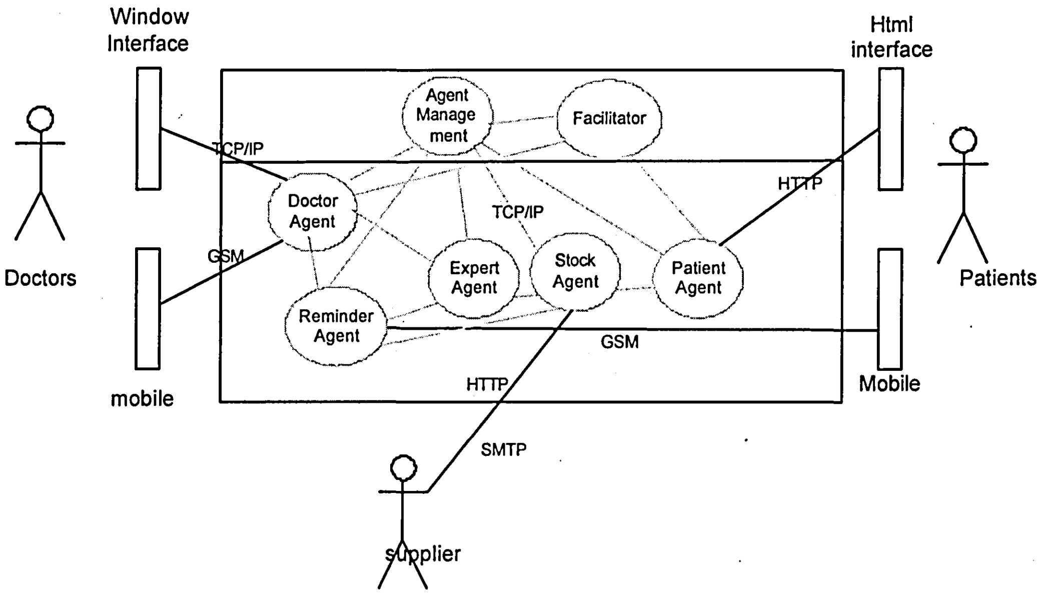


Figure 5.51: Diabetic Consultation Execution architecture

The figure shows the DCS needs ACL to communicates among the gents, actor, resource and agent management. It also shows that DCS uses various kinds of communication channels that need to consider. In Chapter 2 and Chapter 4 has discussed the common agent environment requirement for agent environment development needs to be considered. The figure shows that communication among the agents preserves the ACL and communication channel using TCP/IP, while communication with other external objects uses various types of communication.

The following section presents the agent environment analysis needed for the execution of the DCS.

5.2.3.1. Agent Environment of Agent Environment for Diabetic Consultation System

Section 4.2.1 describes the process of identifying the agent environment consisting of enterprise and technology environment, which is elaborated into three types of agent environment functional requirements. Following the proposed agent method, the agent environment requirements is captured from requirement of the system described in section 5.2.1, 5.2.2 and Figure 5.51 to captures part of the requirement.

*Agent architecture functional requirement*

Enterprise requirements:

1. The feature of agent architecture is agent capability, which is captured from identifying the agent paradigm. As described in the fifth step of agent system development, it is autonomous and reactive behaviour. This means each agent is provided with functionality to observe and capture any events that may occur. These basic features are utilised to provide pro-active agent. The agent action is based on the events using captured.
2. Patient uses HTML user interface to access, e.g. the doctor uses any window operating system.
3. Both patients and doctors receive reminders through mobile phone using the second or third generation of mobile phone technology. [Wire00] provides information on the integration of digital data to mobile devic es.

Technology requirements

1. Provide agent states: busy, dead, remove, waiting.
2. Provide an open architecture, so agents can be allocated anywhere.
3. Based on the agent system design result, the appropriate solution is to use any reactive architecture that is able to react with any event received from internally or through other agents.
4. Provide with capability to communicate with other agents and other objects through STMP and mobile.

The choice of technology is an important part of how to perform those requirements. In reality, the solution will use the current technology are available.

*Communication environment requirement*

Technology requirements

1. The agent system design presents the conceptual and locational distribution characteristics. The fast growing patient population needs flexibility for agents to be located anywhere.

2. Provide with standard information exchange among agents using standard ACL. Provide interaction protocol of FIPA-inform, FIPA-request and FIPA-proposed-when.
3. The analysis of the data structure of types of information exchanged shows all exchanged information is as standard text information.

*Physical Environment Requirement*

1. The physical environment requirement stated the physical requirement of inter-agent communication. It should be provided with aplatform, that provides the agent management functionality. Chapter 3 discussed the technology used to provide agent management among agents and agent management of interaction. The used of CORBA platform because it provides heterogeneous distributed criteria.
2. Communication should support the following communication channels: HTTP, SMTP, GSM and TCP/IP.

5.2.3.2. Criteria of Agent Environment for Diabetic Consultation System

Based on the outcomes of the requirements analysis above, the decision should be made for development agent environment approach, whether to develop a bespoke agent environment or reuse an off-the-shelf environment or reuse part of the components of an existing agent environment. All these approaches preserve the proposed methodology.

At this stage, the decision is made to reuse an off-the-shelf agent environment, but we do not justify how far we can reuse the existing components or agent environment. Therefore, the designer observes various off-the-shelf agent environments to see, which match the agent requirement as stated above. The process is similar to the process of identifying off-the shelf agent environment as discussed in the section on agent the environment phase of the FAS case study, but the DCS provides the priorities of the following criteria:

- Support communication through mobile, using GSM or Bluetooth. Refer to the [Wise00] for these technologies.
- Support SMTP – sending email to supplier.
- Support Fuzzy rules. Refer to [Fuzzy00] for use of use the technology.
- Can integrate with HTML user interface.

As the result, the prioritised criteria of 'off-the-shelf' agent environments listed as below:

Agent capabilities:

- support basic properties of autonomy and reactivity.
- support AI- provide intelligent features, fuzzy rules, genetic learning mechanism.
- support at least FIPA request, FIPA inform and FIPA contract-net interaction protocol.

Technical issues:

- distribution –support heterogeneous distribution of agents.
- support TCP/IP, HTTP and SMTP, GSM communication.
- concurrency.
- real time.
- open architecture.

Software development issues:

- easily to add an additional features
- level of usage -the agent environment has been applied for execution in many agent systems.
- language - Java
- supportive- the agent environment developer provides good support for installation and deployment.
- easy to install- clear guidance to install the agent environment
- licence –free

### 5.2.3.3. Analysis of Existing Agent Environments

The analysis process is similar of the FAS. We used the same multi-agent based system environments: JACK, JAFMAS, JADE, ZUES, Agent Builder, COMTEC, APRIL and FIPA-OS, with different prioritised criteria. The following show how the agent environments fulfilled the criteria.

| **Product /Criteria** | **JADE** | **JAF-MAS** | **ZUES** | **JACK** | **Agent Buider** | **COM-TEC** | **APRIL** | **FIPA-OS** |
|---|---|---|---|---|---|---|---|---|
| Agent capabilities Issues | | | | | | | | |
| Basic properties | Reactive | reactive | Reactive | BDI | BDI | reactive | reactive | reactive |
| AI features | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 |
| ACL | FIPA | KQML | KQML/FIPA | FIPA | KQML | FIPA | FIPA | FIPA |
| Support Distribution Location | 3 | 2 | 3 | 3 | 3 | 2 | 2 | 2 |
| Technical Issues | | | | | | | | |
| Openness | 3 | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| Communication Channel | HTTP<br>SMTP<br>TCP/IP | UCP socket,<br>TCP/IP | TCP/IP | TCP/IP | TCP/IP | TCP/IP | TCP/IP | TCP/IP |
| Support GSM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Concurrent | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 |
| Open architecture | 3 | 1 | Yes | Yes | No | - | - | Yes |
| Software Development issues | | | | | | | | |
| Easy to extend | 4 | 3 | 4 | 2 | 1 | 3 | 3 | 4 |
| Level usage | ++++ | ++ | +++ | ++++ | ++++ | +++ | ++ | +++ |
| Language | Java | Java | Java | Java | Jess | Java | April | Java |
| Supportive | 4 | 2 | 3 | 4 | 4 | 2 | 2 | 2 |
| The Easiness to use | 4 | 3 | 3 | 3 | 3 | 2 | 2 | 2 |
| License | Open source | Open Source | Open Source | Open-fees | No | open source | open | open |
| Economic | | | | | | | | |
| | 0 | 0 | 0 | 2 | 3 | 0 | 0 | 0 |

Table 5.10: The Comparison analysis for Diabetic Consultation Agent Environment

The main difference between of the DCS and Filtering agent application is that various kinds of communication channel are used such, as HTTP, SMTP, TCP/IP and GSM. Table 5.10 shows that JADE provides the most support for various kinds of communication channel. However, none of the existing agent environments provide a GSM communication channel. Therefore the criterion of 'easy to extend' becomes a priority.

5.2.3.4. Decision Making

Following the decision making process stated in proposed of agent method, we try to eliminate the criteria that diminish the priority for the DCS. The result shows that none of the existing agent environments meet all the Diabetics Consultation requirement. Therefore, we decided to reuse some part of the existing agent environment that most supports the criteria and makes it easy to add other features, when integrated with other components. This approach preserves the proposed methodology as shown in Figure 4.17 (the component-based structure of architectural design).

In this case, we reuse an agent platform which similar agent architecture and basic communication. Based on the analysis of existing agent environments, conducted in the first case study, we decided to reuse the JADE environment as a component of the agent environment for the Diabetic system and focused on agent environment that is easy to extend.

Even though the list of prioritised criteria is changed, JADE was found to be the most convincing agent environment to reuse. This is because it provides more alternative communication channels and the highest value of easiness for extension. The economic factor and the research objective also influence the decision making as well.

The integration of the JADE component with other components applies the software architecture development process. The technology for integrating digital data in TCP/IP into mobile device is described in more detail in [Wise00]. The components used for sending messages to mobile phone are discussed in [Phone02][Phone03.] The fuzzy rules to provide intelligence to expert agents use the fuzzy rules component available in [Fuzzy00] and similar to the technology for sending email to suppliers can be found in [Mail01].

The identification of a reusable component or development of a new component is part of the agent environment development process. Changes in technology will lead the changes in the agent environment and change the deployment agent system as well. For the purpose of the research, we are not going to present how those components are integrated, as it is beyond of the scope of the research.

### 5.2.4. PHASE 3: Agent System Deployment

The deployment process of DCS is similar process of the UserAgent deploys with JADE component in FAS. This means there are five agents needs to deploy in order to execute the DCS. The class diagram for each agent should shows the integration of other components (not supported by the JADE environment) are integrated in the deployment stage.

In this section, Figure 5.52 shows an example how applet components integrates with Patient Agent shown in class diagram.

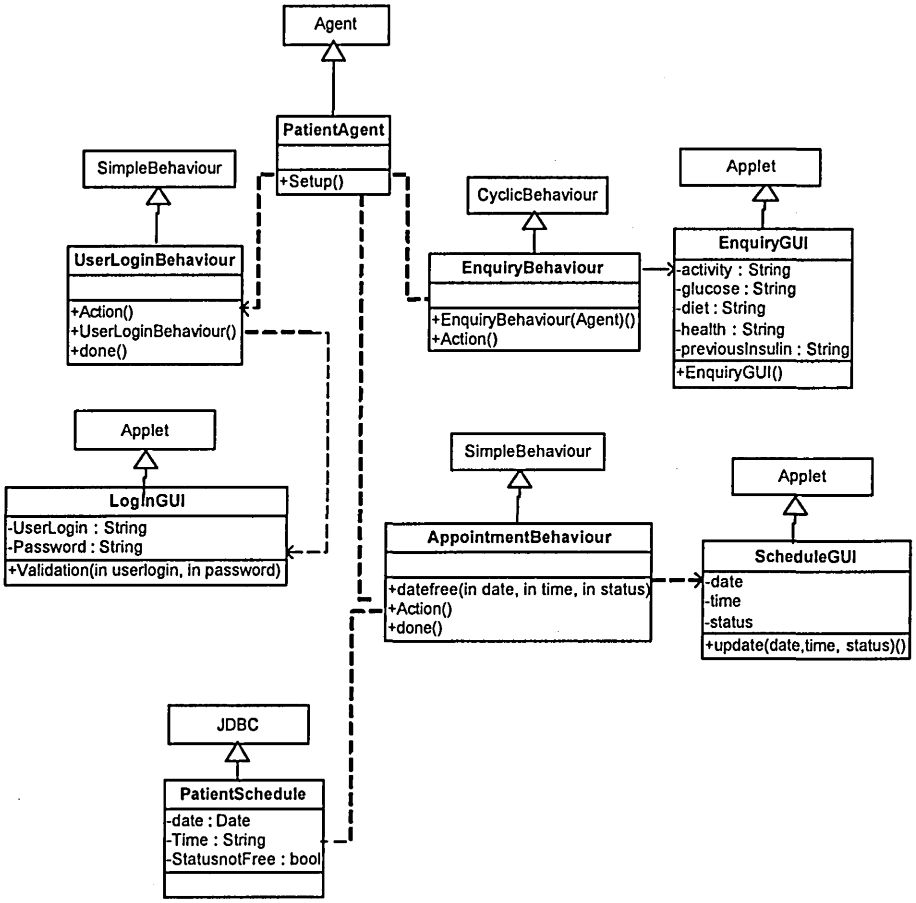


Figure 5.52 :Patient Agent Class Diagram.

The figure shows the transformation of agent roles as shown in the agent model into a class diagram by assigning the type of behaviour as *cyclicbehaviour* or *simplebehaviour.* The Patient Agent model in Figure 5.40 consists of two roles, which are assigned into *Enquirybehaviour* and *AppointmentBehaviour classes.* The *UserLoginBehaviour* is a class of local action behaviour as is task no 1. The internal plan model as shown in Figure 5.41 shows the relationship between the events received in role and the action of the role. This design role is assigned in *action()* method as described for case study one. The class diagram shows the deployment of the agent component, applet component, JDBC and behaviour component. The ACLMessage component is not shown in this class diagram. It is used as a component of the interaction of role in action() method to create interaction between agents as we show in the example of implementation.

The similar processes are performed to other agent as well. The deployment process is similar process to deployment of FAS.

The aim of the proposed methodology is to show systematic transformation from stage to stage, which at this stage is transformed from an agent model design to an object level design. It shows the transformation of agent properties into object properties.

## 5.3. SUMMARY

This capture has shown the applicability of the proposed agent method by showing two development agent systems: Filtering and Diabetics agent system. The deliverable documentation shown in this capture presented the development process that applies the proposed of agent method. It showed the development process from the requirements to the deployment stage (very close to the implementation stage). The documentation describes how the development was followed the step by step process of the proposed of agent method shown Chapter 4.

The deliverables of documents and modelling in each stage has provided a clear process of agent development. Each stage provides appropriate analysis and design modelling. The deliverable of document describes clearly the system requirements, then, showed how to abstract the user requirements into system goal, scenario, use cases and so on. The transformation process is clearly identified. With such a method, the system is easy to understand and easy to maintain, if we compare with the thousands of lines of implementation code which are difficult to understand and maintain, which were common when our research is started.

In the agent environment development phase, we did not show the case studies of development of bespoke agent environments but rather reused an existing agent environment. However, it preserved the proposed agent method. DCS shows that the JADE environment is a component that can be reused, but does not fulfil all the agent environment requirements for executing the DCS; therefore the JADE environment is integrated with other components to present the appropriate agent environment for DCS. This method shows the nature of development agent environment that applies the software architecture approach. A component may be small or big, depending on how we can abstract and separate the components, reuse and integrate them. On the other hand, the development of the agent environment is tied to technology. It changes to a better technology to perform the function, but this does not change the requirements as we noted in discussing quality aspect of development. The changes of the technology may change the architecture of agent environment. It influences the changes of the agent system deployment phase as well. For example, the first version of the development DCS is developed in JADE Version 1.4 and now JADE environment is upgraded to Version 2.6 which influences the techniques and technology which are used such as sending message methods. The implementation of agent system is changes conjunction to the changes of the agent environment. The development process follows the spiral model of the three phases of development agent system. Chapter 4 had showed the agent environment development process by re-casting the development of JADE environment that follows the proposed development process.

The agent system deployment phase has shown the deployment process at the level of individual agent design. The integration of components is shown in the object class diagram. The class diagram is a very low level of agent design, which can be transformed into implementation code using tools such as Rational Rose [Rati97]. There are other object models suitable for this design level such as the collaboration diagram and sequence diagram, but our research did not focus on them (follow the UML diagram notation). The class diagram at the deployment stage of the individual agent design is able to show the integration of individual agent design with the components uses by each agent executed in the environment to perform it roles.

The milestones and deliverable of documents in each phase and stage presented in this chapter has shown a systematic process of the development of both agent systems, following the proposed methodology. In the next chapter, we will discuss the achievements and the shortcomings of the proposed of agent method.

# CHAPTER 6

# 6. Evaluation and Conclusion

## 6.0 INTRODUCTION

This chapter aims to evaluate the proposed agent method. The evaluation result validates the research project's contribution to the development of an agent-oriented methodology community.

There are two elements involved in evaluating a methodology. The first is the methods used to evaluate the methodology and the second is the criteria used to justify the methodology.

There are three categories of methods used to evaluate a methodology:

- Self justification through the development of a few case studies.
- Using a pilot test, which is distributed to several people who are asked to use the methodology and evaluate it.
- Using data from a controlled experiment, supported by the case tools to prove the methodology.

The third category is the most precise method of evaluation; the second category is moderately accurate, while the first method is enough to demonstrate the viability of a methodology. The choice of the evaluation method is a benchmark of the completeness or maturity of the methodology. The third category is what is traditionally known as controlled experiment and involves two groups of matched subjects, each given a different treatment. In this case, use of the two different methodologies of our proposed methodology and a traditional approach, and effectiveness would be compared via some appropriate metric. This would involve training of the subjects and their willingness to contribute in the experiment. This often demands a considerable time of the subjects. In our case, a human factors experiment in software development is rarely feasible.

The second category, whilst not quite so time-consuming, still involves the availability of a sample of suitable subjects that needs to be trained. In view of the time available, this was also impractical.

The first category of evaluation is based on a set of criteria. The criteria represent the required features of a methodology. There are several criteria use to evaluate a methodology. For example, Martin and McClure evaluate the features of a methodology based on the following criteria: easy to read, quick to draw and change, user-friendly (easy to teach end-users) and good stepwise refinement [Mart+88]. [Chap81] focused evaluation on graphic techniques, based on the following criteria: purpose, manner of production, ease of production, ease of use and ease of maintenance. On the other hand, NIMSAD evaluated a methodology by looking at the following elements: looking at the problem solver and the problem-solving process [Jaya94].

The current AOMs can be categorised into two groups, based on how they define the methodology itself. The first group is the pioneer AOMs, published between 1995 and 1999 and the second group is the AOMs proposed between 2000 and 2003 (developed in parallel with this one).

The first group defines methodology according to [Hice+74], [Avis95] and [Chec81], very loosely. They provide a very general agent concept, limited stages of the development process, and propose techniques and modelling that are tied to the traditional methodology, mainly object-oriented and knowledge-base approaches; very little modelling notation is shown (been discussed in Appendix A).

The second group of methodologies uses a more fine-grained definition of methodology, proposed by [Jaya94][Hubm97][Graha+97] as we used in the first stage of evaluation. These two groups of methodology definition reflect different levels of maturity in representing methodologies, as we can see that they have come out with a clear definition of the agent paradigm, show the stages of the development process and provide techniques modelling and notation.

The first group of AOMs used two broad approaches to defining agent-oriented methodology. The first is adapting the existing paradigms of methodology for analysis and design, whereas the second is based on agent theory. The first approach adapts either the object-oriented or knowledge based approaches. The main problem of object-oriented approaches is that it uses object-oriented decomposition to help identify objects, not agents, which are more coarse-grained than objects. We clearly indicated in Chapter 3 that many agent features are difficult to model in purely object-oriented terms. The second approach is based on agent theory, for example MAS-CommonKADs and CoMoMAS, which of limited use for agent oriented approach.

The achievement of our research is that we are able to reduce the gap between these two approaches and come out with agent concepts as discussed in the section on the concept in

Chapter 3; these were captured through integration of several derivation processes. This agent concept is described as an agent paradigm for an agent-oriented methodology proposed, which use an extension of the object-oriented approach. Therefore it preserves several features of agents, such as distribution, openness and real time, and is able to cover the development process from analysis to design. The second approach uses a fine grained design for each agent.

We conclude that the selection of criteria is associated with the goal of the methodology. For the purpose of our research, we present three stages of evaluation: comparing with a methodology's criteria, comparing with the current AOMs and informal evaluation by potential community as shown in Figure 6.1. The evaluation was performed using the Case studies as shown in Chapter 5.

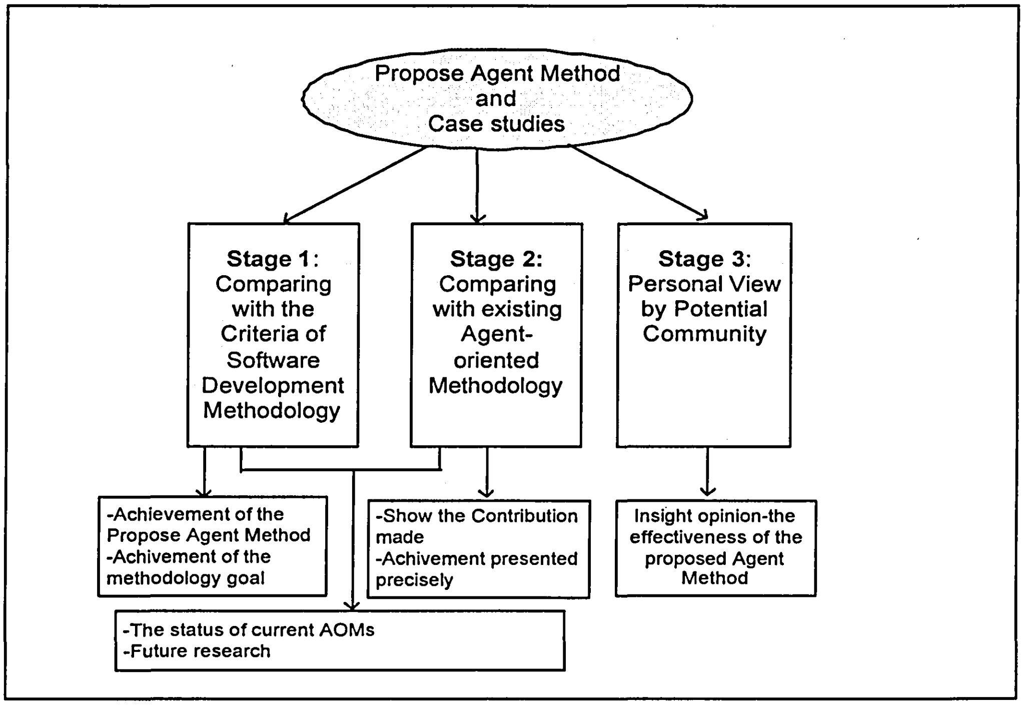

Propose Agent Method and Case studies
Stage 1: Comparing with the Criteria of Software Development Methodology
Stage 2: Comparing with existing Agent-oriented Methodology
Stage 3: Personal View by Potential Community
-Achievement of the Propose Agent Method
-Achivement of the methodology goal
-Show the Contribution made
-Achivement presented precisely
Insight opinion-the effectiveness of the proposed Agent Method
-The status of current AOMs
-Future research

Figure 6.1: Evaluation Process.

The *first* stage of evaluation is based on criteria of a methodology as defined by the first group. This method is able to present the progress we have made so far, identify whether the research has achieved the methodology goal and has fulfilled the criteria of a software development methodology. At this stage we also evaluate the techniques and notations used in the proposed methodology with the current techniques used by other AOMs. The advantage of this method is that it is possible to identify the meaning of a complete agent-oriented methodology from the Information System Methodology point of view[Chec+98].

The *second* stage compares the methodology with the second group of existing AOMs. The comparison is focused on similar kinds of AOMs, published after the year 2000. Using this method, we are able to identify the achievement that we have made more precisely and demonstrates the strengths of our methodology and the contribution made by it.
This evaluation method is able to identify the achievement of the agent-oriented methodology community as well. Thus, we can decide whether the status of an agent-oriented methodology community has been achieved so far. Consequently, we are able to determine which areas should be the focus for future research in order to achieve a complete Agent-Oriented Information System methodology.

The *third* stage is informal evaluation by two research students in the area, who use the proposed agent method and know some of the existing AOMs to be compared. This method provides considerable insight into how the effectiveness of our method was obtained. Each stage of the evaluation is presented in the following sections.

## 6.1 EVALUATION AGAINST THE CRITERIA OF A SOFTWARE DEVELOPMENT METHODOLOGY

The first method of evaluation is to identify whether the proposed methodology has presented the common criteria of software development methodology. The discussion in this section is to evaluate the proposed methodology according to the criteria of a software development methodology as defined by Jayaratna[Jaya94], Graham[Graha+97] and Hubmann[Hubm97] (refer Chapter 1 and Section 2.2). Jayaratna and Graham have similar ways of characterising a methodology.

Based on a synthesis of their work, we identify six criteria to present a methodology:

- A set of methods that illustrates the agent paradigm.
- Covers the stages of analysis, design and implementation.
- Provides techniques and notations or modelling tools in each stages of development process.
- A set of models that represent different aspects of the problem domain as well as the solution at different stages.
- Shows transformation from one model into another model.
- Provides guidelines that define an order for the systematic application of the methodology's steps.

The first of three criteria are captured from the [Jaya94] and [Graha+97], while the last three criteria are captured from the Hubmann[Hubm97]. In addition [Graha+97] defines a methodology that has a 'tailorable life cycle model' as we also focus on this criterion as well.

The first stage of our evaluation is based on the criteria of a methodology, as discussed above. This method of evaluation can provide evidence that the proposed methodology fulfilled the common criteria of a software development methodology. The descriptive evaluations are discussed below.

***The first criterion***

A methodology should provide a set of methods that illustrate the concept of a software solution. The set of methods should be able to describe the paradigm of software solution. In the context of our agent-oriented methodology, the set of methods should be able to describe the agent paradigm of the software solution. The set of methods that describes the agent paradigm is discussed in the section on agent concepts in Chapter 3 as the relationships between agent attributes: roles, tasks, actions, plan, capability, resource and event. The contribution of this research was to abstract the common attributes and their relationships that describe the agents(refer section 3.2). In a later stage this was used throughout the modelling process such as system goal, agent use case, context diagram, system plan, system architecture, agent plan, agent model. The relationships of these entities are important to represent the philosophy of the agent paradigm as shown in Figure 3.5 and is used as a pattern throughout analysis and design.

When the research started the agent paradigm was prone to individual definition. At this stage, we have come out with a clear description of agent paradigm which is appropriate for most agent solution. The agent paradigm is used in both of the development processes of the case studies. It shows a consistent use of semantics from analysis to design.

The agent design framework in Figure 3.11 shows the relationships between the entities as guidance, while designing the agent system. These agent concepts, agent overviews and agent design framework are discussed in Chapter 3. In comparison in the existing AOMs, the agent is defined very generally without showing the relationship of the attributes that present the agent.

We have identified a set of agent methods to describe agent paradigm as association of the following attributes: goal, role, capabilities, resource, communication act (interaction), action, plan, action and state. The description and relationship between those concepts illustrating the agent paradigm is shown in the agent overview in Figure 3.5. The agent abstraction (Figure 3.6) shows the attributes that describes an agent, which consists of agent name, goal, role, task, which is similar to object and abstracted by its name, attributes and methods. The role concept

is the core attribute of agent identification, in the same way as the object-oriented approach use objects to identify objects. The derivation result as discussed in Chapter 3 also identified the abstraction of role that shows the relationship between goal, capability, resource and state.
The four agent characteristics of behaviour, distribution, communication and openness were used throughout analysis and design. They were used in the refinement process to produce an optimum design, while designing the agent system at the agent system level design stage. The refinement section in Chapter 4 provides the criteria of an optimum agent system design. These four agent characteristics also describe the criteria an agent-based solution. The development of the agent system in the two case studies proved that the agent concept is applied systematically throughout the development process.

On the other hand, most methodologies just use the agent concept to capture agents rather than explaining the process of capturing the agents from the requirement. In the GAIA methodology, they propose a method to identify agents based on role, just as the object-oriented approach is based on objects. GAIA proposes the role organisation approach to extract roles from the requirements. It is suitable for problem domains in a business perspective or organisation such as the Travel Agent System previously cited as an example. This is because the role in the system is easy to identify from the actor. However, this approach was found not to be applicable in our case studies. Both case studies show several actors in the system but we find other appropriate agents, where the roles do not belong to the actor. Relying solely on actor roles is not an accurate method for identifying agents.

Capturing roles from requirements can be very confusing, similar to the problem of identifying objects from the requirement. This is because objects cannot easily be identified from the requirement. Not all objects defined from the requirements are objects for the design solution. However, the object-oriented approach provides a process for capturing the right objects for designing the solution[Rumb91] as we did to the agent.

Our approach is suitable for designing agent systems, where we cannot identify roles straight away from the user requirements but we provide the process for identifying the role. The FAS shows how activities or tasks become roles using the role concept.

***The second criterion***

A methodology should cover all stages of development process. A methodology should at least cover the analysis, design and implementation, to ensure that a consistent set of concepts has been identified that show a smooth transition between phases and stages.

The overview of the agent-oriented methodology as shown in Figure 3.7 and the refinement of the methodology shown in Figure 4.2 revealed that the proposed methodology has covered these three stages of development process from analysis to implementation. The development of the agent system as shown in the case studies in Chapter 5 has shown the development of agent system from analysis to implementation by applying the proposed agent-oriented methodology.

The analysis that compares the proposed methodology with the existing agent-oriented methodology (Appendix A-3) shows the additional stages and features of the proposed methodology. It clearly shows that the proposed methodology has several more stages in the development process. This means we have proposed a more precise development process that ensures the smooth transformation between stages.

Most of the existing AOMs focus on the agent system development phases and the methodologies also present different perspectives of the agent paradigm as we discussed in Chapter 3.

***The third criterion***

A methodology should provide modelling techniques and notation, which cover all the stages of the development process. The techniques used in the proposed methodology were initially described in the derivation results sections (Chapter 3). However, a more detailed description of the techniques was given in the modelling process (Chapter 4) such as context diagram, agent use case, event scenario, system goal, system plan, system architecture, agent model and agent plan model. The modelling notation is presented in Appendix E.

The proposed methodology consists of several levels of abstraction, phases and stages. Each stage provides appropriate modelling techniques and notations. It is structured in a similar way to the actual documentation of agent development process shown in Chapter 5. The development process is easy to follow. The case studies in Chapter 5 have demonstrated that the proposed methodology makes effective use of various techniques and modelling notations in each stage.

The analysis techniques and methods used, the analysis and design of the existing AOMs in the Chapter 3 are at a proposal stage. For example [Moul+96] uses scenarios without use case and [Kend+97] uses use case without scenarios, while [Elam+99] uses both methods to analyse the user requirements, as we did in our methodology. However, they only propose using the scenario method, rather than showing how to develop the scenario. They make assumption that using the traditional scenario is applicable. However, our research found inadequacies of the traditional scenario in conjunction the agent concept itself. The traditional scenario is a textual based description or narrative of a use episode[Wexe87]. We proposed to use event-based scenario that describes the scenario with series of activity that related to resource. Based on the activities, we were then able to capture the roles. Therefore, the requirement helps the designer in developing an agent system is to understand the agent concept itself.

We noted this problem during the reverse engineering of Newsfilter and Filtering applications to produce the Newsfilter and Filtering applications requirements respectively. We proposed to add the agent concepts while producing the scenario and use case. By this means, we were able to show a clear transition from the user requirements to scenario and later from the scenario into use case.

The research also found a prior step before producing an agent scenario. This step is to capture the events that occurs in the system. The events are captured from the user requirements. The event concept and examples were discussed in Chapter 3. This event concept was found out while comparing the use case result of the reverse engineering of Newsfilter application and Filtering with the known agent concepts of having less intervention from the user or reducing the user's workload. We discovered that the use case that represented in Figure 3.12 was not presenting the correct use case of the Filtering application. Hence, the concept of event was introduced as a prior step before producing a use case for an agent system. We propose to capture any events that occur, whether they come from inside or outside the system, that initiate the flow of the system. The case studies show examples of such events. The scenario produced is based on the events captured. Consequently, each scenario produces an essential use case that in a later stage is used to produce a concrete agent system use case.

The system goal is the stage that differentiates this proposal from the traditional approach. The system goal is a model that complements the scenario. In Chapter 4 we have discussed the modelling process to produce scenarios, use cases and system goals for agent system analysis.

Another difference in our methodology is that it defines the next stage of identifying agents as a matter of design rather than analysis, as most of the existing agent methodologies do. They assume that agents can be identified from the user requirements. This is the lack of existing AOM. Our method has proposed the process of analysis modelling on identifying agent. This method is good because it shows a process of how to identify the agent rather than capture the agent and its attributes from requirement specification, for example as shown in MESSAGE methodology.

### *The fourth criterion*

A methodology should consist of a set of models that presents different aspects of the problem domain as well as the solution at different stages. This criterion means that the methodology is able to present various aspect of modelling domain in each stage. This criterion is presented in the proposed methodology. In the analysis stage, for instance we have shown various sets of models to show different aspects of agent analysis. The system goal, use case, event context diagram and scenario show different aspect of analysis. Similarly, in agent system level design, we provide the dynamic and static interaction modelling that presents different aspects of design. The agent system architecture (Figure 4.9) represents the static agent interaction, while the plan sequence diagram portrayed (Figure 4.10) represents the dynamic system interaction. In addition, the modelling of each interaction as shown in Figure 536 presented the autonomous interactions between two agents.

In the agent environment development phase, we identified five sets of models used for analysis and design. Appendix B shows examples of various types of modelling which at this stage adopted the UML notation. The description of the agent environment development phase presented in section 4.2 shows three stages of the development process where the modelling refers to the Appendix E. We do not provide other case studies to show agent environment development, but rather emulate the JADE environment development using the proposed development process as shown in Chapter 5.

At the agent system deployment stage we also present various sets of modelling. However, we only presented the agent class diagram in the methodology because it is able to show the integration of agents with the components. Because at this stage we are using object level design and applied UML notation, therefore object-oriented set models are applied at this stage, such as object based collaboration diagram and sequence diagram.

### *The fifth criterion*

A methodology should show the transformation process from one model into another model. The proposed agent method has met this criterion. The agent system artefact as shown in Figure 4.2 presents the activities in each stage and it shows the transformation from a model at each stage into another model in the next stage (the arrow shows the relationships). For example, the agent use case model is developed based on the scenario model and requirement specification; however the agent identification technique is based on identification of role, which is captured from the use case and system goal. The figure also shows the transformation between stages.

For example the model of the agent system architecture is developed from agent use case when the roles are assigned to agents. The agent is identified using agent use case, agent paradigm (capabilities) and system goal as shown by the arrow. A similar process applies to the modelling in other stages as well.

***The sixth criterion***

A methodology should provide a set of guidelines that define the systematic application of the methodology's steps [Hubm97]. The overall guideline of the proposed methodology described in the introduction section of Chapter 4, while the procedural guidelines for the techniques and modelling processes are discussed in relation to each activity of each stage. For example the use case modelling process is described in the section on requirements analysis for the use case activity process. Presentation of the agent system development process in Chapter 5 refers to the technique and modelling process stated in the proposed methodology in each stage, rather than just presenting the documentation of the agent system development.

As an additional point we note that Graham [Graha+97] stated that a methodology has a 'tailorable life cycle model' that consists of actions (techniques), process (life cycle) and representation(notation). Whilst the methodology criteria proposed by Graham is similar to those of Jayaratna and Hubmann, as he focuses on the issue of life-cycle model for development process. The advantage of our proposed methodology is that it presents three separations of concern in the development process into three phases that can be described as a spiral model in each of the three following phases: agent system development, agent environment development and agent system deployment. The first two phases can be developed in parallel by utilising the reusable elements of software development. Moreover, they can be tailored to suit the needs of a particular system development in that the two parallel phases can be intertwined as depending on the ease with which the agent system development or its environment can proceed.

Furthermore, this life-cycle presents the separation of the software development process, which is good for large and complex systems because a group of people with appropriate expertise can focus on each phase of development process. This method can reduce the development cost by reducing the time of development and increase accuracy. The flow of the life-cycle development process depends on the requirements specification provided.

The capability to identify common requirements for agent environment development makes it possible to perform the development process in parallel. Each phase shows the sub-life cycle development process as a spiral model as well. The agent development life-cycle is clearly shown in the overview of the proposed methodology in Figure 3.7 and a detailed illustration is given shows in Figure 4.1.

At this stage, we cannot compare with the first group of existing AOMs because they are not at the same level of achievement. In the next section we are going to discuss more precisely the contribution we have made while evaluating the proposed methodology in relation to with the AOMs developed in parallel with our research.

## 6.2 COMPARISON WITH THE CURRENT AGENT-ORIENTED METHODOLOGIES

This stage of evaluation was performed based on the three elements of software development methodology as discussed in the Chapter 3: *agent paradigm, life cycle of development process, techniques modelling and notation.* At this stage, the evaluation is performed more precisely because most of the current AOMs have similar achievement to our proposed methodology in the following respects:

- A clear description of the agent concept by defining the agent attributes or properties and the relationships to describe the agent paradigm compared with the methodologies that were proposed when the research started.
- Coverage of most of the software development process of analysis, design and implementation.
- Provision of at least with one case study to show the applicability of the methodology by applying the proposed techniques, modelling and notation.

In general, we can make a statement that one methods are similar in the above respects but also different in many aspects, as we discuss in detail through out this chapter. The evaluation is conducted by comparing the methodologies according to the three elements of a methodology as stated above. The analysis is able to identify the similarity and differences between those methodologies and highlight the strengths in each agent-oriented methodology. At the later stage, we are able to discuss the challenges that face to development of matured agent-oriented methodology.

The comparative evaluation analysis is conducted with four AOMs. Even though more then 10 AOMs have been proposed, we chose only four of them, based on following criteria:

- They are the most popular AOMs that have been produced by a group of agent researchers.

- Provide detailed documentation of the methodologies, and the documents were accessible.
- They provide case studies that allow us to evaluate the good features.
- They cover the development stages from analysis to implementation.

Some agent methodologies focus on certain aspects of agent features, such as security[Secur+02], interactions[Ricci+02] or intelligence [Carver+00], but do not show the whole process of agent development. However, they help towards the development of a complete agent-oriented methodology.

The four AOMs are: 'ODAC: An agent-oriented methodology Based on Open Distribution Application Construction [Gerva02], TROPOS: The TROPOS Software Development Methodology: Processes, Models and Diagrams[Giun+02], MESSAGE: Methodology for engineering system of Software Agent[MESS00] and PROMETHEUS: A Methodology for Developing Intelligent Agents [Padg+02].

The comparison process is based on three elements of software development methodology for agents:

- Agent paradigm,
- Development process
- Techniques modelling and notation.

Each comparison is discussed in the following sub-sections. Besides that, an evaluation of the proposed methodology was performed as part of the case studies shown in Chapter 5. It was conducted in three phases of development process: analysis, design and implementation.

### *6.2.1. Agent Paradigm*

The major difference in these methodologies is the way they present the agent paradigm. This is because the term agent itself is multi-dimensional and various kinds of agent properties are used to define the agent paradigm, as discussed in Chapter 2, section 2.1.1. This is the main challenge to developing a general and mature agent-oriented methodology that can be applied in various domains.

The current AOMs can be categorised into four groups. They show their way of presenting the agent paradigm:

- autonomous and BDI agent [Giun+02][Padg+02],

- autonomous and reactive [Wool+00][Gerva02],
- autonomous agent(active object)[Elam+98][Burm+96] and
- autonomous and decision making[Buss+00].

All of them are agreed that agents should at least have autonomous behaviour, in addition to other properties. However, the comparison was only conducted for the first two groups, which we called BDI agent and reactive agent. This is because the last two groups do not fulfil the evaluation criteria as mentioned before. Our proposed agent method has similar strands to the autonomous and reactive group, which means it is similar to MESSAGE and ODAC methodology based on GAIA methodology, while PROMETHEUS and TROPOS apply the BDI agent paradigm. The second group claims that providing BDI preserves the intelligence criterion for agents with belief, desire and intention properties. However, we argue that it is enough to have the autonomous and reactive behaviour to present the agent paradigm. For us, belief, desire and intention behaviours are additional properties to autonomous and reactive behaviour, which can be added as appropriate in particular cases. There are various techniques and strategies to provide agent intelligence, which are not confined to belief, desire and intention. For example, the case studies showed that the pro-active behaviour provided in DCS is added based on these basic autonomous and reactive behaviours. Similarly in the FAS, adaptive behaviour was also based on these basic behaviours. The strength of the autonomous and reactive agent paradigm approach is the fact that it is of a more general purpose. More significantly for us autonomous and reactive behaviour is sufficient to preserve the four agent characteristics of behaviour, distribution, communication and openness as discussed in Section 2.4.

These two agent paradigms influence different ways of thinking about the analysis and design of agent systems. We believe that the use of the BDI agent paradigm approach is limited to cases where the requirement specification indicates belief, desire and intention properties, whatever its advantages of showing agent intelligence. Indeed, the requirement specifications of the case studies as shown in Chapter 5 were shown to be unsuitable for a solution using the BDI paradigm rather than as autonomy and reactive. For us the BDI is an optional behaviour. It can be added, based on the basis properties of autonomy and reactivity.

Due to the differences of agent paradigm between our agent method and the PROMETHEUS and TROPOS methodologies, the detailed comparison of the agent concepts was performed by comparing with MESSAGE and ODAC. ODAC methodology is an extension of the design stage of GAIA methodology. The agent concept of GAIA methodology describes the relationship between agent, goal, role, services, permission, responsibility and activity to present the autonomous and reactive behaviour. However, we describe the agent concepts according to the following attributes: role, goal, capabilities, roles, resource, plan, action, event, communication act and direct communication. Our description of the agent concept is quite similar to the MESSAGE agent overview, but it is different in defining the term attributes.

MESSAGE defines the role concept similarly to GAIA methodology. Role as defined in the GAIA methodology as a similar concept to the object-oriented, which defines roles as types of component that inherits as a type of object. For example there are two types of role, RoleA and RoleB. ObjectC may be RoleA or RoleB. This concept is applied to agents and therefore an agent may presents it self as a customer or seller role. As we stated in the methodology we agree with this role concept, which we applied in the FAS, where the UserAgent may be an *enquirer* or *feedbacker*. These are two different roles assigned to the same agent.

The main MESSAGE concept is based on knowledge rather than data. Knowledge level is described based on *concrete entity, activity* and *mental state entity*. The concrete entity describes the agent, organisation, role and knowledge. Activity is based on task and interaction, while *mental state entity* is based on goal. MESSAGE has proposed an agent modelling language that built upon these concepts.

For us, the core agent concept is based on the relationships between agents, roles and resources as shown in Figure 3.5 as grey attributes. Roles are relationships between goal, capabilities, resource and state. The agent design framework illustrates the relationship between a multi-agent system with agent, goals, roles and tasks to develop the organisation of the agent system based on a plan and the system goal. The attributes of the plan are captured as relationships between agents' goals, roles, actions and communication.

Even though PROMETHEUS and TROPOS are based on the BDI agent paradigm, they have a similar concept of role, goal, plan, action and interaction as well. The belief, desire and intention focus on representing knowledge, which is associated with defining role.

The analysis and design process in our methodology is based on those agent concepts. We discuss this in more detail in at a later stage, when we perform the comparative evaluation of the third element of a methodology, the technique, modelling and notation stage.

In the next section we focus on the comparative evaluation of the second element of a methodology, the life cycle of the development process.

### *6.2.2. Development process*

In general, all the methodologies had covered the common three stages of software development process. The problem of performing the comparative analysis is that they are different in the way they defined the meaning of those stages; especially at the analysis and design stage.

Most of the methodologies do not mentioned the approach of the life cycle of development process except MESSAGE that follows the spiral model similar to the object-oriented development process as it preserves the development process of the waterfall model [Royce70]. We assume that TROPOS, ODAC and PROMETHEUS also preserve the waterfall model since they show their development processes as a sequence of the following stages: analysis, design and implementation stages but no mention is made of any life-cycle model being adapted. The methodologies are different in presenting the stages of development process and define the meaning of each stage.

As discussed in the previous section, we classified the agent paradigm into two groups or schools of thought on viewing agents. These two groups differ in the way they define the stages in the development process of analysis and design:

- The first group is the group of autonomous and reactive methodologies, i.e., MESSAGE, ODAC.
- The second group is the group of autonomous and BDI agent based methodology, i.e. PROMETHEUS and TROPOS.

Figure 6.2 shows the comparison of the development process in the methodologies. The figure shows that MESSAGE and ODAC methodology follow a similar trend to traditional object-oriented development process, which consists of analysis, design and implementation. This group defines architecture design as part of design stage similarly to the object-oriented approach; we discussed in Chapter 2 the importance of skill in architecture design and the use of architecture patterns as guidance in developing the architecture design.

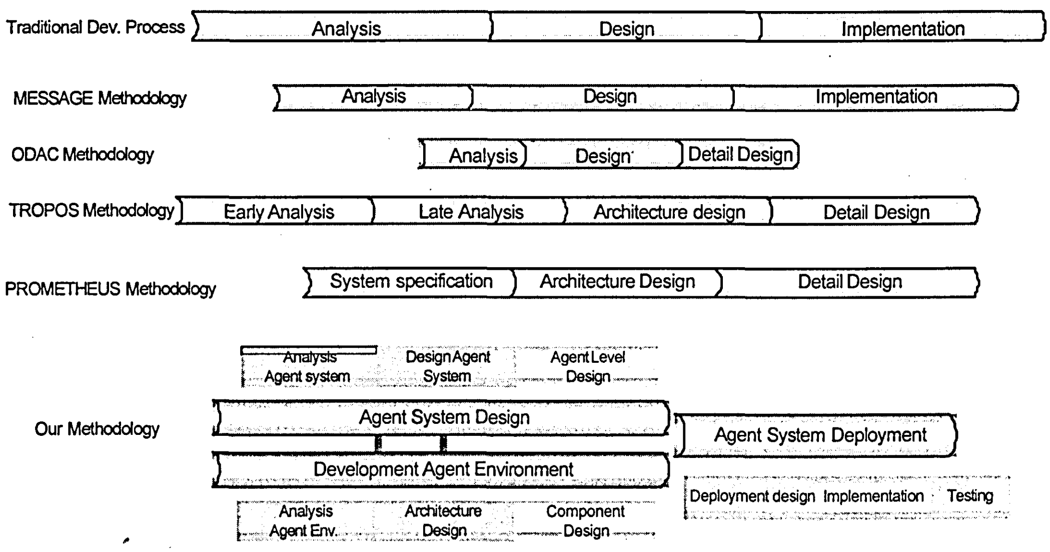


Figure 6.2: Comparison of the Stages of Agent Development Process

In ODAC methodology, the architectural design is located at the design stage, which is decomposed into two stages: design and detail design. ODAC adapted the GAIA methodology for analysis stage. Therefore the first stage for ODAC methodology is the requirement specification for the design, which focuses on developing a methodology that presents the criteria of distribution and openness. This design stage is similar in this respect to our agent environment phases, which focuses on the four agent characteristics. In addition, ODAC proposed three behaviour specifications that influence the architectural design.

The second group defines architecture design as the organisation of agents, illustrated in term of agents and their interactions. According to MESSAGE, this term of 'architecture design' is defined as the analysis stage. This group of methodology has eliminated the term of 'architectural design' that defined by MESSAGE and ODAC as design stage in their development process. They defined the 'architectural design' as implementation stage, as both of them declare JACK as an agent language and they use it at implementation stage. This method describes the limitation of agent methodology (specific type of agent). The first group presents a more general approach that suits to agent-oriented development methodology, while the second group is limited to a specific architecture, i.e. BDI agent architecture.

The first group applies a contingency software development approach, while the second group applies a refinement approach. The refinement approach is similar to a top down approach, which refines the analysis model into a design model, whilst the contingency approach is an integration process consisting of synthesising several model views at the analysis and design stage as shows the modelling at analysis level 1 in MESSAGE. The contingency approach is similar to an object-oriented approach, which provides various types of modelling for analysis and design. The problem with this approach is that it is difficult to show a clear transformation process from analysis to design stage, as the object-oriented approach had found [Kain99]. However, at agent development environment we applied this method.

Utilising the concept of software abstraction, our methodology presents a combination of both approaches, as we have separated the analysis and design stages into two phases: *agent system development* and *agent environment development.* These two phases consist of analysis and design stages, but the methods, techniques and approaches for analysis and design in these two phases are different. In general, Figure 6.2 shows the differences between our development process and other methodologies, either first or second schools of thought.

The *agent system development* phase applies the refinement software development approach, while agent environment development applies the contingency approach. We also defined the third phase as agent system deployment rather than implementation, because we defined another pre-implementation stage, called the *deployment design* stage.

The comparative analysis of the development process is evaluated in more detail using Table 6.1. The table shows the comparison of the phases and stages of our development process with others' methodologies. In general, we are similar to the first group of thinking that does not eliminate the architectural design in the development process, as we preserve the criterion of a general-purpose agent-oriented methodology.

The evaluation of the development process is performed based on the phases and stages of our methodologies (the first column in the table). The first phase focuses on agent system development, specifically the 'what' and 'how' of agent use for the solution of the problem domain. The table shows that our agent system the development process consists of *analysis requirement, agent system level design* and *individual agent design* stages. This phase is similar in pattern to the whole development process of TROPOS and PROMETHEUS methodologies.

| | Methodologies | Stages Name |
|---|---|---|
| Agent System Development | MESSAGE | Analysis Level 0 |
| | ODAC | Use GAIA Analysis |
| | TROPOS | Late analysis |
| | PROMETHEUS | System Specification |
| | Our Methodology | Analysis Requirement |
| | MESSAGE | Analysis Level 1 |
| | ODAC | Use GAIA macro level |
| | TROPOS | Architectural design |
| | PROMETHEUS | Architectural design |
| | Our Methodology | Agent System Level Design |
| | MESSAGE | Design |
| | ODAC | Use Gaia design |
| | TROPOS | Detail design |
| | PROMETHEUS | Detail design |
| | Our Methodology | Individual Agent Design |
| Agent Environment Development | MESSAGE | Analysis Level 0/ Level 1 |
| | ODAC | Analysis |
| | TROPOS | - |
| | PROMETHEUS | - |
| | Our Methodology | Agent Environment Analysis |
| | MESSAGE | Design Low level |
| | ODAC | Design |
| | TROPOS | - |
| | PROMETHEUS | - |
| | Our Methodology | Agent Environment Architectural Design |
| | MESSAGE | Design Low level |
| | ODAC | Design |
| | TROPOS | - |
| | PROMETHEUS | - |
| | Our Methodology | Component Design |
| Agent System Deployment | MESSAGE | - |
| | ODAC | - |
| | TROPOS | - |
| | PROMETHEUS | - |
| | Our Methodology | Deployment Design -UML modelling |
| | MESSAGE | Implementation |
| | ODAC | Implementation |
| | TROPOS | Implementation |
| | PROMETHEUS | Implementation |
| | Our Methododogy | Implementation |
| | MESSAGE | Not mentioned |
| | ODAC | Not mentioned |
| | TROPOS | Not mentioned |
| | PROMETHEUS | Not mentioned |
| | Our Methododogy | Testing |

Table 6.1: Comparison (stages) between the Propose of Our Agent Method with the Current AOMs

The *first* evaluation is according to the requirement analysis stage. In our approach, this stage aims to analyse the user requirement to produce a concrete specification of system requirements. The requirement analysis process consists of an elicitation and analysis process that provides techniques and a modelling process to develop a concrete requirement specification. It is similar to the analysis stage of TROPOS methodology, which comprises early stage analysis and late analysis, and similar to system specification in PROMETHEUS methodology. Indeed, PROMETHEUS adapted the TROPOS early analysis stage to produce the system specification. The MESSAGE methodology defines this stage as analysis level 0. The main strength of the MESSAGE is the fact that they proposed an agent-oriented analysis modelling languages, so at this stage the requirement specification is captured into the analysis modelling language, that consists of five views of analysis modelling such as organisation view, goal view, agent view, interaction view and domain view, which is good to represent a systematic modelling.

The *second* stage is to compare the methodologies at the ASLD stage. In our approach, this stage aims to identify agents and their interactions. This aim is similar to the analysis level 1 of the MESSAGE methodology, which refines the analysis level 0 into level 1 as organisation diagram, delegation structure diagram, interaction diagram, domain information diagram. At this stage, our approach is similar to the second group of methodologies, where at this stage is defined as a design stage rather than an analysis stage. Therefore, this stage is similar to the 'architectural design' of PROMETHEUS and TROPOS methodologies.

The *third* stage is to compare the individual agent design stage. Here our aim is to provide the internal design of each individual agent as a continuous process from the previous design stage. Therefore, the modelling process focuses on internal agent design. The MESSAGE methodology defines this stage as a design stage, while the TROPOS and PROMETHEUS define it as a detail design stage that is closer to our methodology.

The next evaluation is based on the agent environment development phase (Chapter 4), which consists of *agent environment analysis, agent environment architectural design* and *component design*. The aim of this phase is to develop architecture that defines the environment for agents to perform their roles through socialisation among them, as captured in the previous phase. Therefore, in general, the aim of this phase is similar to the low level design of the MESSAGE methodology. PROMETHEUS and TROPOS do not consider this phase as part of their development process.

The first stage is to compare with our *agent environment analysis* stage, which aims to analyse the requirement for development of the agent environment. At this stage, the analysis process includes identifying the multi-agent system specification and identifying appropriate technology for presenting the multi-agent system criteria as four agent system characteristics: present the basic agent behaviour, distribution, communication/socialisation and openness.

This stage is similar to what in the MESSAGE methodology is part of analysis level 0 and level 1, which consists of the five view of analysis but do not separated them into two phases of analysis as proposed in our methodology. ODAC defines this stage an analysis stage. The strength of ODAC is that they had proposed five types of requirement at analysis stage: enterprise, information, technology, engineering and computation. It also focuses on providing the environment with distribution and openness criteria. The methodology is a similar strand to our agent environment analysis, which focused on three types functional requirements: agent architecture, communication environment and physical environment(platform). The identification of each functional requirement is influenced by enterprise and technology factors.

The second stage is comparing with agent environment architectural design and components design. MESSAGE defines these two stages as low level design and ODAC defines them as the detail design stage.

The third phase of our methodology is agent system deployment. The aim is similar to the implementation stage, which is to refine each individual agent design. However, we define a pre-implementation stage, called the deployment design stage. Our third phase consists of three stages: deployment design, implementation and testing. At this stage, the agent design is at a fine-grained design level, as it can transfer to the current implementation language such as Java.

The next stage of the evaluation process corresponds to the third criteria of a methodology such as techniques, modelling and notation. However, the basis of evaluation is according to the stages of the methodology as discussed in this section.

### *6.2.3. Modelling Techniques and Notations*

At this stage, the comparison process is difficult to perform because comparison needs to be at a precise level, while there are many differences among the methodologies. As stated in Chapter 3, one of our research aims is to focus on the use of software abstraction at the analysis and design stages to show a clear transformation processes from one stage to another. Therefore the modelling techniques and notations presented in our methodology are evaluated against this.

The different concept of agent has influenced the use of modelling techniques and notations based on the stages. In terms of defining the agent paradigm, we are similar to MESSAGE but we are different in the methodology goal. We used different techniques and modelling processes in the analysis and design stages. MESSAGE uses a contingency approach from analysis to design. Its provides five views of analysis modelling, which are transformed into design modelling. Its shows a consistent semantic modelling process by proposing a modelling language, represented in diagrammatic notation. This is a clearly a strength of MESSAGE methodology. At present our methodology has not reached this stage, but rather is stated for future work as discussed in later section.

The advantage of modelling language is that it defines all the attributes that illustrate the agent concept and the analysis shows the relationships among the attributes at two levels of analysis, which covers most of the agent attributes for analysis and presents a schematic language for analysis and design. Analysis level 0 captured the attributes from requirement while analysis level 1 builds relationships from the attributes from following views: organisation, agent, goal, interaction and domain into following diagrams: organisation, agent, delegation structure, interaction and domain, which mainly build from the following attributes: role, knowledge, goal, agents, and task. We have this modelling but located in several stages, since the aim of our methodology is to show a smooth transformation process and we have disagreement on defining analysis and design stage. For example MESSAGE identify agent at analysis stage, while we defines agents as design stage similar to TROPOS and PROMETHEUS.

A weakness of the MESSAGE methodology is the failure to provide the elicitation process to produce the concrete requirement specification. MESSAGE makes the assumption that the requirement specification provides enough information for agent system analysis. Therefore, in the analysis process, information is extracted from the requirement specification into the analysis modelling using the modelling language as what object-oriented do with UML. The weakness of this approach is the difficulty of capturing the attributes for analysis modelling from the requirements. For example, what does the requirement define as goal, resources, role, tasks. The process of capturing those elements is not straightforward unless the requirement specification is represented in such way that it easy to identify those elements. For example, the requirement specification is produced through the requirement engineering process, which consists of elicitation and analysis process.

The strength of our methodology is that we provided an elicitation and analysis process to produce a concrete requirement analysis that is not tied with any agent paradigm concept at the earlier stage but rather capture the agent attributes in the analysis process. Figure 5.2 shows the flow of process and activities in each stage and the type of modelling appropriate in each stage.

Therefore, the modelling process in the requirement analysis itself is a recursive process. In our methodology, this part of analysis focuses on agent system development while some part of analysis is performed at the agent environment analysis in agent environment development phase.

The TROPOS provides an elicitation process at the early stage of analysis, but the major difference is that TROPOS focuses on BDI agent. Therefore, it uses the '*i** ' technique at the early stage of analysis process, which is tied to the goal-oriented analysis process. In addition, TROPOS has invented various kinds of modelling for BDI agent and invented new concepts and notation for BDI agent-oriented methodology. This is the strength of TROPOS. The designer needs some effort or training to understand the terminologies and modelling process.

However, the strength of PROMETHEUS is that it utilises the current techniques and modelling, modified by the addition of agent features such as the use of use case and scenario, which is similar to our approach. However, PROMETHEUS does not provide early stage of analysis but rather uses the TROPOS's early stage analysis to produce a requirement specification. In this respect PROMETHEUS is similar to MESSAGE.

Our approach is similar to PROMETHEUS, which utilises the current techniques. The comparative analysis of technique and modelling is performed according to the most similar techniques and modelling used in those methodologies.

The case studies shows that the development process utilises current techniques and modelling with modification to fit the agent paradigm and shows the refinement process of analysis to design, from top level to lower level design. The comparison of the techniques and modelling is discussed as the following.

*Context Diagram.*

At high level analysis, we utilised the context diagram that is used in the structured method, but we changed the context diagram concept to show the flows of the events from outside the boundary of the system with the resource, rather than the flow of function or data. This imitates basic agent architecture [Rusel+95]. The inside of the context diagram is a decomposition of agents and roles linked with events and resources. None of the other methodologies provides this modelling. This modelling is useful to show the boundary of the system at high level design.

*System goal.*

Another model is the system goal which aims to show the agency criteria presented in the problem domain. Most of the methodologies provide this modelling but they have a different perspective of how to develop the system goal.

This is the hardest stage to build. In Chapter 5 we stated that the development of system goal needs a rationale to choosing the right technique to build the system goal, whether based on an organisation, task, or goal perspective. It is justified based on the requirements itself. GAIA methodology uses an organisation perspective and TROPOS uses a goal-oriented perspective. These two perspectives are not suitable for the system goal in our case studies, which did not present either organisation or goal features in the requirements. In this case our approach is more like to MESSAGE. Our methodology defines the term of goal not only provides with mental state of achieving goal defines in more general term that present a 'level of objective'. For example, in the DCS, the PatientAgent's goal is to receive an efficient treatment and the DoctorAgent's goal is to provide effective treatment thereby reducing consultation. We use a task perspective and build the system as a similar process of defining the hierarchy tasks to achieve objective of system, but it is different because the hierarchy is built based on a hierarchy of tasks that influence the achievement of the system goal. The system goal of the DCS shows that the sub-goals represent the agency of the system, which in this case is to reduce consultation and increase efficiency. Sub-goals are prioritised accordingly so additional sub-goals can be added to increase the level of agency of the system goal without changing the goal of the system. For example, in order to increase efficiency, we could change the requirement for patient, so that instead of using a computer to make an enquiry and set an appointment, the patient uses a mobile phone as the user interface. So, an additional sub-goal is added under the sub-goal, but this does not change any additional agent in the system. We proposed the modelling and notation for system goals as shown in the case studies. The TROPOS defines goals that consists of mental states and has plans to change the agent's mental state, as it is able to represent an aspect of intelligence.

*Scenario.*

Another technique is to use scenarios based on the events that captured in the context diagram. Our concept of scenario is not similar to the traditional scenario, which describes the series of activities in natural language description. Most of the methodology not modelled the scenario accept PROMETHEUS. PROMETHEUS provides a template of scenario that describes the series of roles, tasks associated with resource. The template is presented in a table similar to the scenario modelling proposed in our methodology. However, PROMETHEUS adds features of BDI role in the scenario template. The main difference of our scenario is that we are able to show the process how the scenario is built from event context diagram. It shows the flow of activities when an event occurs, which is able to capture the incoming and outcoming events in each activity. Each activity is related to resources. By combining of these elements, we can capture the agent role. The scenario modelling and notation are shown in the case studies. The DAS shows two scenarios based on the two events coming from patient and doctor. In PROMETHEUS, the scenario is built from the requirement specification by making assumptions that requirement specification provides a concrete requirement that enabled to capture the elements of incoming events, message, activity and actions.

*Use case.*

PROMETHEUS transfers each scenario into a use case and shows the relationship of resources or knowledge among the use cases as a similar strand to our methodology. The Diabetic Consultation case study shows how the scenarios in the Table 5.7 and 5.8 were transferred into use case Figure 5.31 and 5.32 respectively. However, we are different from PROMETHEUS because we aim not only to show this transform action but also to eliminate the redundancy of activities in those use cases. The agent system use case is developed as a combination of the events use cases. At this stage it should be able possible to identify the concrete analysis of requirements, which are presented as the requirement specification.. The linkage between activities in the use case is needed to conform that at this stage the use case has covered all the requirements.

*Agent System Architecture.*

This modelling aims to show the agent and interactions. All methodologies presented this element but they are different in the notation used and at different stages. MESSAGE uses the agent system architecture as system organisation in the analysis stage. The modelling shows agents and interactions, including roles, knowledge, task, etc. PROMETHEUS and TROPOS define this modelling as architectural design. PROMETHEUS architectural design shows the relationship of those elements: agent, use case, data, message, protocol, action, event, plan and message together. They propose a notation for each element.

In terms of modelling, PROMETHEUS shows a complex architectural design, which brings together all the agent elements in the diagram. We separate it into two models. The agent system architecture shows the agents, interactions, goals, outside events (from context diagram) and data, while the roles, events, actions and relationships are defined in general in agent abstraction as shown in Figure 4.8.

*System Plan.*

Another additional feature of our methodology at this stage is the provision of various types of modelling at the agent system level design stage to show the system plan adapting the UML sequence diagram and activity diagram. Various advantages of presenting the sequence diagram and activity diagram are discussed in the proposed methodology. In addition it uses an intermediate modelling to show a clear transformation to the next stage of design.

The next stage is individual agent design, which as we stated before, is similar to the detail design of PROMETHEUS and TROPOS methodologies. PROMETHEUS does not provide any modelling at this stage, but uses plan-based agents, using a library of user-defined plans. This plan-based can be mapped directly to implementation construct using any BDI architecture framework such as DMARS[DMARS98], JAM[JAM00] and JACK[JACK00], but it is stated that the case study was implemented in JACK.

The strength of our methodology is the fact that we provide techniques, modelling and notation at this design stage such as Agent Model and Agent Plan, but it focused on the plan of action to be performed rather provided a specific strategic plan, defined plan.

*Agent interaction.*

Another type of modelling is interaction between agents that apply to certain complex interaction protocols. For example, the DCS makes use of autonomous interaction of Contract-net protocol. The modelling shows how the 'request-when' occurs in the DCS. However, for modelling of agent interaction we adapted the AUML [Odel+01b][AUML00]. This modelling is good when the case study shows an interaction protocol, which is not similar to any existing interaction protocol. Most of methodologies applied the AUML to present the agent interactions modelling.

*Agent Model.*

The agent model is a refinement of agent abstraction, which describes the goal and the role of the agent and shows the incoming and out-coming information from internal and external agents including the parameters that describe the content language and protocol interaction are used.

The agent model of each agent, as shown in the case studies, uses the modelling notation of the agent model. It also describes how extracting information from the previous models creates the agent model.

*Agent Plan Model.*

The agent plan model provides the internal plan modelling for each agent. It shows the plan of how agents capture events from communication with other agents, external events or internal events and shows what action should be performed, whether as internal communication or interaction with another agents or external actor. The agent plan shows which task belongs to each role. At this stage, the definition of tasks is finer-grained, such as intelligence for each agent. The Case study shows how learning behaviour is provided to UserModel Agent and how Fuzzy Logic is provided to ExpertAgent.

The next discussion is concerned with the modelling at agent environment development phase. Most of the methodologies did not focus on the internal agent design, for example MESSAGE make the assumption that the internal design follow the object-oriented design and TROPOS and PROMETHEUS are defined as implementation stage using JACK. We are able to show the integration of agent level design into object level design. .

*Modelling at the Agent Environment Phase.*

At this stage, comparative analysis can be performed between our methodology and MESSAGE and ODAC only because only these two consider this phase as part of the development process. In general, we make a judgement that all of them propose techniques and modelling at the proposal stage without showing any case studies to provide this phase. MESSAGE defines this stage as a design stage, which refined the five analysis view models for developing the architectural design. MESSAGE also stated the needs of choosing architecture in at design stage similar to the agent environment architectural design. It shows the example of design modelling in a UML notation by extracting the design modelling of the JADE platform.

The strength of the ODAC methodology is that it focuses on this phase. It proposes three behaviours of requirement influence on development architectural design as computation, enterprise and technology[Gerva02]. It proposes the target of environment of execution must be comprise these three behaviours of specifications. ODAC uses the open distribution processing (ODP) engineering viewpoint [Musc+00] as ongoing research in this area.

In this phase, our methodology adapted the object-oriented framework application approach for development agent environment, which uses the software architecture development process for development of the architectural design. Therefore, we propose that development of agent environment consists of three stages: analysis agent environment, agent environment architectural design and component design.

Agent environment analysis is the process of producing a requirement specification for agent environment development. We abstracted three types of functional specifications, which influence the identification of architectural design: agent architecture, communication environment and physical environment, captured from the enterprise environment and technology environment. The enterprise environment identifies the basic functionality for agent environment development to satisfy the agent characteristics: behaviour, distribution, openness and communication. The three types of functional specifications influence the choice of technology used. The strength of our methodology is that we have proposed the common requirements for an agent environment that provides a choice of the current technology as a reference architectural design, and provides reference for architectural patterns and design patterns as guidance to design and choose the technology. The hardest stage in development of an agent environment is to produce the best architectural design since the quality of development is based on experience. The document of the scrutiny process in Appendix C provides is guidance on the various architectural designs that provide certain functionality.

We also imitate the agent environment development process by showing the JADE environment development process as shown in Chapter 5 consists of three stages:

- JADE agent environment analysis.
- JADE environment architectural design and
- Components design, which some of modelling shown in Chapter 4 and Appendix C.

The strength of MESSAGE is that it has proposed the integration of five views of design modelling that enable to show how agent environment architectural design is build.

This phase is similar to ODAC that use ADL and component design as part of developing architectural design, while MESSAGE applied object-oriented approach. However, our level of achievement is similar to the other methodologies as we are not able to show applicability of the agent environment development process in a case study development of agent environment. However, our proposed methodology applies the reuse component techniques in development of agent environment that the case studies show in Chapter 5 preserved the proposed methodology.

Both case studies show that the requirements of agent environment are presented in three types of functional specification as shown in section 4.1.4 and 4.2.3. Utilising reused components, the case studies shows the process of reusing the agent environment. The first case study shows the analysis process to reuse an existing agent environment, while the second case study show the integration of reused existing components for agent environment development.

*Agent Deployment Model.*

The advantage of our methodology is that we are able to show the link between the agent level design and object level design. At this stage, the object-oriented concept is applied. The agent attributes such as role, goal, task, interaction, event, action and resources as modelled in the previous stage are designed at object level.

It is a fine-grained level of individual agent design. In addition it shows integration with the components as defined in the agent environment. At this stage, object-oriented modelling such as class diagram, collaboration diagram, and state diagram can be applied. The case studies in Chapter 5 show the transformation from individual design into deployment design as we show the integration using class diagrams. The class diagram is able to show the integration with the reused components. The proposed methodology shows part of the implementation and suggests how to test the agent system.

## 6.3 POTENTIAL COMMUNITY EVALUATION

The proposed agent method is evaluated by two PhD research students (development agent system), who know the MESSAGE methodology and the proposed agent method. The first is Saedeh Maleki who applied the methodology for agent development in Adaptive Information Retrieval. The second is Saadat Al-Hashmi, who applied it in the development of Supply-Chain Management System.

The evaluation was conducted by comparing the proposed of agent method with MESSAGE methodology according to five criteria: description of agent concept, coverage of method, transaction stage, guidance to use the methodology and modelling notation. The evaluation results are shown in Table 6.3 and 6.4. Each criterion is merits according three variables: good, moderate and poor.

| Methodology criteria | MESSAGE | Our agent method |
|---|---|---|
| Description of agent concept | Clearly defines the agent concepts (autonomous reactive agent) | Provides a room of various other agent concept may applied. A clear agent concept describes as a type of agent(autonomous reactive) |
| Coverage of method | the methodology specific to an example provided | more general for various problem domain |
| Transaction between stages | clearly shows the transactions (but less in analysis to design) | separate process into ASD and AED. Clear transaction analysis and design in each phases |
| Modelling Notation | Good modelling notation | Less modelling notation |

Table 6.2: Afsaneh Personal Evaluation

| Methodology criteria | MESSAGE | Our agent method |
|---|---|---|
| Description of agent concept | Clearly defines the agent concepts, but very specific type of agent | Defines agent concept clearly, accepts various kind of agents |
| Coverage of method | the methodology focuses on example given | shows applicable for different domain of agent system |
| Transaction between stages | clearly shows the transactions process between stages | Clear transaction analysis and design in ASD phases |
| Modelling Notation | Provide a diagrammatical analysis modelling language | Less modelling notation |

Table 6.3: Saadat Personal Evaluation

Afsaneh concludes by saying,

> *'I appreciate Othman's agent system development process that guides me to development system in a structured way and shows the milestones of my development '*

Again she said

> *'I agree with Othman's proposed methodology which provides a separate way of thinking while analysis the system into design agent system and analysis agent environment as I did when developing agent system'*,

and Saadat said

> *'As someone from a non-computing background I appreciate Othman's proposed methodology. It gave me an idea for identifying agents and their*

*interactions, it also helped me to transform the system requirement into design modelling, which immensely guided me in implementation aspects'*

## 6.4 ANALYSIS OF EVALUATION RESULT

In conclusion, we can say that the proposed methodology has presented the common criteria of software development methodology because we are able to describe the agent concepts for the agent paradigm and show the life-cycle of development process including the milestones and deliverables. The methodology presents a smooth development process between stages from analysis to design and implementation and we have proposed various appropriate techniques and modelling including the notation as well. Beside that, the development process of the case studies presented in Chapter 5 demonstrated the applicability of the proposed methodology.

Comparison with the AOMs discussed in Chapter 3, it clearly that we have made a significant contribution in the area of agent-oriented methodology. In general, we can say that we have proposed a new approach to the software development process pertinent to the nature of agent development itself. It is distinctive from the traditional object-oriented methodology and adapts all the benefits of a traditional software development approach by employing abstraction and reuse elements into the agent arena. Abstraction provides a separation of concerns in the development process at the various stages of requirement, analysis and design. Moreover, the proposed development process preserves the traditional spiral model as we discussed in detailed in Chapter 5. The advantage of this method is that it can separate the designer's thinking at requirement analysis and design into two separate areas: the 'what' and 'how' of agent solution, and 'where' and 'how' agents are performed, which were represented as the agent system development and agent environment phases. Each phase consists of requirement, analysis and design phases, but employs different philosophical design concepts. The decision on an agent solution at the stage of identifying the agent paradigm influences the decision of where the agents performed and what technology is used for this purpose, as we discussed in the introduction section of Chapter 5.

The agent system development is the phase in which the agent concept is applied, and the focused of the research. Agent environment development is mainly based on the components development approach that we used to represent agent concept uses the object-oriented approach. The research focused on identifying common agent environment requirements through scrutiny of existing agent environments. The agent system deployment is a refinement of the agent system development phase. At this stage, the agent concepts applied in the agent system development phase are presented at an object level and the deployment process uses the object-oriented concept.

The development of two case studies presented in Chapter 5 shows that they follow the development process. When analysing the requirement specification, the process of identifying agent solution, and identifying the components and technology used for agent solution are performed in parallel. However, in terms of documentation as presented in Chapter 5, they are presented as sequences of three phases of the development process.

Due to the research time constraint, we were not able to present case studies that show the development process of a bespoke agent environment. The process of agent environment development presented in Chapter 5 is at the proposal stage. It imitates the software architecture approach for the architecture design process and imitates the object-oriented application framework approach for agent environment development as we thought this is an appropriate approach for agent environment development. Therefore, the research focused on identifying the common requirement for agent environment development to develop a generic agent environment requirement that can be integrated with patterns.

Although in both case studies, reused components were employed, the Diabetics Consultation case study showed that no existing agent environment matched the requirements. It reused a part of the agent environment component and integrated it with the other components needed for executing the DCS. In development of the agent environment for the DCS, the JADE environment was defined as a component, which could integrate with the mobile phone system environment, email component and fuzzy rule component.

Referring back to the early stage of our research, the traditional method of agent development is in an ad hoc fashion as we see in the NewsFilter presented in [Bigus+97] and our first attempt at developing a Filtering application, which consists of thousands of lines of implementation code that are difficult to understand and maintain. We took a long time to identify the requirements and understand the flow of the Newsfilter application itself, while we reverse engineered the system. The development of the case study of the FAS presented in Chapter 5 shows how easy it is to understand the system. It shows the delivered document in each stage that is refined from a high level design into a lower level design.

The discussion throughout this thesis has shown that we have proposed a new life-cycle of the agent development process. Moreover in terms of techniques and modelling process, we introduce new techniques of modelling which utilise existing models by changing part of the concept in conjunction with the agent concept itself such as scenario, context diagram and use case. This research found that the use case modelling is useful in analysis process to produce a concrete scenario of requirement specification.

In this respect we follow the same approach as previous researches, by proving the usefulness of the existing methods and techniques for agent based systems. We also introduce new techniques and modelling notation appropriate to modelling on agent system, e.g. agent system architecture, system goal and agent model..

Referring back to our methodology goals stated in Chapter 2 they were to produce a systematic process, provide coverage of the development process including the milestones and deliverables and to provide general purposes methodology. The first two of these goals have been discussed above. We now turn the third criterion, which to propose a general purpose of an agent methodology that corresponds to the common philosophy of the agent paradigm. This is because the analysis of existing AOMs revealed that the proposed methodologies are geared to a specific agent paradigm such as active object[Elam+99], BDI agent[Kend+95][Kend+97] and autonomous and reactive[Wool+00]. Ours does not focus on specific techniques or problem solutions. General purpose means that the methodology is not confined to a specific agent type of solution or specific area of application.

This criterion is achieved by introducing a stage called *identifying agent paradigm,* located between requirement analysis and the agent system level design stage to identify the right agent paradigm solution.

The agent overview presented in this thesis contains the minimum entities needed to represent agents, and the basic entities needed to analyse agent systems. Other additional entities may be needed for a specific type of agent. The use of the role concept presents a consistent method to identify an agent abstraction. Other entities such as agent behaviour, distribution, communication and openness are attributes of agent abstraction that should be considered during the development process. Therefore, we proposed a recursive process of several layers of agent abstraction.

Even though the methodology presents the whole life-cycle of agent development, the modelling process presented in this thesis is focused on the *agent system development,* in accordance with the boundaries set for this research as stated in Chapter 3. Beside that we also focused on transformation modelling from *agent system development* to *agent system deployment phase.*

This is because both phases present a similar concept of design, which focuses on what system to develop and how.

The agent environment development phase is an additional boundary of the research to show the actual life-cycle of development agent system. As the result of scrutiny of agent frameworks the development process is not a new approach, but utilises the existing methods and techniques in the development of an object-oriented framework application. The life-cycle of agent environment development represented in this thesis is imitated from the software architecture development process to produce agent environment architecture. Therefore, our research is concerned to meet the criteria of an agent environment, produce a set of agent environment requirements appropriate for agent environment, identify the common domain and entities that can be reused in association with the appropriate existing technology in agent system development. The nature of agent environment development is suitable for use of patterns as a guideline on the development process. The derivation process in Chapter 4 shows several architecture patterns and agent design patterns used in the development of existing agent framework, while in Chapter 5 we compile the appropriate design patterns for development agent environment according to the research in those areas.

At the *agent deployment stage*, we show the modelling process of for deployment. However, the modelling process is very limited to a particular agent environment (JADE).

The comparative analysis in section 6.2 also shows that our proposed methodology represents a similar level of achievement to the existing of AOM. However, Akhgar in his thesis [Akhg03] has a similar argument with [Jaya94] that IS methodology should not only consist of the criteria as presented in section 6.1 but also has following elements:

1. Tools: this refers to computer based applications to support the methodology (e.g. case tools)
2. Strategy: provides rationale in analysis and design aspect. So, the methodology considers wider aspects of Information System.

Most of the proposed AOMs have fulfilled the methodology criteria stated in section 6.1, but none of them has covered this two criteria. Most of the methodologies are at the stage of proposing modelling notation but not yet providing case tools.

Similarly to the second criterion, which provides guidance to be rational while in analysis and design. The rational is able to guide the designer to be rational in designing rather than follow the steps. Information System methodology should not only tell the designers what steps to perform and how to perform the steps, but also indicate why those steps should be taken in a particular order, and if they are not followed the order, what the consequence will be. The rationale also provides alternative solutions rather than very specific ones confined to particular methods or techniques, and indicate in what situation a particular technique is more useful. In this way, the methodologies are able to cover wider aspects of the Information System.

Based on the discussion throughout the comparative analysis with existing AOMs (section 6.2,), we can conclude that our methodology achievement is at a similar stage to the other AOMs. All these methodologies show significant progress in analysis and design modelling, compared to the AOMs discussed in the early stage of research, section 3.2, which are only at proposal stage.

Having discussed the achievement of our methodology and others as well, we can see the contribution we have made in the area of development of agent-oriented methodology. In the next section we want to see whether our methodology is a methodology from the information system point of view.

In order to see the contribution made, we break down those criteria into five itemised criteria, which we use to compare the methodologies and show the strengths of each methodology. In addition we can identify the area of future research for producing mature agent-oriented methodology. Table 6.4 shows the result of comparison of the methodologies according to those itemised criteria. The number of notation '+' in the table indicates the level of strength of a criterion presented in each methodology.

| Agent-Oriented Methodology | Development Process | | | | | Goal | Underlying Philosophy |
|---|---|---|---|---|---|---|---|
| | Life cycle model | Cover-age | Model-ling | Trans Formation | Tools | | |
| MESSAGE | Spiral model | +++ | +++ | ++ | Agent Modelling Language | Systematic Software Engineering | Reactive |
| ODAC | Spiral model | ++ | ++ | + | Description and notation | Agent Environment | Reactive |
| Our Methodology | Double Spiral Model | ++++ | ++ | +++ | Modelling Notation | Systematic process | Reactive |
| TROPOS | Waterfall | +++ | +++ | +++ | Modelling notation | Intelligence | BDI |
| PRO METHEUS | Waterfall | ++ | ++ | +++ | Case tools | Systematic process | BDI |

Table 6.4: Comparison Result among the Methodology according Information System Perspective.

The table shows that the methodologies are different in most of the criteria. Each methodology has its own strength. The strengths of our methodology is that it provides the most coverage of the development process and provide options, which allows the designers to be rational with analysis and design. For example, we propose a stage of identifying agent paradigm between analysis and design stage for making a decision of on the appropriate agent paradigm for the problem domain. However, we still do not explain exactly how to make a rational choice of particular a paradigm, but rather use analysis of the type of behaviour provided in the requirement specification as we plan for future research. In addition, we do not focus on a specific method to develop system goal, but rather provide choice to the designer to be rational in choosing the most appropriate approach.

In summary, the strength of our proposed agent method is that we separate the development process concerned into three phases, each of which has different aims and methods for analysis and design. The methodology provides an alternative development method for agent environment development by utilising the reuse criterion.

Based on these three phases, the strength of our methodology is that it has proposed the life-cycle of development agent system as a double spiral model, which covers all the methodologies as well. The first spiral is a recursive process among the phases and the second spiral is in each phase, which consists of three stages that develop in a spiral model as well. Based on our three phases we can see the contribution of the methodologies to the three of agent system development phases.

Among the reactive agent group, we have provided an analysis stage as a pre-analysis stage to that in the MESSAGE methodology. We provide elicitation analysis to produce the requirement specification rather than assume the requirement specification includes agent-based elements. The requirement specification is developed through a series of modelling processed from high level analysis to lower level analysis using tools such as event context diagram, system goal, scenario and then use case.

At agent system level design, we propose three types of modelling to present the agent system architecture as static and dynamic models. These models are useful to show a clear transition process from agent system level design into individual agent design. Moreover, these models are not used only to show agents and interactions but also used in the refinement process as well, as we discussed in section 4.1.4.

Furthermore, our methodology has provided several types of design modelling at the individual agent design level, which shows the transformation into finer-grain modelling. In addition we propose an agent system deployment phase to show the transformation process between design and implementation, whilst most other methodologies focus on design to implementation.

An additional advantage of our methodology is that we use architecture patterns and design patterns in the development process and provide a collection of architecture patterns and design patterns for development of a bespoke agent environment. Besides that, the methodology also proposes the development process for reuse of an existing agent environment. An alternative of reuse development process is proposed by [Eiter+02]. The discussion above highlights the strengths of our methodology that are not provided in other methodologies. However, we also discuss the strengths of other methodologies as well.

As we discuss in previous section, ODAC's contribution is in the agent environment phase but it is still at the proposal stage and is limited on the issue of developing the agent environment such as distribution and openness. MESSAGE is in a similar strand with us, but has a problem in showing a clear transition process between the analysis models and design model. The strength of MESSAGE is that it provides agent modelling language for agent system analysis, which consists of five types of analysis modelling, but it does not yet provide agent modelling language for design.

The strength of TROPOS is that it provides a great contribution in modelling agent intelligence, i.e. BDI agent, which covers in fine detail of the stages for agent system development from early stage analysis, to design and implementation. However, it eliminates the agent environment architectural design. The TROPOS methodology is very specific in applying a BDI agent solution.

PROMETHEUS similar to TROPOS and the strength of PROMETHEUS is that it provides a case tool for modelling and uses the notation of methodology for analysis and design and transforms the modelling into implementation in JACK language.

This methodology is useful for the analysis and design and implementation of BDI agents, which do not really show the features of intelligence, as in TROPOS.

The evaluation by the two Phd research students indicates presented that we have provided significant contribution and strengths compared to other methodologies, even thought the comparison is limited to MESSAGE methodology. However, their statements confirm the strength of our methodology compared to others. Both of them adapted the proposed agent system in the documentation of the development of their agent systems.

## 6.5 DISCUSSION

Producing a mature methodology is neither an easy nor a quick process. Figure 2.1 shows the past history of the development of the traditional methodology. The figure shows that structured methodology takes about 20 years to produce a matured Information System methodology, while the object-oriented approach which was proposed in the mid 80s started to mature after 1995. Indeed, completeness of the methodology in specific areas such as architectural design, design pattern, development framework and components is still an ongoing process. The initial idea of agent as an approach for system development started in 1995 and the achievement of agent-oriented methodology development so far is very impressive. However, in order to produce complete agent-oriented information system methodology there is still a long way to go.

There are several factors that instigate difficulty in development of a complete agent-oriented Information System methodology. *The first* is the dissimilarity on viewing agent and dissimilarity in defining the coverage of agent development process, which leads to disagreement on techniques and modelling as we discussed in a previous section.

Because of the differences in context and objectives in different stages and aspects of modelling technique and process, it is not surprising that differing agent abstractions have developed. So the challenge in agent software engineering is to acquire greater abilities to reason strategically, especially about the agent system itself, and its relation with the world, whether agent as software, agent as the world, agent as actor or agent as inter-mediator and other specific agent-related issues. Moreover, several issues may arise in relation to the agent concept; this requires designers to be creative in overcoming additional problems while designing agent systems. It seems that achievement of a complete agent-oriented methodology is still far off, but our concern was to build the foundation of agent-oriented methodology, which look at different perspective with traditional software development by utilising the separation of concerned in software development. Another issue on specific technique would most likely adopt other methodologies such as knowledge based or object-oriented methodology.

Referring back to the two groups of agent-oriented methodology as stated earlier they present two perspectives of agent-oriented methodology: BDI agent and reactive agent. These two groups have different views of agent-oriented methodology. The former is focused on a case tool approach. It views agent-oriented methodology as focused on analysis and agent level design, while the implementation uses case tools. This approach is very close to a specific agent area and tends to produce a precise methodology. The latter approach uses the traditional approach, which is an extension of the object-oriented approach. Therefore, at the detail design stage it mainly adopts the UML, whereas the implementation stage uses object-oriented language using any OO tools such as Rational Rose[Rati97]. The agent-oriented methodology needs software engineering for the *agent system development* phase and *agent environment development* phase. Therefore agent modelling is focused at these two phases.

However, the evaluation result presented in this chapter provides a basic foundation for future research, which would aim to combine the strengths of each methodology towards development of a complete agent-oriented Information System methodology.

*The second* problem faced by some agent systems development is that the difficulty to execute in the real world [Mamd98], because so many issues need to be considered before execution of the system such as trust, confidentiality, security and performance, which are still ongoing issues. One indicator of a matured methodology is looking at how many systems succeed being implemented and executed, using the methodology proposed. We believe development of a complete Agent-oriented Information System has a long way to go.

Referring back to our two case studies, the FAS is easy to execute and test, because it depends less on critical issues such as trust and confidentiality, while the DCS involved various issues to be considered before applied it in the real world.

Due to the limitations of time, human resources and expertise in certain areas, we were unable to produce a complete agent-oriented methodology, which covers the whole life-cycle of agent development. However, we have achieved a similar level of progress to other methodologies that are likely to influence on-going initiatives in this area. The other methodologies in Table 6.2 are sponsored and involve many researchers and experts.

The limitation for us was lack of experience in the development of an agent environment. Rather we abstracted this process from scrutiny of the existing agent frameworks, for incorporation in the modelling design. Consequently, there are several aspects of modelling that are still not covered yet. However, agent environment reference architecture shown in Figure 3.12 is useful guide to analysis agent environment, while the architecture patterns and design patterns presented in Chapter 4 serve as guidance on designing the agent environment. The modelling for the agent environment proposed here is still only a proposal.

Beside that, there are several strengths in TROPOS, PROMETHEUS and MESSAGE as stated before that useful to be gathered with our proposed methodology to produce a matured agent oriented methodology as we leave it as our next future research.

## 6.6 FUTURE RESEARCH

Based on the discussion in previous sections, the strength of our methodology is that we provide more stages of the development process, which is divided into phases and stages. Besides that, we were able to come out with the life-cycle of agent development, which matches the stages of the development process proposed by other methodologies, as we showed in Table 6.1. Therefore, we are convinced that the development of agent-oriented information system methodology (AOIS) should consist of the three phases of the development process being proposed which mainly applied to others methodologies as well. In MESSAGE the design stage stated the need for detailed analysis which generate use case, refine use case, generate sequence diagram and refine sequence diagram of object oriented approach. This is part of our analysis agent environment and selecting the architecture is part of architectural design process in our agent environment development phase. The layer architecture for agent is a pattern we proposed in our methodology. The three phases also applies to PROMETHEUS and TROPOS as they focused on agent system development phase and eliminates the agent environment phase as JACK language. In addition, the ODAC methodology have similar agreement with us, which the methodology is focuses on development agent environment.

Our aim for future agent-oriented methodology is to have a clear process of agent development, which provides agent modelling language at each phase: *agent development modelling language, agent architectural modelling language* and *agent deployment language* respectively. The identification of these three stages of modelling languages can then be applied using case tools. The end result using this modelling is suitable for implementation using any object-oriented tools.

***Towards development of an Agent Development Modelling Language.***

Our research result shows a method and guideline of the agent development system throughout the three stages as a preliminary stage towards the development of agent modelling languages and modelling process. The attributes and relationships of the agent concept and modelling notation presented in this thesis represent s foundation towards development of agent development modelling languages. The research result shows that we have been able to show the modelling process of the first phase as shown in Appendix E, but the second phase of modelling is at the proposal stage.

In order to achieve the development of *agent development modelling language*, the next research stage is to arrive at agreement on the terms in the agent concept, such as role, goal, resource, action, event, BDI and their relationship towards agent. Comparative research between the requirement analysis of our methodology, the analysis stage of GAIA and early stage and late analysis stage of TROPOS was able to identify the feature of analysis for the reactive and BDI agent. Such a result can be used to identify the strategy for analysis of problem domains using an appropriate agent paradigm as we proposed in the methodology as identifying agent paradigm.

Another dimension of research is in the area of analysis of agent patterns. For example Reactive agent pattern, BDI agent pattern and Adaptive agent pattern to overcome the problem of various agent behaviours and provide guidance to designers on choosing the right solution. Beside that, based on our experience, we also discovered patterns for agent system design. For example, a common agent is identified in a context. For example in what context do we need an interface agent? and what elements are needed to provide an interface agent? The second example is when two agents use similar resources by means of updating or accessing, which causes conflict as to are which is the latest or right information to update the resource. In this case, we need an 'inter-mediator' agent to manage such conflict. Agent design patterns provide a strategy for design just as design patterns do for the object-oriented approach.

***Towards development of an Agent Architectural Modelling Language.***

The research on Architecture Descriptive Language (ADL) and design patterns as discussed in Chapter 2 are promising approaches towards the development of architecture modelling language. Both areas are still ongoing towards development of mature ADL, similarly in design pattern as well. More development of agent frameworks with specific architecture will make it possible to identify the ADL for development of the agent environment and later to produce a mature architectural modelling language. Having maturity of the Agent ADL research community, our aim in future would be to develop case tools for the agent environment development, so the designer can choose the appropriate components to be integrated to develop an agent environment.

In this research, we have identified agent *environment reference architecture* as a preliminary attribute towards defining the architecture modelling language. However, the *agent environment reference architecture* defined here was limited to scrutiny of a few agent frameworks. Further research focusing on wider scope of the agent environment.development by scrutiny of more agent frameworks in wide area of case studies would enable us to identify more concrete *agent environment reference architecture.*

Detailed comparative analysis between ODAC methodology, our agent environment development, design level 1 of MESSAGE and research in ADL made it possible to identify an architectural modelling language and at the same time produce a clear development process for agent development. Object-oriented architecture and design patterns are useful at this stage. The development process of agent environment is parallel to the research of ADL [Faul+03] [Lixin+00]. Therefore there are several ongoing studies in this area. Now ODAC has started this research by developing the agent ADL for agent mobile architecture [Gerva+01]. Other research in similar areas is stated in [MADL03].

***Towards development of an Agent Deployment Modelling Language***

The modelling at this stage aims to focus on the interrelation between *agent development modelling language* and *architecture modelling language*. The identification of a common integration. process in both modelling languages would make it possible to identify the agent deployment language. Even though, the *agent system deployment* uses the object-oriented concept, but the object-oriented approach does not clarify the agent attributes and needs modelling for the integration process.

Even though the agent deployment modelling shown in this thesis is very focused on a specific agent environment (JADE) as shown in the case studies, it is at a preliminary stage of showing transformation from agent based design level to object level design. Future research could produce an *agent deployment Modelling language.*

Having modelling language in each phase, the next research stage would be to produce case tools in each phase. The PROMETHEUS has produced the case tools for the agent system development phase but focused on BDI agent. In future research to identify AADL and development of open components in build the architecture, the case tools for agent environment development could be developed. [Lixin+00] has started the research in this area; even though they not focused on agent environment development their work provides the basis for using

ADL for development of AADL. We believe at this stage, designers can choose appropriate components for developing their own agent environment architecture.

A later stage would be to produce case tools for agent deployment, as the Rational Rose [Ratio97] does for the implementation of object-oriented design.

Before going to that stage of research, there is still much research that needs to be done at this stage such as producing a standard in modelling for each phase. The current research issue now is how to integrate these methodologies to come out with a standard agent definition, techniques, modelling and notation for agent development.

The modelling process in this thesis is imitated from the two case studies, but there may be other issues that need to be taken into consideration. Even though socialisation or interaction among agents is one of the major issues in agent system development, our research has not focused on this area but only used it in the methodology, for example, at the analysis stage, an interaction protocol is used to analyse the user requirement, to identify the roles in the system.

There are other issues that need to be addressed, such as modelling of agent specific behaviour, specific task modelling and intelligent modelling, specific type of coordination model and specific type of interaction protocol that will rise up through specific research in particular area. Therefore we believe that designers should be more creative in addressing a specific particular area, which is uncovered. The aim of agent-oriented methodology is to provide a foundation for development of agent-based systems from analysis to implementation that are applicable to most agent concepts rather than a specific area.

Another important area of agent system development is patterns, because of the lack of entities used to abstract agents, reduce complexity and duplicated effort, and the lack of a common vocabulary such as agent patterns, communication patterns, travelling patterns, task of patterns, interaction patterns and coordination patterns. The research on patterns is parallel with ADL as well.

## 6.7 CONCLUSION

Based on the above discussion we conclude that the research has made a significant contribution towards development of an agent-oriented methodology. We have

- *Proposed a new life cycle for future Agent-oriented Information Systems.*

- *Provided a general- purpose methodology.*
- *Made novel contribution by showing two case studies of agent system development.*
- *Identified the four agent characteristics as basic criteria for agent-oriented solution and used them throughout the development agent system.*
- *Compiled the agent concepts and produced an agent overview and agent design framework.*
- *Modified the existing modelling techniques in accordance with agent concepts, i.e. scenario, use case, agent system architecture, internal agent plan and agent deployment modelling*
- *Proposed a developed process for development of bespoke agent environments, including reference architecture, identifying characteristic of agent environment and identifying appropriate technology to develop agents.*
- *Proposed a development process of reusing off-the-shelf agent environments and components.*
- *Provided a collection of design patterns associated with the attributes of developing agent environments.*

We conclude that our proposed methodology is not yet identified as a methodology but rather is a software development process. It is at a similar stage to other AOMs that are still ongoing processes of producing a complete agent-oriented methodology. None of the methodologies are yet defined as a methodology in the Information System methodology perspective. However, the comparison discussion in previous sections show that the achievement of each methodology is complimentary to each other and some of the term are ambiguous.

The discussion throughout this thesis, including the comparison with the existing AOMs has provided clear insight of what mature agent-oriented methodology should be. This can guide future research. The strength of our methodology is that we show the whole process of agent development, which applies in most of other methodologies as well. We believe at this stage, we have shown a systematic process for development of agent systems, which consists of several levels of abstraction in a top-down approach as the result of our desire to have a rational systematic way of designing an agent system. The integration of several activities of derivation process as discussed in Chapter 3 and 4 influenced us to be rational while designing the agent system. We believe that the research has provided a significant contribution towards the development of agent-oriented methodology. Indeed, [Parn+86] said that

*' we will never find a process that allows us to design software in a perfectly rational way. '*

We do not and cannot say that the agent community developed agents in a rational way. Indeed they may and should decide on what stages of our methods are helpful in the development of agent system. We would, however strongly advise that they do document it in some systematic framework akin to ours. The merit of this is that it has a documented development, rational and accessible to others.

[Aars+96] Aarsten, A., Brugali, D., Menga, G., (1996), Patterns for Co-operation, *The Third Conference on Pattern Languages of Programs(PLoP'96),* Illinois USA.

[Adel90] Adeli H., (1990), *Knowledge Engineering Fundamentals Vol. 1* and *Vol. 2*, McGraw Hill College.

[AIP99] Extending UML for the Specification of Agent Interaction Protocol, *OMG Document*, ad/99-12-03.

[Akhg03] Akhgar, B., (2003), *Design and Development of a Methodology for Strategic Information System*, PHD Sheffield Hallam University, Thesis, to be appeared.

[Alan88] Alan H.B., Gasser, L., (1988), *An Analysis of Problems and Research in Distributed Artificial Intelligent*, In Alan, H. Bond, and Less Gasser, editor, pages 3-36, Morgan Kaufmann, Publisher, San Mateo.

[Amun02] Amund, T., (2002), JFipa an Architecture for Agent-based Grid Computing(2002), *Proceedings of the Symposium of AI and Grid Computing*, AISB Convenion.

[App+94] Appleby, S. & Steward, S., (1994), Mobile Software Agents for Control in Telecommunication Networks, *BT Technological Journal* 12 (2), pp. 104-113, April.

[Arid+98] Aridor, A. and Lange Danny, B., (1998), Agent Design Patterns: Elements of Agent Application Design, *Proceeding of Autonomous Agents,* ACM Press.

[Arno+98] Arnold, K. and Gosling, J., (1998), *The Java Programming Language*, Addison-Wesley Publishing Co., 2nd., edition.

[Ash+90] Caroline Ashworth and Mike Goodland, (1990), *SSADM: A Practical Approach*, Published McGraw-Hill, 1990.

[Avis+95] Avison,D.,E. and Fitzgerald,G., (1995), *Information System Development Methodologies, Techniques and Tools*, 2nd, edn. MacGraw-Hill, New York.

[Bacl92] Baclace P.,E., (1992), Competitive Agents for Information Filtering, *Communications of the ACM*, December 1992/Vol.35, No.12, pp.50.

[Bake96] Baker, A. D.(1996), Chapter 8-Metaphor or Reality: A Case Study where agents bids with actual costs to schedule a factory, Market-based Control, Scott, H., *Clearwater (ed.), World Scientific.*

[Barc+96] Barbuceanu, M., Fox, M.S., (1996), Capturing and Modelling Coordination Knowledge for Multi-Agent System, *Journal on Intelligent and Cooperative Information System.*

[Bato+89] Batory, D.S., Barnett, J.R., Twichell, B.C., Garza,J., (1989), Construction of File Management System from Software Components, *Proceedings of COMPSAC*, 1989.

[Baum+97] Baumer, D., Riehle, Gryczan, G., Knoll, R. & Züllighoven,H., (1997), Framework development for large system, *Communication of the ACM,* Theme Issue on Object oriented Application Frameworks, Mohamed E. Fayad and Douglas Schmidt, Editors, 40(10), Oct., 1997.

[Baum+97b] Baumann, J., Hohl, F., and Radouniklis, N., (1997), Communication Concepts for Mobile Agent Systems, *Mobile Agents - First International Workshop, MA '97* (Berlin, Germany, April 7-8, 1997), Published as Kurth Rothermel and Radu Popescu-Zeletin, editors, LNCS 1219, Springer, 1997.

[Baum+97c] Baumann, J., Hohl, F., Rothermel K., and. Strasser, M., (1997), Mole - Concepts of a Mobile Agent System, *Technical Report 1997/15,* Faculty Information, University of Stuttgart, Aug. 1997.

[Baum+97d]Baumann, J. and. Radouniklis, N., (1997), Agent Groups in Mobile Agent Systems, *Distributed Applications and Interoperable Systems (DAIS'97)*, pages 74-85, H. Koenig, K. Geihs, and T. Preuss, editors. Chapman & Hall, 1997.

[Baum+98] Baumann, J. and Rothermel, K. (1998), The Shadow Approach: An Orphan Detection Protocol for Mobile Agents. *Mobile Agents - Second International Workshop, MA '98* (Stuttgart, Germany,. Published as K. Rothermel and F. Hohl, editors, *Lecture Notes in Computer Science*, 1477:2-13., Springer, Sep. 1998.

[Bell+98] Bellay, B., Gall Harald, (1998), Reverse Engineering and Describe a System's Architecture, In Development and Evolution of Software Architecture for Product Families, *2$^{nd}$. International ESPRIT Workshop,* Spain, LEC 1429, Springer-Verlag.

[Bell+99] Bellifemine, F., Rimassa, G. and Poggi, A., (1999), JADE - A FIPA-Compliant Agent Framework. In: *Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-agents*, UK.

[Bigus+97] Bigus, J.P. & Bigus, J. (1997), *Construction Intelligent Agents with Java*, John Wiley & Sons, USA.

[Boeh+77] Boehm, B.W., Brown, J.R. and Lipow, M. (1977), Quantitative Evaluation of Software Quality, Software Phenomenology- Working paper of the *Software Lifecycle Management Workshop.*

[Boeh88] Boehm, B.W., (1988), A Spiral Model of Software Development and Enhancement, *IEEE Computer*, 21(5), pg.61-72.

[Bond+88] Bond, A.H. and Gasser, L.(1988), *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, San Mateo.

[Booc94] Booch, G.,(1994), *Object Oriented Analysis and Design with Applications*, The Benjamin Company. Inc.

[Booc94b] Booch,G., (1994), *Designing an Application Framework*, Dr. Dobb's Journal 19, no 2, February 1994.

[Booc98] Booch, G., Rumbaugh, J., Jacobson, I., (1998), *The Unified Modelling Language User Guide*, Addison Wesley. Canada.

[Bosc00] Bosch, J., (2000), *Design and use of Software Architecture, Adopting and Evolving a Product-line Approach*, Addison-Wesley, UK.

[Bosc99] Bosch. H. (1999), Measurement system framework, *In Domain-specific Application frameworks.* M. Fayad and R. Johnson editor, New York: John Wiley & Sons.

[Brad97a] Bradshaw, J.M., (1997), An Introduction to Software Agents. In: *Software Agents*, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press,, pages 3-46.

[Brads97b] Bradshaw, J.M., Dutfield, S., Benoit, P. and Woolley, J.D., (1997), KAoS: Toward An Industrial-Strength Open Agent Architecture. In: *Software Agents*, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press.

[Bratm+88] Bratman, M.E., Isreal, D.J. and Pollack, M.E. (1988), Plans and Resource-Bounded Practical Reasoning, *Computional Intelligence*, 349-355.

[Bren+98] Brenner, W, Zarnekow, R., Wittig, H., (1998), *Intelligent Software Agent, Foundation and Application*, Springer Verlag, Germany.

[Breu+94] Breuker,J., Velde,W.V., (1994), *CommonKADs Library for Expertise Modelling, Reusable Problem Solving Components*, IOS Press, Amsterdam.

[Broo86] Brooks, R.A., (1986), A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, Vol. 2., pp.14-23.

[Bube80] Bubenko,J.A., (1980), Information Modelling in the Context of System Development, In *Information Processing 80'*, pg. 395-411, North-Holland.

[Burm96] Burmeister, B., (1996), Models and Methodology for Agent-oriented Analysis and Design, in K.Fisher(ed), *Working notes of KI'96 workshop of agent oriented programming and Distributed system*, , DFKI Document D-96-06.

[Burm98] Burmeister,B, Bussmann,S., Haddadi, A., and Sundermeyer, K.,(1998), Agent-Oriented Techniques for Traffic and Manufacturing Applications: Progress Report, In Jennings., M.R. and Wooldridge, M.J, , *Agent Technology foundation, Application and Markets,* Springer-Verlag, Berlin, New York.

[Busch+96] Buschman,F., Meunier,R., Rohnerts,H., Sommerlad,P., Stal,.M., (1996), *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley.

[Buss+00] Bussman, S., Jenning, N.,R., and Wooldridge, M., (2000), On Identification of Agent in the Design of Production control system, Agent-Oriented Software Engineering, AOSE2000, LNCS 1957, Springer-Verlag, Germany.

[Carver+00] C. A. Carver, J. M. D. Hill, J. R. Surdu, and U. W. Pooch, (2000), A Methodology for using Intelligent Agents to provide Automated Intrusion Response, in *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, West Point, NY, June 6-7.

[Cash00] Mark Cashman, (2000), An agent based architecture for a distributed diabetes advisory system, MSc Thesis, School of Computing and Management Sciences, Sheffield Hallam University.

[Caste00] Cristiano Castelfranchi, (2000), Engineering Social Order, *Engineering Societies in the Agent World*, Springer, pp 1-18.

[Chap81] Chapin, (1981), Graphic Tools in the Design of Information System, eds. Cotterman, W. Couger. D., Enger, N. and Harold, F., *System Analysis and Design: A Foundation for the 1980s,* North Holland.

[Chau97] Deepika Chauhan, (1997), *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*, ECECS Department Thesis, University of Cincinnati, Cincinnati,

[Chav+96] Chavez, A., and Maes, P, (1996), Kasbah: An Agent Marketplace for Buying and Selling Goods, *In proceedings of the 1st International Conference on the Practical Application of Intelligent Agent and Multi-Agent Technology, (PAAM96')*, UK, London.

[Chec+98] Checkland, P., & Holwell, S., (1998), *Information System and Information System Making Sense of the Field*, John Wiley & Sons.

[Chec81] Checkland, P, (1981), *Systems Thingking*, Systems Practice, John Wiley & Sons.

[Chec90] Checkland, P., Scholes, J. (1990), *Soft Systems Methodology in Action*, John Wiley & Sons.

[Chen+92] Chen, W. and Warren, D.S., (1992), A Goal-Oriented Approach to Computing Well-Founded Semantics, *Proc. 1992 Intl. Conf. on Logic Programming* (ed. K.R. Apt), MIT Press.

[Chik+90] Chikofsky, E. J., & Cross, J. H., (1990), Reverse Engineering and Design Recovery: A Taxonomy, *IEEE Software*, 7, 13-17.

[Cock+96] Cockburn, D., & Jennings, N.R., (1996), ARCHON, A Distributed Artificial Intelligent System for Industrial Applications, in G.M. P. O'hare and N.R., Jenning editors, *Foundation for Distributed Artificial Intelligence*, John Wiley & Sons.

[Cohen+94] Cohen, P.R., Cheyer, A., Wang, M., and Baeg, S.C., (1994), An Open Agent Architecture, in *Proceedings of the AAAI, Spring Symposium.*

[Cons97] Constantine, L., (1997), The Case for Essential Use Cases, *Object Magazine,* SIGS Publications, NY.

[Cook00] Cook, S., (2000), Architecture Standards, Processes and Patterns for Enterprise Systems, in Barroca, L., Hall, J., and Hall, P., *Software Architecture Advances and Applications*, Springer Verlag, UK.

[Copl+95] Coplien, J. O., & Schmidt, D. C. (Eds.), *Pattern languages of program design*, Reading, MA, Addison-Wesley..

[Copl92] Coplien, J.O., (1992), *Advanced C++ Progamming Styles and Idioms*, Addison Wesley.

[Cunn+87] Cunningham, W. and Beck, K., (1987), Using Pattern Languages for Object-Oriented Programs, *OOPSLA '87* in Orlando.

[Dale01] Dale, J and Mamdani, , E. (2001), Open Standards for Interoperating Agent-Based Systems, *In: Software Focus*, 1(2), John Wiley.

[Deck+96] Decker, K., Williamson, M. and Sycara, K. , (1996), Matchmaking and Brokering. *In: Proceedings of the Second International Conference on Multi-Agent Systems* (ICMAS-96), December.

[Deck+97] Decker, K., Sycara, K. and Williamson, M., (1997), Middle-Agents for the Internet, in *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI-97)*, January.

[DeMa+78] Demarco,T., Plauger P. J., (1978), *Structured Analysis and System Specification*, Published Yourdon Press.

[D'So+98] D'Souza, D., and Wills, A. C., (1998), *Component and Framework-based Development*, Reading, MA: Addison-Wesley.

[Dubo+94] Dubois, E., Bois, P.D., Dubru, F., Petit, M., (1994), Agent-oriented Requirement Engineering a Case Study the Albert Language, *Fourth International Working Conference on Dynamic Modelling and Object-oriented Paradigm, DYNMOD'94*, Netherlands.

[Dyson97] Dyson, P., (1997), A Framework for Hotel System, *Pattern for Abstract Design*, Phd Thesis.

[Eiter+02] Thomas Eiter and Viviana Mascardi, Comparing Environments for Developing Software Agents, AI Communication, *The European Journal on Artificial Intelligence*, Vol 15. No4., pages 169-197.

[Elam+97] Elammari,M., Buhr, J.A., Gray, T., Mankovski, S. Pinard, D., (1994), Understanding and Defining the Behaviour of Systems of Agents, with Use Case Maps, Report (presented as a Poster Session at *PAAM'97*), http://www.sce.carleton.ca /ftp/pub/UseCaseMaps /4paam97.ps

[Elam+98] Buhr, R.J.A., Elammari, M. Gray, T., Mankovski, S., (1998), Applying Use Case Maps to Multi-agent Systems: A Feature Interaction Example, *Hawaii International Conference on System Sciences (HICSS'98),* Hawaii.

[Elam+99] Ellamari, M. and Lalonde, W., (1999), An Agent-Oriented Methodology: High-level and Intermediate Models. In G. Wagner and E. Yu, editors, *Proc. of the 1$^{st}$. Int. Workshop on Agent-Oriented Information Systems,* [Etzi96] Etzioni, 0., (1996), Moving up the Information Food Chain: Deploying Softbots on the World Wide Web, In Proceeding of the 13th *National Conference on Artifical Intelligence, (AAAI'96)*, Portland.

[Farh96] Farhoodi, F., Graham,I., (1996), A Practical Approach to designing and Building Intelligent Software Agents, *PAAM'96*, Practical Application, London,UK.

[Fayad+01] Fayad, M.,E., Schmidt, D.C. and Johnson, R.,E., (2001), *Building Application of Framework Design*, John Wiley.

[Fini+93] Finin, T., Weber, J., Wiederhold,G., Genesereth, M. Fritzson,R., McKay,D., McGuire, J., Pelavin, R., Shapiro,S., Beck, C.(1993), Draft Specification of the KQML Agent-Communication Language, *The DARPA Knowledge Sharing Initiative External Interfaces Working Group Technical Report*, Jun.

[Fini+97] Finin, T., Labrou, Y. and Mayfield, J., (1997), KQML as an Agent Communication Language, In: *Software Agents*, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press, pages 291-316.

[Fini+98] Finin. T, Peng, Y., Labrou, Y., Chu, B., Long, J., Tolone, W.J. and Boughannam, A., (1998), A Multi-Agent System for Enterprise Integration. *In: Proceedings of the Third International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology*, H.S. Nwana and D.T. Ndumu (Eds.), London, UK, March, 1998.

[FIPA+97a]FIPA 97 Specification, Part 1: *Agent Management. Foundation for Intelligent Physical Agents*, Version 1.2, October 10, 1997.

[FIPA+97b]FIPA 97 Specification, Part 2: *Agent Communication Language. Foundation for Intelligent Physical Agents*, Version 1.2, October 10, 1997.

[Flet94] Fletcher, M., (1994) Some Further Design Considerations for the Congestion Management Mechanism MENTHOL, Technical report, University of Keele, in Jenning,N.,R., Wooldridge,M.,J., *Agent Technology Foundations, Applications and Markets*, Springer Verlag, Germany.

[Flor99] Flores-Mendez, (1999) R. A. Towards a Standardisation of Multi-Agent System Frameworks, http://www.acm.org/crossroads/xrds5-4/multiagent.html

[Fowl96] Fowler, M. (1996), *Analysis Patterns: Reusable Object Models*, Object-Oriented Software Engineering Series, Addison-Wesley, Co.

[Fowl97] Fowler, M and Scott, K., (1997), *UML Distilled*, Addison Wesley Longman, Inc. Reading, MA.

[Frank+97] Franklin,S., Doran, J. E. Jenning, N. R. and T. (1997), Norman J.,. On Cooperation in Multi-Agent Systems, *The Knowledge Engineering Review*, 12(3), also available at http://www.elec.qmw.ac.uk/dai/pubs/fomas.html..

[Frost+96] Frost, H.R., and Cutkosky, M.R., (1996), Design for Manufacturability via Agent Interaction, Proceedings of the *ASME Computers in Engineering Conference*, Irvine, pg. 1-8.

[Gam+95] Gamma, E, Helm,R.,Johnson, R.,Vlissides,J., (1995), *Design Patterns: Elements of Reusable Object-oriented Software*, John Wiley.

[Garl+00] Garlan, D., Monroe R. T., Wile, D., (2000), Acme: Architectural Description of Component-Based Systems, *Foundations of Component-Based Systems*, Gary T. Leavens and Murali Sitaraman (eds), Cambridge University Press, pp. 47-68.

[Gene+87] Genesereth, M. and Nilsson,N.(1987), *Logical Foundation of Artificial Intelligence*, Morgan Kaufman, San Mateo.

[Gene+92] Genesereth, M. and Fikes, R., (1992), Knowledge Interchange Format, Version 3.0 Reference Manual, *Technical Report*, Computer Science Department, Stanford University, USA. .

[Gene97] Genesereth, M., (1997), An Agent-based Framework for Interoperability. In: *Software Agents*, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press, pages 317-345.

[Geor+95] Georgeff, P.,A. and, Michael, S. R., (1995), BDI Agents: From Theory to Practice, *Proceedings of the First Intl. Conference on Multiagent Systems(ICMAS'95),* California..

[Gerva+01] Gervais M.P., Muscutariu F., (2001),Toward an ADL for Designing Agent-Based Systems, submitted for Proceeding on *Agent Oriented Software Engineering*, May 2001.

[Gerva02] Gervais M.P, (2002), ODAC : An Agent-Oriented Methodology Based on ODP. *Journal of Autonomous Agents and Multi-Agent Systems*, 2002.

[Gins+91] Ginsberg, M.(1991), The Knowledge Interchange Format: The KIF of Death. In: *AAAI Magazine*, Volume 12, Issue 3, pages 57-63.

[Giun+02] Giunchiglia,F., Mylopoulos, J. and Perini., A.(2002), The Tropos Software Development Methodology: Processes, Models and Diagrams. Technical Report No. 0111-20, ITC - IRST, Nov 2001 and Summited to *Autonomous Agent-MAS '02.* A Knowledge Level Software Engineering

[Glas96] Glaser, N., (1996),The CoMoMAS and Environment for Multi-Agent System Development, *2<sup>nd</sup> Workshop on DAI, Multi-Agent, Australia, 1996*, LCNA 1286, Springer-Verlag, Germany.

[Goss90] Gossain,S., (1990), *Object-Oriented development and Reuse*, PhD, thesis, University of Essex, UK.

[Gott+96] Gottlob, G.,G., Schrefl, M, and Rock, B., (1996), Extending Object-oriented System with Roles, *ACM Transactions on Information System*, 14(3):268-296, July 1996.

[Graha+97] Graham I., Henderson-Seller, B. and Younessi, H., (1997), *The OPEN process Specification*, Haelow: Addison-Wesley.

[Gran98] Grand,M., (1998), *Patterns in Java: A Catalogue of Reusable Design Patterns Illustrated with UML,* Volume 1, John Wiley & sons, Inc, New York.

[Gran99] Grand, M., (1999), *Patterns in Java Volume 2*, John Wiley & sons, Inc, New York.

[Grand01] Grand M., (2001), *Java Enterprise Design Pattern, Pattern in Java,* Volume 3, John Wiley, USA.

[Grub+93] Gruber, T.R., (1993), A Translation Approach to Portable Ontology Specifications. In*: Proceedings of the Knowledge Acquisition for Knowledge-Based Systems (KAW'93),* Gaines, B.R. and Musen, M. (Eds.), Banff, Canada, pages 199-220.

[Guar+94] Guarino, N., Carrara, M., and Giaretta, P. (1994), Formalizing Ontological Commitment, *In Proceedings of National Conference on Artificial Intelligence (AAAI'94)*. Seattle, Morgan Kaufmann: 560-567.

[Guar+95] Guarino, N. and Giaretta, P., (1995), Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.) *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing,* IOS Press, Amsterdam: 25-32.

[Hamu+98] Hamu, D.S., and Fayad, .E. (1998), Achieving Bottom-Line Imporvements with Enterprise Frameworks, *Communications of ACM* ,41(8), August, 1998.

[Hare+92] Hare, G.M.P., Wooldridge, M.J., (1992), A Software Engineering Perspective on Multi-agent System Design, Experience in the development of MADE, In Nicholas M. Avouris and Les Gasser, editors, *Distributed Artificial Intelligence: Theory and Practise*, Boston.

[Harm93] Harmon, P., and Hall, C., (1993), *Intelligent Software System Development- An IS Manager Guide*, John Wiley & Sons, Inc. New York.

[Harr93] Harries, S., (1993), *Networking and Telecommunications for information systems: An introduction to information networking*, Library Association.

[Haug+98] Haugeneder H., and Steiner, D. (1998), Cooperating Agents: Concepts and Applications, Progress Report, In Jennings.M.R. and Wooldridge, M..J., *Agent Technology foundation, Application and Markets,* Springer-Verlag, Berlin, New York.

[Hayn97] Haynes Thomas, (1997), *Collective* Adaption: Explicit Cooperation in a Competitive Computational Agent Society, *Proceedings of ACM Symposium on Applied Computing.*

[Heid+92] Heidelberg. and Jacobson, I. (1992), *Object Oriented Software Engineering, A Use Case Driven Approach*, Reading MA, Addison-Wesley.

[Herr+00] Jorge L. Daz-Herrera, (2000), Integrating Architectures, Frameworks and Patterns: A Model-based Approach, *Proceeding of Object Oriented Programming Language, (OOPSLA '00)*, also available in www.sei.cmu.edu/ activities/plp/oopsla/jorge.htm.

[Hice+74] Hice G.F., Turner W.S. & Caswell L.F., *System Development Methodology*, North-Holland Publishing Company, Amsterdam, 1974.

[Hofm+99] Hofmeister,C., Nord,R., Soni, D., (1999), *Applied Software Architecture*, Addison Wesley.

[Holt+82] R.C. Holt, J.R. Cordy,(1982), An Introduction to S/SL: Syntax/Semantic Language, in *ACM Transactions on Programming Languages and Systems (TOPLAS),* Vol 4, No. 2, April 1982.

[Hubm97] Hubmann, H, (1997), *Formal foundation for Software Engineering Methods*, LCNS 1322, Springer Verlag, Berlin, Germany.

[Huen+95] Hueni, H., Johson, R., Engel R., (1995), A Framework for Network Protocol Software, Proceeding *OOPSLA*'95, Austin.

[Huhn+98] Huhns, M.N. and Singh, M.P.(1998) Agents and Multi-agent Systems: Themes, Approaches, and Challenges. In: *Readings in Agents*, Huhns, M.N. and Singh, M.P. (Eds.), San Francisco, Calif., Morgan Kaufmann Publishers, pages 1-23.

[Hun+95] Huntback,M.M.,Jenning,N.R., and Ringwood, G.A., (1995) Hows Agent Do it,in Stream Logic Programming, *Proceeding of the First International Conference on MAS*, pp177-184.

[Igle+98] Iglesias, C.A., Garijo, M.,Gonzalez, J.R., (1998). Analysis and Design of MultiAgent System using Common:MAS-CommonKADs, In Singh, M.P.,Roa, A., Wooldridge, M.J., *Intelligents Agents IV (ATAL'97),* LNAI 1365, pages 314-327. Springer Verlag, Berlin, German.

[Jacob+92] Jacobson, I., Christerson,M., Jonsson P., Overgaard, G.,(1992), *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional.

[Jacob+95] Jacobson I., Ericsson M., and Jacobson A.,(1995), *The Object Advantage: Business Process Reengineering With Object Technology,* Addison-Wesley, Reading, Massachusetts, 1995.

[Janc96] Janca, P.C. (1996), *Intelligent Agents: Technology and Application*, GiGa Information Group.

[Jaya94] Jayaratna, N., (1994), *Understand and Evaluation Methodologies, NIMSAD, A Systemic Framework*, McGraw Hill, London.

[Jenn+98] N. R. Jennings, K. P. Sycara, and M. Wooldridge,(1998), A Roadmap of Agent Research and Development In *Journal of Autonomous Agents and Multi-Agent Systems*. 1(1), pages 7-36. July 1998

[Jenn+98a] Jenning, N.R & Wooldrige, M., (1998), *Agent Technology Foundation, Application and Markets*, Springer, 1998.

[Jenn00] Jenning, N.R., On Agent based Software Engineering, *Artificial Intelligence,* 117, 277-296.

[Jenn93] Jennings, N.R., (1993), Commitments and Conventions: The Foundation of Coordination in Multi-agent System, *The Knowledge Engineering Review*, 8(3):223-250.

[Jiro94] Jirotka, M., and Goguen, J., (1994), *Requirement Engineering Social and Technical Issues*, Academic Press.

[Kain99] Kaindl, H, (1999), Difficulties in the Transition from Object Oriented Analysis to Design, *IEEE Software*, September/Oktober), pg. 94-101.

[Kend+00] Kendall, E.A, Deugo, D., and Weiss,M., (2000), Slide at http://scs.carleton.ca/~deugo/Patterns/Presentations/AgentPatterns/ (access on Jan/2000).

[Kend+95] Kendall, E.A., Margaret T.Malkoun and Chong H.Jiang, (1995), A Methodology for Developing Agent Based Systems for Enterprise integration, *First International Workshop Distributed Artificial Intelligent, Architecture and Modelling Canberra, Australia, Nov, 95*, Spinger-Verlag, Germany. LNAI 1087.

[Kend+96] Kendall, E.A. and Margaret T. Malkoun, 1996, Design Pattern for the Development of Multi-agent System, *2nd Workshop on DAI, Multi-Agent, Australia, 1996*, LCNA 1286, Springer-Verlag, Germany.

[Kend+97] Kendall, E. A., M.T. Malkoun and C.H. Jiang, (1997), The Application of Object-Oriented Analysis to Agent Based Systems, The Report on Object Oriented Analysis and Design, *The Journal of Object Oriented Programming*, February, 1997.

[Kend+97b]Kendall E. A., Pathak, C. V., Murali Krishna P. V., Suresh, C. B.,(1997), The Layered Agent Pattern Language, *Pattern Languages of Programming (PLOP'97).*

[Kend92] Kendall, P. (1992), *Introduction of System Analysis to Design*, Prentice Hall, New Jersey, USA.

[Kend98] Kendal, E. (1998), Agent Roles and Role Models, News Abstraction for Intelligent Agent System Analysis and Design, *International Workshop on Intelligent Agent in Information and Process Management,* German Conference on Artificial Intelligence, German.

[Kinn+96] Kinny,D.,Georgeff,M., Rao,A.,(1996), A Methodology and Modelling Technique for systems of BDI agents, In W.van de Velde and J.Perram, editors, Agents Breaking Away: *Proceedings of the Seventh European Workshop on Modelling Autonomous in Multi-agent World (MMAMAW'96)*, LNAI vol. 1038, Springer-Verlag, Germany.

[Kirn92] Kirn,S. and Schneider, J.,(1992), STRICT: Selecting the Right Architecture, *Industrial and Engineering Application of Artificial Intelligence and Expert System,* LCAI 604, Belli, F. and Radermacher, F.J., Springer Verlag.

[Klie95] Klien,M., (1995), Business Process Re-enginneering: Methodologies and Multi-Agent Technologies, *Tutorial at First International Conference on Multi_Agent System, San Francisco, California, USA.*

[Krau97] Kraus, S.,(1997), Negotiation and Cooperation in Multi-agent Environment, *Artificial Intelligence*, 94(1-2):79-97.

[Kris+96] Kristensen,B.B, Olsson, J., (1996), Roles and Patterns in Analysis, Design and Implementation, *Proceeding of the 2nd International Conference on Object Oriented Information System OOIS'96*, London, England.

[Kris96] Kristensen,B.B., (1996), Object Oriented Modelling with Roles, *Proceeding of the 2nd International Conference on Object Oriented Information System, OOIS'96*, Dublin, Ireland.

[Krist+96] Kristensen, B., and Osterbye, K,(1996), Roles Conceptual Abstraction Theory and Practical Languages Issues, *Theory and Practice of Object System*, 2(3):143-160, 1996.

[Larm97] Larman, C., (1997), *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design*, Prentice-Hall, US.

[Lav+95] Lavender, R.G. and Schmidt. D.C. (1995), Active Object: An Object Behaviour Patterns for Concurrent programming, in *Pattern Languages of Programming, PLOP'95, Mountiello, Illinois, USA.*

[Less87] Victor R. Lesser and Daniel D. Corkill. Distributed Problem Solving, *In Encyclopedia of Artificial Intelligence*, pages 245--251. John Wiley & Sons, 1987.

[Lieb95] Lieberman, H. (1995), Letizia, An Agent that Assist Web Browsing, *Proceedings IJCAI 95*, AAAI Press.

[Lixin+00] Lixin Zhou, Wu Zhaohui, Yunhe Pa,(2000), An New Software Architecture Description Language, *Proceeding ICS2000,* (International Conference on Software) Theory and Practice (ICS2000), Beijing, China.

[Ljun+92] Ljungberg,M., Lucas, A. 1992, The OASIS: Air Traffic Management System August, 1992 *Technical Note, 28.*

[Lund96] Lundberg,L., (1996), Multiprocessor Performance Evaluation of Billing Gateway System for Telecomunication Applications. *Proceedings of the ICSA Conference on Parrallel and Distributed Computing System*, Sept, Tokyo, Japan.

[Maes90] Maes, P.(1990), *Designing Autonomous Agents*, The MIT Press, Cambridge,MA.

[Maes94] Maes, P., Agent that reduce work and information overload, *Communication of the ACM*, 37(7), 31-40.

[Mage96] Magendanz, T. (1996) Mobile Agent Based Services Provision in Intelligent Network in *Proceeding ECAI Workshop on Intelligent Agent for Telecom Applications IATA'96*, Budapest.

[Male+00] S. Maleki-Dizaji, H.O.Nyongesa, J.Siddiqqi, Adaptive Multi-agent System for Information Retrieval, Complex Adaptive Structures, Hutchinson Island, Florida USA,4-6 June 2001,

[Mamd98] Mamdani, A. (1998), The Social Impact of Software Agents. *In: Proceedings of the Workshop on The Impact of Agents on Communications and Ethics: What do and don't we know?*, Program presentation, Foundation for Intelligent Physical Agents (FIPA), Dublin, July 15, 1998.

[Mart+88] Martin, J. and McClure, C. (1998), *Structure Techniques, The Basic for CASE*, Prentice Hall, New Jersey, USA.

[Mart+99] David L. M., Adam J. C., Douglas B., M., (1999), The Open Agent Architecture: A Framework for Building Distributed Software Systems, *Applied Artificial Intelligence*, 13(1-2), 91-128, 1999.

[Mats+93] Matsuoka, S. and Yonezawa, (1993) Analysis on Inheritance Anomaly in Object Oriented Concurrent Programming Language, in *Research Direction Concurrent Object-oriented Programming*, Eds. G. Apgha, P. Wegner and A. Yonezawa, MIT Press.

[Matt+96] Wohlin, C., Gustavsson, A., Höst, M., Mattsson, C., A Framework for Technology, Introduction in Software Organizations, *International Conference on Software Process Improvement*, Brighton, UK, 1996.

[MESS00] MESSAGE: Methodology for Engineering System of Software Agent-Initial Methodology,*EURESCOM Participants in Project* p907-GI, July 2000.

[Meye+93] Bertrand Meyer and Jean-Marc Nerson., *Object-oriented Applications*, Prentice-Hall, 1993.

[Meyr86] Meyrowitz, N., (1986), Intermedia: The Architecture and Construction of an Object Oriented Hypermedia System and Application Framework, *Proceedings of OOPSLA 1986*, Printed as SIGPLAN Notices21(11).

[Mitch93] Mitchell, R.J. (1993), *C++ Object-oriented Programming*, Macmillan, 1993, [Mller+97] Mller, J.: A Cooperation Model for Autonomous Agents. In: J.P Mller, M. Wooldridge and N.R. Jennings (eds) *Intelligent Agents III*, LNAI, Vol. 1193. Springer-Verlag, Berlin Heidelberg New York (1997) 245-260.

[Molin+96] Molin. Peter and Lennart Ohlsson, (1996), Points and Derivations: A Pattern Language for Fire Alarm Systems, *Proceedings of the 3rd International Conference on Pattern Languages for Programming,* Monticello, September.

[Moul+96] Moulin, B. and Brossard, M, (1996), A Scenario-based Design and Method and an Environment for Development of Multi-agent System, *First International Workshop Distributed Artificial Intelligent, Architecture and Modelling Canberra, Australia, Nov, 95*, Spinger-Verlag, Germany. LNAI 1087.

[Mow+97] Mowbray, T.J., Malveau, R.C., (1997), *CORBA Design Patterns*, John Wiley, New York.

[Mull93] Mullender ,S.,(1993), *Distributed System*, ACM Press, Addison Wesley.

[Mull96] Muller, H.J. (1996), Towards Agent System Engineering, *International Journal on Data and Knowledge Engineering,* Special Issues on Distributed Expertise, Vol (23).

[Mull99] Muller, J.P. (1999) The Right Agent Architecture to Do the Right Thing, *In Intelligent Agents,* editor J.P.Muller, M..P. Singh and A.S. Rao, LNAO 1555, Springer Verlag, Berlin.

[Neal+94] Nealon, J.L, Smale, A.D. and Holman, R.R. (1994), An Advisory Expert System for Diabetes Treatment, *Expert Systems 94,* Cambridge, Published in Applications and Innovation in Expert Systems II, ed. R. Milne, SGES Publications, 1994.

[Neal+96] Nealon, J L and Greenwood, S (1996), Designing User-Centred Expert Systems: A Medical Case Study, *Third World Congress on Expert Systems*, Seoul, S. Korea.

[Nels99] Nelson, J.,(1999), *Programming Mobile Objects with Java*, John Wiley & Sons, USA.

[Nick+00] Nickel, U., Niere, J. and Zündorf, A., (2000), Tool Demonstration: The FUJABA environment, in *Proc. of the 22$^{nd}$ International Conference on Software Engineering (ICSE),* Limerick, Irland, pp. 742--745, ACM Press.

[Nick+99] Nickel,U., Klein, T. and Niere, J. and Zündorf, A. (1999), From UML to Java And Back Again', *Tech. Rep. tr-ri-00-216*, University of Paderborn, Paderborn, Germany, September 1999.

[Nii89] Nii, H.P., (1989), Blackboard Systems, In: *The Handbook of Artificial Intelligence*, A. Barr, P.R. Cohen and E.A. Feingenbaum (Eds.), Addison-Wesley, New York, Volume IV, chapter XVI, pages 1-82.

[Nwan+99] Nwana, H., Ndumu, D.,Lee, L.and Collis, (1999), ZEUS: A Toolkit for Building Distributed Multi-Agents Systems, In *Applied Artificial Intelligent Journal,* Vol13(1), pp129-186.

[Odel+00] Odell,J., Parunak, H.V.D, Bauer,B., (2000), Extending UML for Agents, Gerd Wagner, Yves Lesperance, and Eric Yu eds, *Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, AAAI'00, Austin, Texas, USA.*

[Odel+01a] James Odell, Bernhard Bauer, Jörg P. Müller, (2001), Agent UML, A Formalism for Specifying Multiagent Interaction, *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103, 2001.

[Odel+01b] James Odell, H. Van Dyke Parunak, Bernhard Bauer,(2001), Representing Agent Interaction Protocols in UML, *Agent-Oriented Software Engineering,* Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 121–140, 2001.

[O'Mall+01] O'Malley, S.,A, DeLoach, S.A. (2001), Determine When the Use an Agent Oriented Software Engineering Paradigm, *2nd International Workshop on Agent-Oriented Software Engineering (AOSE'01),* Montreal, Canada.

[OMG92] Object Management Group: The Common Object Request Broker: Architecture and Specification , *OMG Document,* Number 91.12.1. Revision 1.1.1992.

[Oshi+98] Oshima, M., and Aridor, Y., (1998), Infrastructure for Mobile Agents: Requirements and Design in *Mobile Agents - Second International Workshop, MA '98* (Stuttgart, Germany). Published as K. Rothermel and F. Hohl, editors, LNCS 1477:36-49, Springer, Sep. 1998.

[Othm+02a] Othman,.Z.A. Al-Hashmi,S., Mohamed,S., and Nyongesa,H., (2002), Biometrics-based Authentication in Supply Chain Management, *Supply Chain Accounting and E-business,* Sheffield Hallam University, U.K, .

[Othm+02b] H.O.Nyongesa, S. Maleki-Dizaji, Z. Othman (2002), An Adaptive Multi-Agent Based Search Engine, *EurAsia – ICT 2002:Agents for Information Management*, Iran.

[Othm+02c] Othman, Z.A., Al-Hashmi,S., and Nyongesa,H., (2002), Design of an Internet-based Fuzzy Medical Advisory System based on Multi-Agent Approach, 2nd. *European Medical & Biological Engineering Conference*, December 04-08, 2002 Vienna(Austria).

[Othm+03] Maleki-dizaji S., H.O.Nyongesa, Othman Z.A., (2003), Utilising a Multi-agent approach for Designing An Adaptive Search Engine, *International Conference on Web Intelligence (WI 2003),* Canada, to be appeared.

[Othm00] Othman Z.A (2001), Filtering Application- Reverse Engineering- Multi-Agent System approach, *Technical Report*, Sheffield Hallam University.

[Othm99] Othman, Z.A (2000), CIAgent Reverse Engineering -Technical Report, Sheffield Halam University.

[Padg+02] Padgham, L. and Winikoff, M., (2002), Prometheus: A Methodology for Developing Intelligent Agents, *Proceedings of the Third International Workshop on AgentOriented Software Engineering, at AAMAS'02*. July, 2002, Bologna, Italy.

[Parn+86] Parnas, D.L. and Clements, P.C., (1986), Rational Design Process: How and Why to Fake it, *IEEE Transactions on Software Engineering*, Vol. SE-12, Febuary.

[Paru+01] Parunak, H.V.D. and Odell. J.J., (2001) Representing Social Structures in UML, *Agent-Oriented Software Engineering II, AOSE 2001*, Canada, LNCS 2222, Springer-Verlag.

[Paru+97] Parunak,V., Sauter,J., Clark,S., (1997), Toward the Specification and Design of Industrial Synthetic Ecosystems, *Fourth International Workshop on Agent Theories, Achitectures and Languages (ATAL'97), Rhode Island, USA.*

[Prec97] Prechelt, L. An Experiment on the Usefulness of Design Patterns: Detailed Description and Evaluation. 1997, *Technical Report* 9, Faculty Information, University Karlsruhe, Germany. -chapter 4

[Pree96] Pree, W.,(1996), *Framework Patterns*, New York, SIGS Books.

[Raji97] Rajiv, K, and Dillon,T. 1997, *Engineering Intelligent Hybrid Multi-Agent Systems,* Boston, London Kluwer Academic.

[Rank94] Ranky, P. G.,(1994), *Concurrent / simultaneous engineering* : Methods, Tools and Case Studies, Cimware.

[Rati97] Rational Software Corporation,(1997), *Unified Modelling Language (UML) version 1.0*, Rational Software Corporation.

[Reic+98] Reichert, M., Dadam, P. (1998), ADEPT Flex Supporting Dynamic Changes of Workflows Without Loosing Control, *Journal of Intelligent Information Systems*. Separate Issues on Workflow Management 10 (2), Kluwer Academic Press.

[Ricci+02] Ricci, A, Omicini,A, and Denti, E., (2002), Engineering Agent Societies: A Case Study in Smart Environment, *Autonomous Agent Multi-Agent System, 2002*, Springer Verlag.

[Rieh96] Riehle, D., 1996, Understanding and Using Patterns in Software Development, *Theory and Practice of Object System*, 2,1, pages 3-13. Wiley and Sons.

[Rieh96b] Riehle,D.1996, Explorating of Framework Design Principles, *proceeding of OOPSLA '96, San Jose, California*

[Rieh97] Riehle, D., (1997), A Role-based Design Pattern Catalog of Atomic and Composite Patterns Structure by Patterns Purpose, *Ubilab Technical reports* 97.1.1 Zurich, Switzeland, Union Bank of Switzeland.

[Royce70] Royce, W.W., (1970), Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, August.

[Rumb91] Rumbaugh, J. (1991), *Object Oriented Modelling and Design*, Prentice Hall.

[Rusel+95] Rusell, S. and Norvig, P, (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hal, USA.

[Russ90] Russo, V.F., (1990), *An Object Oriented Operating System*, Phd Thesis, University of Ilinois at Urbana-Champaign.

[Schm+95 Schmidt, D.C. and Vinoski, S., (1995), Object Interconnections, Comparing Alternative Client-Side Distributed Programming Techniques, *SIGS C++*, Report Magazine.

[Schm96] Schmid, H.A., (1996), Creating Applications from Components: A Manufacturing Framework Design, *IEEE Software,* 13(6), Nov.

[Scho98] Schoepke, S.H., (1998), A Business View Regarding the Selection of Agent Development Toolkits, *In Proceeding of the AAAI-98 Workshop on Software Tools for Developing Agents*, Wisconsin, USA, Technical Report WS-98-01, AAAI Press.

[Shaw+93] Shaw,M., Garlan, D., (1993) An Introduction to Software Architecture, in Ambriola, V., and Tortora, G., editor, *Advances in Software Engineering and Knowledge Engineering,* Vol. 1.

[Shaw+96] Shaw, M., Garlan, D., (1996), *Software Architecture -Perspective on the Emerging Displine*, Prentice Hall, New Jersey.

[Shen97] Shen W. & Norrie D., Facilitator, Facilitator, Mediator or Autonomous Agents, In Proceedings of the *Second International Workshop on CSCW in Design,* Bangkok, Thailand, November, 1997. pp 119-124.

[Shoh93] Shoham. Y.,1993, Agent Oriented Programming, *Artificial Intelligent*, 60(1):pg.51-92.

[Simo+99] Simon, B, McRobb, S. and Farmer, R. (1999), *Object-oriented System Analysis and Design using UML*, Mc-Graw Hill.

[Somm+97]Sommerville, I. & Kotonya, G., 1997, *Requirements Engineering Process and Techniques*, John Wiley & Sons.

[Somm99] Sommerville, I., (1999), Software Engineering, Addison Wesley.

[Soni+95] Soni, D.N. and Hofmeister. C., (1995), Software Architecture in Industrial Applications, in *Proceeding of the 17th International Conference on Software Engineering*, Seattle, WA, ACM Press.

[Sou+98] Silva, N.; Sousa, P. and Ramos, C. (1998) A Holonic Manufacturing System Implementation. *Proceedings of the Advanced Summer Institute (ASI'98)*. Bremen, Germany, 14-17 July.

[Sycr95] Sycara, K. (1995), Intelligent Agents and the Information Revolution, UNICOM seminar in *Intelligent Agents and their Business Applications*, 8-9, Nov, London.

[Tans93] Tansley ,D.S.W and Hayball,C.C(1993), *Knowledge-based System Analysis and Design, a KADS developer's handbook*, Prentice Hall.

[Tayl90] Taylor, D.; A.(1990), *Object oriented Technology : A Manager's guide*, Addison Wesley.

[Theo98] Lantzos Theodoros, Helen M. Edwards, Anthony Bryant, Neil Willis: ROMEO: Reverse Engineering from OO Source Code to OMT Design. 191-200 *5th Working Conference on Reverse Engineering, WCRE '98*, October 12-14, 1998, Honolulu, Hawai, USA. IEEE Computer Society Press, 1998,

[Treu+98] Treur,J., Jonker,C.M., Brazier,F.M.T. (1998), Principles of Composition Multi-agent System Development, *In proceeding of IPIP'98, The Conference IT&KNOWS'98*, J.Cuena(ed.) Chapman and Hall.

[Treu+99] Treur,J., Jonker,C.M., Brazier,F.M.T (1999), Design of Intelligent Multi-agent System, *Course Notes,* 10-16 February, 1999, Computer Science Department, Vrije University, Amsterdam

[Vally+98] Vally, J.D., Courdier, R., (1998), A Conceptual ' Roles-Centered' Model for Design of Multi-Agents Systems, *MultiAgent Platform, Toru, Ishida, First Pacific Rim International Workshop on Multi-Agent*, LNAI 1599, Springer-Verlag.

[Vere+90] Vere, S., and Brickmore, T., A Basic Agent, *Computational Intelligence*, 6, pg. 41-60.

[Ves93] Vestal, S. (1993), A Cursory Overview and Comparison of Four Architecture Description Language, *Tenhnical Report*, Honeywell Technology Centre.

[Vird+95] Virdhagriswaran, S., Osisek, D. and O'Connor, P. Standardizing Agent Technology. In: *ACM StandardView*, 1995, Volume 3, Number 3, pages 96-101.

[Waer+93] Waern,A., and Gala,S., (1993), The CommonKADS Agent Model, *Document of KADS-II/M4/TR/SICS/005/2.0*, December 1993, Swedish Institute of Computer Science.

[Weih+95] Weihmayer,R. and Velthuijsen,H., (1995), Distributed AI and Cooperative Systems for Telecommunications, In J. Liebowitz and D.S. Prerau, editors, *Worldwide Intelligent Systems*. ISO Press.

[Weis99] Weiss, G., (1999), *Multiagent System, A Modern Approach to DAI*, The MIT Press, London, UK.

[Wen+98] Wenger,D. and Probst,A.,R., (1998), Adding Value with Intelligent Agents in Financial Services, in Jenning,N.,R., Wooldridge,M.,J., *Agent Technology Foundations, Applications and Markets,* Springer Verlag, Germany.

[Wexe87] Wexelblatt, A. (1987 May). *Report on Scenario Technology,* Technical Report STP-139-87. Austin, TX: MCC.

[Whit+97] White, J.E., (1997) Mobile Agents, in *Software Agents*, J.M. Bradshaw (Ed.), Menlo Park, Calif., AAAI Press, pages 437-472.

[Wied+92] Wiederhold, G., (1992), Mediators in the Architecture of Future Information Systems, in: *IEEE Computer*, March 1992, pages 38-49.

[Wils75] Wilson, E.O., (1995), *Sociobiology,* Belknap Press, Cambridge, MA.

[Wins79] Winston, P.H. and Richard, H.B., (1979), Expert Problem Solving, Natural Language Understanding, Intelligent Computer Coaches, Representation and Learning, *Artificial intelligence : an MIT perspective/edited by Vol.1,*MIT Press.

[Wood+00] Wood, M., and DeLoach, S.A., An Overview of the MultiAgent System Engineering Methodology, In P.Ciancarini and M.J.Wooldridge, editor, *Proceeding 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE'00),* Springer-Verlag, Berlin.

[Wood+01] Wood, K.L., Jensen, D., Bezdek,J, Otto, K. N., (2001), Reverse Engineering and Redesign:Courses to Incrementally and Systematically Teach Design, *Journal of Engineering Education,* July.

[Wool+00] Michael Wooldridge, Nicholas R. Jennings, David Kinny, (2000), The Gaia Methodology for Agent-Oriented Analysis and Design, *Autonomous Agents and Multi-Agent Systems 3(3):* 285-312 (2000)

[Wool+95a]Wooldridge,M., and Jenning, N.R,(1995), Intelligent Agent, Theory and Practice, *The Knowledge Engineering Review,* Vol.10(2).

[Wool+98] Wooldridge, W. and Haddadi,A., (1998), Making it up as they go along: A Theory of Reactive Cooperation. In W. Wobcke, M. Pagnucco, and C. Zhang, editors, *Agents and Multi-Agent Systems- Formalisms, Methodologies, and Applications (LNAI Volume 1441).* Springer-Verlag, June.

[Wool+98a]Wooldridge, M., and Jennings, N. R.1998. Pitfalls of Agent-Oriented Development. *In Proceedings of the Second International Conference on AutonomousAgents*, 385-390. Minneapolis/St. Paul, MN. USA, May 9-13, and *IEEE Transactions on Software Engineering*, 144(1): 2637.

[Wool+J95]Wooldridge, M.J. and Jenning, N.R. 1995, Agent Theories, Architecture and Languages, In Intelligent Agents: ECAI-94 *Workshop on Agent Theories, Architecture and Languages* eds. M.J.Wooldridge and N.R. Jenning, Springer-Verlag, Berlin, German.

[Wool01] Wooldridge, M., (2001), *An Introduction to Multiagent System*, John Wiley and Sons. Ltd.

[Wool97] Wooldridge,M., (1997), Issues in Agent Based Software Engineering, *Proceeding on Co-operative Information System*, 97, LNAI 1202, Springer-verlag, Germany.

[Wu99] Wu, C. Thomas, (1999), *An Introduction to Object-oriented Programming with Java/C*, McGraw-Hill, London.

[Wyne93] Wynekoop, J.L., and Russo N.L., (1993) System Development Methodologies: Unanswered Questions and the Research- Practice gap, in DeGross, J.I., Bostom, R., Robey D., (eds) *Proceedings of the Fourteenth International Conference on Information System*, Orlando.

[Your79] Yourdon, E., (1979) *Classics in SoftwareEngineering*, Press, 1979.

[Your89] Yourdon,E. (1989), *Modern Structures Analysis,* Prentice Hall.

[Zam+00] Zambonelli,F., Jennings,N., Wooldridge, M., (2000), Organisation Abstractions for the Analysis and Design of Multi-agent system, *Agent Theories, Architectures, and Languages (ATAL'99),* LNAI 1757, Springer-Verlag.

[Zam01] Franco Zambonelli, (2001) Abstractions and Infrastructures for the Design and Development of Mobile Agent Organizations. In Michael Wooldridge, Paolo Ciancarini, Gerhard Weiss, editors, *2nd. International Workshop on Agent-Oriented Software Engineering (AOSE'01)*, Montreal, Canada. Proceedings.

[ADL00] ADLs and Related Languages, http://www-2.cs.cmu.edu/~acme/adltk/ index.html.

[Agle98] IBM Aglets, (1998), Web page: http://www.trl.ibm.co.jp/aglets/

[Alta99] AltaVista, (www.altavista.com)

[App97] Appleton, B.,(1997), Patterns and Software: Essential Concepts and Terminology, http://www.enteract.com/~bradapp/docs/patterns-intro.html.

[APRIL00] APRIL Agent Platform, http://sf.us.agentcities.net/aap/

[AUML00] Agent Unified Modelling Languages, http://www.auml.org/.

[Broke99] Broker agent collections, http://www.swi.psy.uva.nl/projects/IBROW3/ related-brokers.html

[BUIL00] Agent Builder ToolKit, http://www.agentbuilder.com/

[COM00] Comtec Agent Platform, http://ias.comtec.co.jp/ap/

[Comm97] CommonKADs, http://www.commonkads.uva.nl/frameset-commonkads.htm

[Cool99] Cool, 1999, http://www.eil.utoronto.ca/ABS-page/ABS-overview.html

[Coper99] Coopernic, http:// www.coorpenic.com

[Corba99] Catalog of CORBA/IIOP Specification, http://www.omg.org/technology documents/corba_spec_catalog.htm

[Craw99] MetaCrawler, http://www.metacrawler.com

[DMARS98] AAII agent, DMARS, http://www.agents.org.au/19990625AAII-UoM990624.pdf

[Faul+03] Faulkner, S, Kolp,M, Towards and agent architectural description language for Information Systems,

[FIPA00] FIPA,2000, http://www/fipa.org.

[FIPA99] FiPa Rational, (1999), http://www.cselt.stet.it/fipa/fipa-rationale.htm.

[FIPAOS00] FIPA-OS http://fipa-os.sourceforge.net/

[Fly98] Firefly, http://www.firefly-games.com/

[Fuzzy00] Fuzzy rule framework, http://file.inf.bi.ruhr-uni-bochum.de/ infdoc/doc_new/Programming/Java/Packages/nrc.fuzzy/APIdocs/overview-tree.html

[Goog00] Google, http://www.google.com

[Hotb00] Hotbot, http://www.hotbot.com

[JACK00] JACK Intelligent Agent, http://www.agent-software.com/shared/products/index.html

[Jade98] Jade Development Environment, http://sharon.cselt.it/projects/jade/

[Jaf97] JAFMAS, Java-Based Framework for Multi-Agent System, http://www.ececs.uc.edu/~abaker/JAFMAS/

[Jat97] Jatlite, http//java.stanford.edu/index2.htm

[Jat98] Jatlitebean, (1998), http://kmi.open.ac.uk/people/emanuela/JATLiteBean/

[Jess00] Jess expert system shell, http://herzberg.ca.sandia.gov/jess/

[Lea93] Christopher Alexander: An Introduction for Object-Oriented Designers Doug Lea, 1993, http://g.oswego.edu/dl/ca/ca/ca.html.

[Leap02] LEAP (Lightweight Extensible Agent Platform, http://leap.crm-paris.com/

[MADL03] Mobile Enterprise Architecture Description Language, http://www.dstc.edu.au/m3/architecture.htm

[Mail02] Java Mail API, http://java.sun.com/products/javamail/

[Medv+97] Domains of Concern in Software Architectures and Architecture Description Languages, http://www.ics.uci.edu/~neno/dsl/dsl97.html.

[MEM00] Fipa Memberships, http://www.fipa.org/about/membership_list.html

[OAA01] Open Agent Architecture: Technical White Paper-http://www.ai.sri.com/~oaa/whitepaper.html, http://www.ai.sri.com/~oaa/

[Obje+98] ObjectSpace Voyager 2.0, User Guide. 1998. Web page:http://www.objectspace.com/developers/voyager/

[Open99] Open Sesama software agent, http://groups.yahoo.com/group/OpenSesame/

[Phone02] Java Mobile Phone, http://java.sun.com/j2me/

[Phone03] Java Mobile Phone, http://javamobiles.com/#JavaPhones

[Secur+02] Blue Iguana Security Methodology and Infrastructure: www.blueiguana.com/pdf/BI_Security0902.pdf

[Smart00] Smart search engine and buying selling agent collection, http://www.njads.net/www_smartbots_com.html

[Temp96] AG, 1996, http://hillside.net/patterns/Writing/template.htm

[Toolkit03] Agent Toolkits, http://www.cems.uwe.ac.uk/~rsmith/ECOMAS/agent_toolkit_list_(courtesy_of_bt).htm

[Tools99] Agent Framework Development Tools, 99, http://www.agentbuilder.com/agentTools/index.html

[Trieme99] Abstraction in Software models and how to refine a an Software Requirement Specification,http://www.trireme.com/whitepapers/process/definition/abstraction.html

[Watch01] NewsWatcher, http://www.newswatcher.com

[Wcraw01] WebCrawler, http://www.WebCrawler.com

[Webd99] WebDoggies, http://www.webdoggie.net

[Who98] Global Burden of Diabetes, Reuter Report, 1998, http://www.who.int/inf-pr-1998/en/pr98-63.html

[Wire01] Guide to Mobile Data Communication, 2001, http://www.diffuse.org/mobile-g.html

[Yahoo00] Yahoo, http://www.yahoo.com

[ZEUS00] Zeus Agent ToolKit, http://sourceforge.net/projects/zeusagent/
http://www.iag.ucl.ac.be/recherche/Papers/WP75FaulknerKolp.pdf

# Appendix A: Analysis Existing Agent-Oriented Methodologies (AOM) and the Result

## A-1: Extension from Knowledge Engineering Approaches.

Three of the agent-oriented methodologies analysed extended the knowledge engineering approach: CoMoMAS[Glas96], MAS-CommonKADs[Igle+98], and DESIRE[Treu+99] [Treu+98]. The first two are extensions of the CommonKADs methodology, whereas DESIRE is an agent development methodology, based on components where each component consists of several tasks, which interact with each other to achieve the system goal. However, we further detailed design is more in line with the knowledge engineering approach. We classified DESIRE as an extension of the knowledge engineering approach.

CommonKADs is European standard for development of knowledge engineering systems. It is a generic task model involving patterns in knowledge engineering and a template library of expert models which is useful to provide agent intelligence[Comm97]. Even though CoMoMAS and MAS-CommonKADs are similar, MAS-CommonKADs is more advanced. MAS-CommonKADs adopts a CommonKADs methodology for presenting intelligence. It also adopts the object-oriented techniques for process modelling. Each of the methodologies will be discussed here.

*DESIRE*

DESIRE [Treu+99] consists of the following five stages of development. The first is *problem description,* which describes the requirements imposed on the design, and the rationale, which specifies for each level the assumptions with respect to its use. This includes defining the user, system and domain requirements of the system based on a task perspective.

The second is *conceptual design,* which describes the conceptual view of agents, the external world and the relationship between agents and the external world. This stage should be able to define the agents' links with the external world, how many agents are required in the system and what relations exit between agents and the external world. We use the conceptual design in our high level design, but we have a different way of representing the conceptual design. The conceptual design is similar to agent system design in our methodology, which describes the architecture of the whole system that shows how agents interact with each other and which agents interact with the external world, in order to achieve the system goal. Providing knowledge for each agent is similar to the individual agent design of our methodology. Providing behaviour for the agent determines how the information flow between agents is managed in order to be able to present the behaviour. This stage is similar to our agent system design.

The third stage is *detailed design.* At this stage, each component in the conceptual design is decomposed into a detailed design based on the following three types of composition: process composition, knowledge composition and the relation between process composition and knowledge composition. DESIRE focuses on detailed design by providing the knowledge and behaviour for the agent system as a whole using a generic agent model to represent agent behaviours of the system. In detailed design, the DESIRE agent system is composed of several processes that describe the behaviour of agents. The agent behaviour is represented by decomposing into several processes that interact with each other.

The generic agent model mainly consists of the following abstractions: world interaction management, agent interaction management, maintenance of agent information, maintenance of agent information, cooperation management, agent specific task and own process control.

The fourth stage is *operational design.* DESIRE provides a tool that generates the prototype system automatically from detailed design. This stage is similar to our implementation stage.

Even though DESIRE presents a different agent perspective from our methodology, we appreciate that the methodology presents a clear process from conceptual design into detailed design. This idea has influenced us to have a high level design as an inter-mediator before going into further detailed design, which we have called modelling through abstraction.

The second advantage of the DESIRE methodology is the existence of a generic agent model that enables reuse and is useful in reducing the complexity of agent development. The generic agent model provides a core component for agent design. It provides patterns for agent design and it has the advantage of reducing the complexity of designing agents. We adopted this method but since our methodology is extended from object-oriented methodology, we used object-oriented software abstraction tools to fulfil this objective. Agent expertise such as diagnosis, information retrieval, learning, etc. is used to identify agent specific task.

On the other hand, DESIRE differentiates the development system into a single agent and multi-agent system approach. A single agent is used for a task perspective, which consists of an appropriate agent and the external world, whereas a multi-agent system approach is used when more agents are identified, and when more then one external world is observed.

We agree with the DESIRE approach to identify the solution system whether using a single agent or a multi-agent system approach at early stages. The development of a single agent follows the individual agent design, which focuses on an individual agent itself.

Even though DESIRE has an operational design stage it not discuss how the system operates, but rather makes an assumption that the properties hold and does made explicit in the design rationale. DESIRE focuses on the logical design aspect rather than the physical.

*The CoMoMAS methodology*

Glaser [Glas96] proposed CoMoMAS as an extension to the CommonKADs methodology for multi-agent systems. The methodology consists of analysis and design phases. The analysis phase consists of the following models: Agent Model, Expertise Model, Task Model, Cooperation Model and System Model, which are directly adapted from the CommonKADs methodology. CoMoMAS uses the CommonKADs application model as an agent model.

These five models are useful in representing the early analysis stage of our methodology, in a similar to CoMoMAS. Even though CoMoMAS has different views on agent modelling, we can still see the similarity. In the design phase, the design model refines the previous models into an 'operational model'. Each model is described as follows.

*Agent Model*: is the main model of the methodology, which defines the agent architecture and agent knowledge. At this stage the agent type is classified as reactive, social, mobile, etc. The Agent model is similar to our stage of identifying agent behaviour and agent type in the high level design. In this stage, the conceptual distribution should be utilised in the agent model. The agent model is similar to our agent abstraction.

*Expertise Model*: describes the capability of the agent. It distinguishes between task, problem solving and reactive knowledge. For instance, task knowledge consists of decomposition knowledge described in the following task model. In our methodology, the expertise model is used to model the intelligence behaviour of an agent.

*Task Model* describes the decomposition of tasks for the agent. The decomposition of tasks describes the tasks associated with the agent that are needed in order to achieve the agent goal. This task model is a similar concept to our methodology. The expertise model and task model are related to the agent model.

*Cooperation Model*: describes the cooperation between agents. It uses conflict resolution methods and cooperation knowledge, including communication primitives, protocols and interaction terminology. The cooperation model is similar to our methodology for identifying the communication protocol among agents.

*System Model* defines the organisation of the system. It presents the agents' society corresponding to the architecture. The system model is quite similar to our methodology when identifying the agent environment, but Glaser does not provide a detailed discussion of this stage.

*Design model* consists of the refinements of the collection of the previous models into an operational model. This stage is similar to our detailed design, which aims to be directly transferable to a programming language.

*The MAS-CommonKADs methodology*

MAS-CommonKADs has not only adopted the CommonKADs modelling, but has enhanced it using object-oriented techniques (OMT, OOSE, RDD) as a part of the modelling process, for example in a coordination model and organisation model. It also extends the agent protocol (Semantic Description Language). Many parts of MAS-CommonKADs are similar to our methodology.

The extensions of MAS-CommonKADs are defined in the following models:

*Agent model-* describes the agent type, reasoning capabilities, skills (sensors or efectors), services, agent groups and hierarchies.

*Task model-* describes the tasks that the agents can carry out: goals, decompositions, problem-solving methods.

*Expertise model-* provides the knowledge needed by the agents to achieve goals or provide reasoning knowledge.

*Coordination model-*describes the conversations between agents. It includes their interaction protocols and interaction capabilities.

*Organisation model* -describes the organisation in which the multi-agent system is going to be introduced and describes the organisation of the agent society. It uses extensions of OMT to describe the agent hierarchy, relationship between agents and their environment, and the agent society structure.

*Communication model* -details human-software agent interaction and the human factor aspect for developing the user interfaces.

These models are useful for analysis modelling. The communication model is recognised as just a task of an agent in which an agent communicates with the external world.

*Design model-* collects the previous models to produce another three models: application design, architecture design and platform design. The application design consists of composition or decomposition of the agents of the analysis, according to pragmatic criteria and selection of the most suitable agent architecture for each agent.

The architecture design includes the design of the relevant aspects of networking and telematic facilities. The platform design involves the selection of agent platforms that will be used for execution of each agent.

We can make comparisons between MAS-CommonKADs and our methodology in three categories. First, the agent model, task model and expertise model are related to our individual agent design. Second, the coordination model and communication model are related to a high-level system design. Third, organisation model and design model are related to physical design. However, based on their published articles, the papers do not indicate the process of development of the application design, architecture design and platform design. Therefore, the MAS-CommonKADs is only focused on logical design.

Based upon the three existing agent-oriented methodologies extended from knowledge engineering approaches, we conclude that their methodologies have made a significant contribution to the modelling agent systems. Even though the term of modelling is quite similar, the modelling processes are different.

The generic agent model provided in DESIRE is like a pattern for development of agents. This generic agent model is reusable and can reduce the complexity of the development process. This method is similar to our development process using software abstraction tools to allow reusability and reduce the complexity of the agent development process for the development of the agent environment.

### A-2: Extensions of Object-oriented approaches.

Several agent-oriented development methodologies have been proposed which use an extension of object-oriented approaches, such as agent-oriented analysis and design [Burm96], Agent Modelling Techniques for systems of BDI agents [Kinn+96]; Multi-agent scenario-based method [Moul+96], An Agent-Oriented Methodology: High-Level and Intermediate Models [Elam+98][Elam+99], Agent-oriented methodology for enterprise modelling [Kend+95][Kend+97], Methodology for design of agent-based production control system [Buss+00] and Gaia [Wool+00].

Since our methodology is also extended from object-oriented approaches, these methodologies are easy to compare against our methodology. An overview of each methodology is given below, as well as an account of how each methodology has contributed to the development of our methodology.

*Agent-oriented analysis and design*

[Burm96] defines three models for analysing an agent system. The *agent model* contains with agents and their internal structure, like beliefs, plans, motivation and goals using CRC cards. This agent model is similar to our individual agent design.

The *organisation model* describes the roles of each agent, using OMT inheritance hierarchy and relationship between agents. It corresponds with some parts of our agent system design and some parts of development of agent environment.

The *cooperation model* describes how the type of interchange messages and use protocols are analysed. This cooperation model corresponds in our methodology to the identification of the agent communication protocols used for agent interaction in the agent design system stage. For us, the cooperation model is based on interaction of two agents.

*Agent Modelling Technique for Belief, Desire and Intention (BDI) agents system*

[Kinn+96] defines two main levels (external and internal) for modelling BDI agents. The external point of view consists of the decomposition of the system into agents and the definition of interactions. This is carried out through two models: the *agent model* which describes the hierarchical relationship between agents and the relationship between concrete agents and *interaction models*, for describing the responsibilities, services, and interaction between agents and external system(s).

The internal viewpoint is focused on modelling each BDI agent class through three models. The belief model describes how agents make assumptions that lead to the belief of the state in the environment; the goal model describes the goals and events that an agent responds and adapts to; and the plan model describes the plan an agent aims to achieve with its goals. The internal viewpoint is similar to our agent design level, except that it is focused only on BDI agents.

The external model is similar to our agent system design of identifying agents and its interaction, whereas the internal model is similar to our individual agent design, but the methodology is focused on the internal model, based on BDI agents. For us, the BDI just one type of agent and a multi-agent system may consist of several agent types which interact with each other.

The external viewpoint starts the development process by identifying the roles of the application domain, identifying the agents and arranges the class agent using an OMT notation. Then the responsibility associated with each role is identified, thereby identifying the service provided. The next step is to identify the necessary interaction for each service using speech acts and the information content of every interaction.

In our methodology, each agent is associated with a class agent. But before creating the class agent, we develop a core agent which represents similar attributes for all agents which classes can inherit from. The core agent is developed by utilising concepts from the object-oriented approach, i.e. inheritance. This 'core agent' is designed in the high level architectural design stage of the Development of the agent environment.

The internal viewpoint starts with analysing the plan for achieving goals, followed by the events for achieving the goals. Hence, the beliefs of the agents are identified as objects in the environment to be modelled and presented using OMT. The internal viewpoint is similar to our individual agent design.

*Multi-agent scenario-based method.*

This methodology is applied in the field of cooperative work. [Moul+96] proposed an analysis and design agent based on scenarios. We also use scenarios as part of analysis modelling. The analysis stage consists of the following activities: scenario description, role function, data and world conceptual modelling, system user interaction modelling. The design phase consists of the following activities: design of the multi-agent architecture and scenario description; selection of the scenario to be implemented and the roles played by the agents in these scenarios.

Object modelling refines the world modelling in the analysis in terms of procedures and attributes. Agent modelling describes the specification of the elements defined in the data conceptual modelling step of analysis as belief structures.

The multi-agent scenario provides a good foundation for requirements analysis as it helps to understand the system in order to capture the system goals. For us, scenarios are used in the early stage of analysis.

*Agent oriented methodology for enterprise modelling.*

The methodology [Kend+95] proposes the combination of object-oriented analysis methodologies (OOSE) and enterprise modelling methodologies (IDEF) and CIMOSA (Computer Integrated Manufacturing Open System Architecture). The models identified are:

1. *Function model* -selection of possible methods that depend on input and the control. The function method is similar to our task model.
2. *Use case model-* The use case model describes who the actor for each function using OOSE notation. We do not focus only on actors, but also on knowledge and events.

3. *Dynamic model* - shows the object interactions. An event trace diagram is used to represent the use case. We represent the dynamic modelling based on interaction between agents, not objects.

These models are used to describe the agent-oriented system, which consists of the following elements: agent identification, coordination protocols, plan and belief, sensor and effector.

The agent modelling uses a combination of structured and object-oriented modelling. We agree with their suggestion to use the use case model and dynamic model. In contrast, the methodology only covers the analysis to high-level agent design, without consider the physical design aspect.

*An Agent-Oriented Methodology: High-Level and Intermediate Models.*

[Elam+98] has suggested the agent-oriented software engineering approach by introducing the use of object-oriented techniques, which capture the objects in the system, their attributes, structure, and relationship. The agent model captures agents with their attributes, structure and relationship. In our methodology, the agent model is captured based on behaviour, represented as attributes, structure and relationship.

A major difference between objects and agents is their internal structure. Objects encapsulate methods and attributes, while agents encapsulate goals, plans, beliefs, and commitments. This distinction forms an underlying assumption in our approach.

The design process that is tailored to agents includes the following models: high level, internal agent, relationship, conversation and contract model. These models are used in the two stages of the discovery phase and the definition phase. The discovery phase identifies agents and high-level behaviour using the five models mentioned above, whereas the definition phase produces an implementable modelling stage. Some of the models presented in this methodology are quite similar to those in our methodology: the high-level and internal agent stages are similar. The relationship, conversation and contract model are related to interaction modelling. Even though [Elam+98] proposed the implementable definitions stage but they do not discuss that phase and do not show the modelling process to produce the model.

*Methodology for design of agent-based production control system*[Buss+00].

This methodology takes a different approach to agent development. It uses a model of decision-making in production control in order to enable the designer to move directly from the domain to the agent-oriented design aspects and uses a set of criteria for design of agent -related aspects, which guides the designer with no prior agent-related experience. For our purpose, the key element of this methodology is that it uses distributed decision making for identifying agents, akin to the conceptual distribution in our agent methodology.

*Gaia methodology.*

Gaia [Wool+00] methodology provides general agent-oriented analysis and design, which developed based on experience of [Hare+92]. The methodology is similar approach to our methodology. The methodology consists of two levels of agent development: micro-level (individual agent structure) and macro-level(social and organisational structure). Gaia covers the analysis and design stages. The micro-level is similar to our individual agent design. The macro-level is quite similar to the agent system design part. However, Gaia does not address the agent environment.

The analysis process involves finding the roles in the system and modelling the interaction between the roles to produce the role model and interaction model. The attributes that define the roles are: responsibilities, permission, activities and protocol.

Gaia consists of three design processes. The first is to map the role into agent type and then create the right number of agent instances of each type. However, the problem of using this method is that designer have problem of defining role for certain type of problem domain. The case study shown in Gaia is a business process management system, in which the role is clearly identified using an organisational perspective. However, the role technique is difficult to apply in a circumstance where roles are unclear or difficult to define from the user requirement. Therefore, an inter-mediator modelling process is reeded to show how roles are assigned to agents and why an agent is necessary to perform the role.

The second is to determine the service model needed to fulfil a role in one or several agents. For us this service model is used in agent environment development part. Lastly, a term of acquaintance is used to represent the communication between the agents.

The Gaia design process is similar to our refinement of design of the agent system. This stage is similar that of agent construction. We also agree with Gaia in using role as well. However, the proposed inter-mediator modelling used for analysis is different for us. In the design stage, there is a big gap between the analysis and detailed design stages. Our methodology reduces this gap by dividing agent development into two parts: development of the agent system and development of the agent environment. Each part focuses on the attributes related to that part of development.

The detailed design focuses on development of the agent environment. Therefore, the agent environment stage is responsible for identifying the core agent paradigm and agent functionality used for agent development.

### A-3: Result of Analysis.

The previous section compared existing methodologies with our methodologies in terms of stages and techniques used. The discussion above also shows that our methodology uses certain modelling techniques exhibited in the existing agent-oriented methodologies and relates the attributes and stages of the existing agent-oriented methodologies to the stages of our methodology. This analysis is based on a synthesis of the literature. In this section, we present the result of comparisons related to the following issues:

- The concept of agent design.
- The life cycle of the agent development process.
- How far the methodologies focus on each stage.
- How far they are concerned with the four agent oriented system characteristics identified.

The methodologies reveal several methods of modelling agents. CoMoMAS specifies the agent type at an early stage of analysis as reactive, social, etc. The MAS-CommonKADs also uses a similar method with additional entities to describe agents such as reasoning, capabilities, skills, services, etc. But it is not shown how to analyse or judge what agent type is suitable for a given agent solution. [Burm96] models agents as belief, plan, motivations and goals. [Kinn+96] models agents based on belief, desire and intention. [Elam+99] models agents based on active objects, the behaviour is described based on attributes, structures and relationships.

We agree with CoMoMas and CommonKADs that modelling agents are based on agent types, as we use the term of 'agent abstraction' and agent abstraction is represented based on the agent capability. This means Kinny's methodology is focused on a specific agent abstraction of capabilities as belief, desire and intention. In our methodology, we wanted to show the analysis process for identifying the agent abstraction.

In terms of the agent modelling concept, we concur with [Vally+98][Wool+00][Burm96] in using to use the role concept to analyse and design agent systems. The role is able to abstract a representation of functionality, services or identification of many agents at a higher level to an agent at a lower level [Vally+98]. On the other hand, the lower agent plays a role with respect to the agent at the higher level. Therefore the use of the role concept is able to reduce the gap between analyses at the high level to the lower level of identifying agents. Furthermore, we argue that, in the context of an agent-based system, the role concept is used for various purposes. First, the role concept is able to present a collective agent structures from groups or organisation [Zam+00]. An organisation role model incorporates different roles, their relationships and behavioural rules. Second, roles are used for modelling interaction protocols (see e.g., [FIPA+97b]). In this case, each agent participating in a protocol is assigned a certain role, which it plays as long as the protocol is performed. The notion of role provides a well-defined position in an organisation and a set of behaviours expressed by protocols for the roles. Third, the role method is applicable as an extension of the object-oriented approach, as shown by [Gott+96] [Krist+96]. Wooldridge argues the roles are components from which agents are to be composed [Wood+00].

We follow the role concept by [Vally+98] as the following :

Role = <Goal, Properties, Resources, States>

They define a role associates with goal, properties, resource and states. The **Goal** corresponds to a satisfaction function of the agent for its role. An agent defines a global goal correspond to the priorities on its roles and consequently a priority on its goals. The **properties** describe the specification of behaviour, attributes or roles, which an agent must satisfy to be able to play this role. The **resource** describes the resources used by the agent to play the role and the **state** depends on the role. The state can be internal or external. An agent may have many roles, while a role may be performed by cooperation of several agents. The concept of role is suitable for designing agents and is applicable to any agent system. This role concept can also be used to represent the concrete agent itself [Kend98].

In term of the life cycle of development, none of the methodologies mentioned make use of any specific development process life cycle, but rather provide the stages based on the waterfall model, such as requirements, analysis, design and implementation. However, most of the methodologies focused on the analysis and design stages, because these two stages are the most important stages; this is the distinction between the traditional approaches and agent-oriented approaches. But they do not show a clear definition and transaction of the modelling process. Table A3.1 shows how far the methodologies have focused on stages of the agent development process according to the traditional life cycle process (analysis, design and implementation).

The table shows that the problem of current methodology is a big gap from the analysis to the design stage, we believe the modelling we propose reduces this gap. The grey colour describes how far the methodologies discuss the issues of each stage, while the patterned area present the coverage of modelling provided. The implementation stage refers to provision of an implemented case study. The table shows that the CoMoMAS and MAS-CommonKADs focus on analysis modelling rather than design, while the DESIRE methodology is more focused on detailed design rather than analysis. The DESIRE methodology presents a smooth transition between analysis and design.

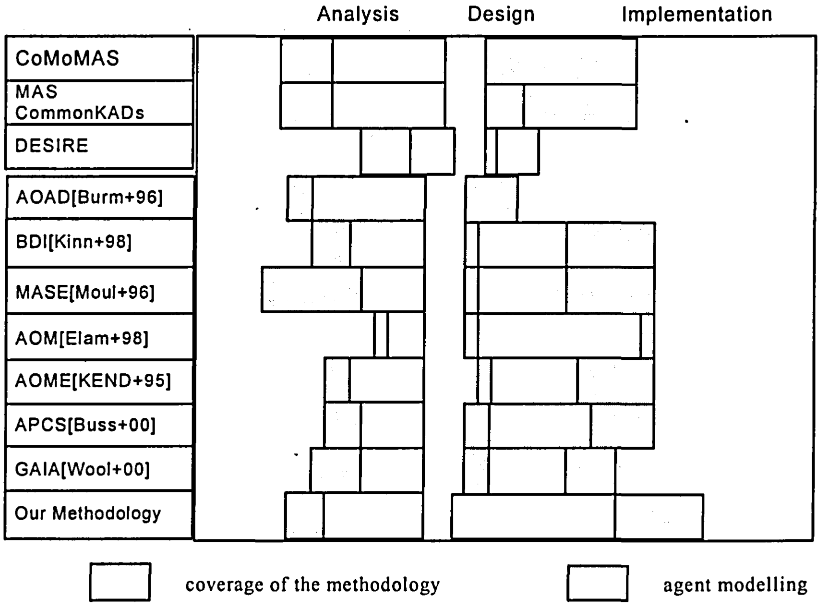


Table A3.1: The Coverage of the Existing Agent-oriented Methodologies.

The methodologies extending the object-oriented paradigm differ in how far they have focused on each stage and how far analysis and design modelling have been taken. For example, MASE [Moul+96] focuses on requirements more than others, while [Elam+98] focuses more on the design stage. On the other hand, these methodologies show a gap between the analysis and design stage. For example, AOM [Elam+98] uses analysis based on agent model and design according to the system view, relationships between agent, conversation and commitment. The gap between these two stages is manifested in unclear description of how the agent model maps to the detailed design, and how the agent model is defined and built. In our methodology, we use the agent abstraction method to define the agent model in different stages of development. In this way we believe our methodology provides an important modelling process between these two stages.

In addition, our methodology is different from its predecessors in the separation of the two parts of the agent development process. Each part consists of its own life cycle of development. The stages of the existing methodologies contain a mixture of these two parts. Most of the methodologies only focus on logical design rather than focus on physical aspect. For us, physical requirements are an important issue to be considered at an early stage of requirements. We can generalise that the existing agent-oriented methodologies have covered the development of the agent system, but not the development of the agent environment part.

| Methodologies | Behaviour | Communication | | | Distribution | | Openness | | |
|---|---|---|---|---|---|---|---|---|---|
| | | with external | inter agents | other entities | conceptual | location | speech Act | message transportatio | service levels |
| MASCommon KADs | | √ | √ | √ | √ | √ | √ | not mentioned | not focused |
| CoMoMAS | √ | √ | √ | | √ | not mentioned | | | |
| DESIRE | √ | √ | √ | | √ | | | | |
| | | | | | | | | | |
| AOAD [Burm96] | √ | √ | √ | | √ | | | | |
| BDI [Kinn+98] | √ | √ | √ | | √ | | √ | | |
| MASE [Moul+96] | √ | √ | √ | √ | √ | | | | |
| AOM [Elam+98] | √ | | √ | | √ | | √ | | |
| AOME [Kend+95] | √ | | √· | | √ | | √ | | |
| APCS [Buss+00] | √ | | √ | | √ | | | | |
| GAIA[Wool+00] | √ | | √ | | √ | | √ | | |
| | | | | | | | | | |

Table A3.2: The summary of the existing agent-oriented development Methodology versus the agent characterisation.

We also analysed the existing agent-oriented methodologies according to the four agent characteristics and the result is shown in Table A3.2. The table shows how far the methodologies focus on the agent characteristics. Some of the methodologies do not mention directly that they focus on the agent characteristics but rather mention them implicitly. The table shows that all the existing methodologies focus on the agent behaviour and communication, but they do not focus on openness criterion. Even though the methodologies present the distribution aspect, they do not focus on it as an aspect of design. In addition, they only focus on conceptual distribution rather than physical distribution.

Our methodology has categorised the distribution characteristics as the second issue in designing agent systems, after agent behaviour. We discovered that the existing methodologies had presented conceptual distribution differently. These differences influenced us to invent agent abstraction for designing agent system. The agent characteristics are also used to describe the agent abstraction.

# Appendix B: Reverse Engineering

## B-1: Reverse Engineering Process Towards Development Agent System.

The Newsfilter application only provides Java Implementation codes rather than provides a proper documentation of the development process. The Newsfilter was developed in Java Version.1.1 using Symantec Visual Development Tools but has been upgraded into Java 1.2, using Jbuilder Version 3 by us in order to execute the system. The execution of the system is part of activities in RE process.

The Filtering application was developed in ad hoc fashion by us [Othm+02b] in Java version 1.2 using Jbuilder version 3. The development of the system was based on Jade Framework Version 1.2.

The reverse engineering process glimpses at the following four perspectives: User, Design, Architecture and Implementation. The 'user perspective' explains what the system offers, using use cases and state diagrams.

The user perspective is identified by understanding the system through the execution of the system. The 'design perspective' is described using the class diagram, interaction class diagram, and the components diagram. The design perspective is produced by classifying the Java implementation code into class diagram and the interaction diagrams are found based on the attributes of each class and how they are related to each other. We also used debugging techniques to trace the dynamics of object interaction [Wu99][Mitch93]. The component diagram is identified by grouping the classes, which relate to each other. Groups of classes that are independent of each other are classified as components. The 'architecture perspective' describes the architecture of the system, whereas the implementation perspective discusses in detail how to implement the system. The architecture perspective is identified based on the execution of the system.

### B-1.1 Reverse Engineering of Newsfilter Application

*Identifying user requirements.*

Identifying user requirements was needed because the system did not provide clear documentation of what the system has offered. The requirements of Newsfilter were found through the execution of the Newsfilter application.

The execution of Newsfilter provides the user with an interface through which the user can interact with the system. The menu and sub-menus provided describe what the system offers. The state diagram for each menu is developed by comparing the action through the menu with the actual implementation code. The state diagram describes the event and process offered by the system. Figure B1.1 shows a state diagram for the file menu.

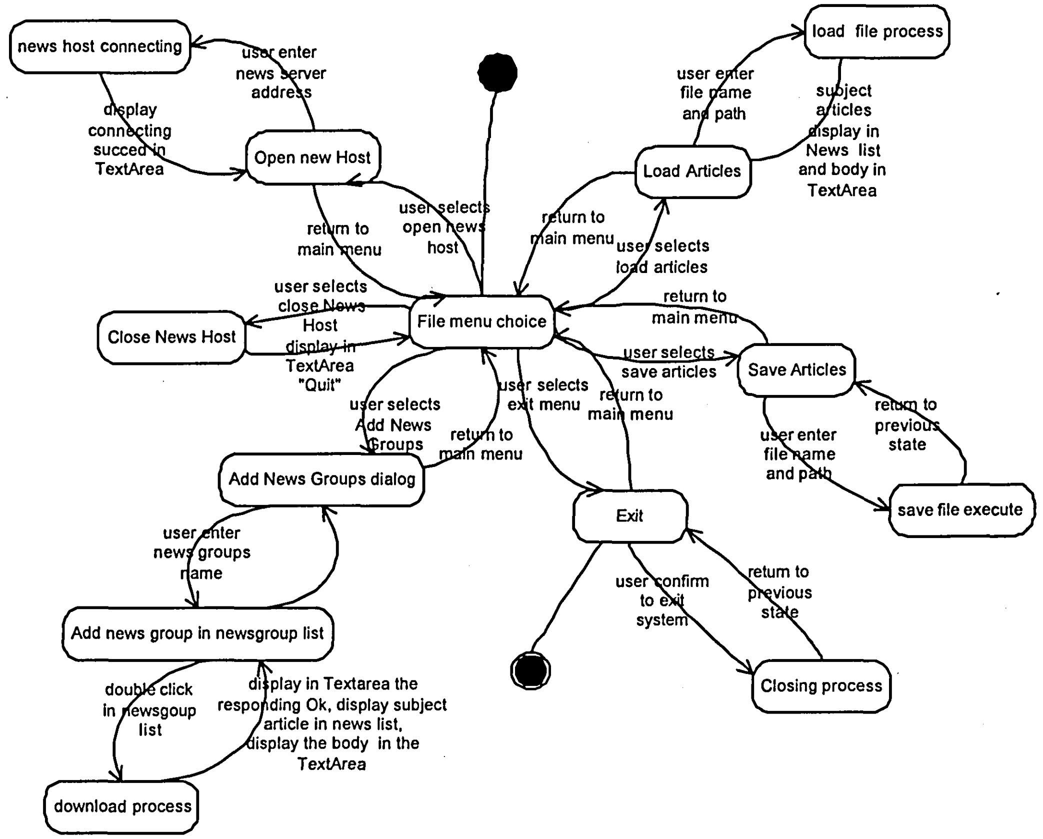
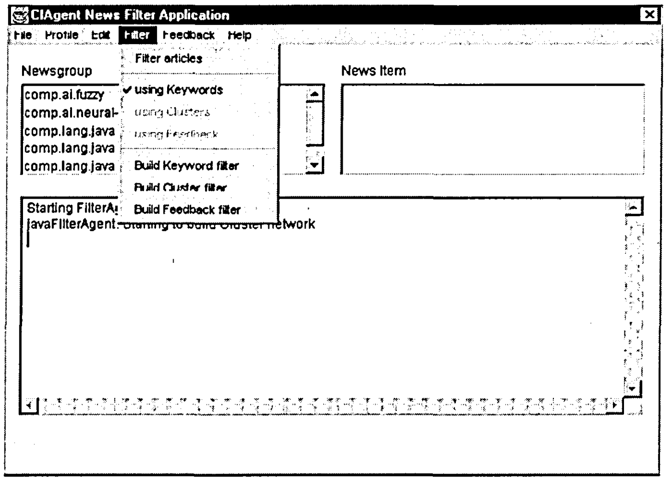

news host connecting
user enter news server address
display connecting succeed in TextArea
Open new Host
return to main menu
user selects open news host
File menu choice
user selects close News Host
Close News Host
display in TextArea "Quit"
user selects Add News Groups
return to main menu
Add News Groups dialog
user enter news groups name
Add news group in newsgroup list
double click in newsgoup list
display in Textarea the responding Ok, display subject article in news list, display the body in the TextArea
download process
user enter file name and path
load file process
subject articles display in News list and body in TextArea
Load Articles
return to main menu
user selects load articles
return to main menu
user selects save articles
Save Articles
return to previous state
user enter file name and path
save file execute
user selects exit menu
return to main menu
Exit
user confirm to exit system
return to previous state
Closing process

Figure B1.1: The state diagram for file menu.

### *Developing Use Case*

Understanding of what the Newsfilter application offers and how to operate the system provided us with a starting point for producing the design document. First, the requirements were transformed to produce use cases for the system. The GUI(Graphical User Interface) presentation during the execution of the system was used to identify the use case model. The GUI presentation consists of four main menus: related to file, user model, filtering and feedback.

Each menu consists of sub-menus. The Figure B1.2 shows the user interface for the Newsfilter application.



```
Starting FilterA
javaFilterAgent: Starting to build Cluster network
```

FigureB1.2: The User Interface for Newsfilter Application

The use case was developed from the state diagram and execution of the system. The state diagram shown in Figure B1.1 is developed from the user interface of Figure B1.2. It shows that the Newsfilter application consists of four main menus: file, profile, filter and feedback. But these four menus do not directly transfer as use cases of the Newsfilter application. This is because each menu provides several other functions. The execution of the system describes how the system is used. The understanding of how the system is operated allows one to identify the use case of the system. These two techniques are carried out in parallel allowing us to identify the use cases of the Newsfilter applications. We synthesise that the use cases are: Create user model, Download articles using keywords, Filtering process (include display the result) and feedback. The adjacent level use cases are portrayed in the technical report. The use case shown in Figure B1.3 shows that the user has to interact with the system many times in order to achieve the system goal: set-up the user profile, select a newsgroup address to download articles related to the keyword and then provide feedback according to the results given, based on the user profile.

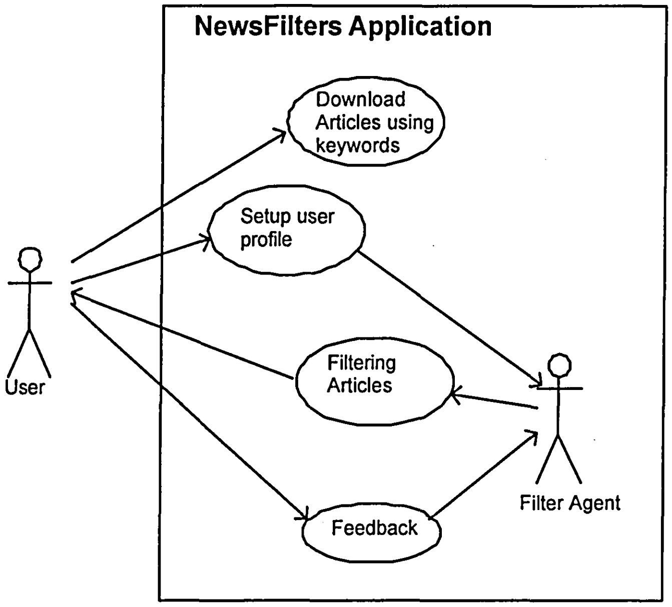


Figure B1.3: Use Case for Newsfilter Application.

*The Design Perspective*

The design of the Newsfilter is extracted by looking at each class of the Newsfilter application. It consists of seventeen classes. Table B1.1 shows the Java classes of the Newsfilter application as implemented. After investigating each class, we grouped the classes that are related to each other, and found out the dependency and links in each class. The process ended up with production of the class diagram, as shown in Figure B1.4. The classes are abstracted into five components. The table below shows the component names associated with the Java classes.

| Abstraction | Name of Java classes and data files | |
|---|---|---|
| Main User Interface | NewsFilter | |
| | NewsGroupDialog, KeywordDialog, QuitDialog and AboutNewsDialog | |
| NewsArtiles | NewsArticle | Newsarticles. dat |
| Agent | FilterAgent.java, Userprofile.dat | |
| | CIAgent, CIAgentEvent, CIAEventListener | |
| Roles | KMapNet.java | |
| | BackProp.java | |
| Supportive | DiscreteVariable, Variable, ContinuousVariable, DataSet | |

Table B1.1: The Java Classes of the Newsfilter Application

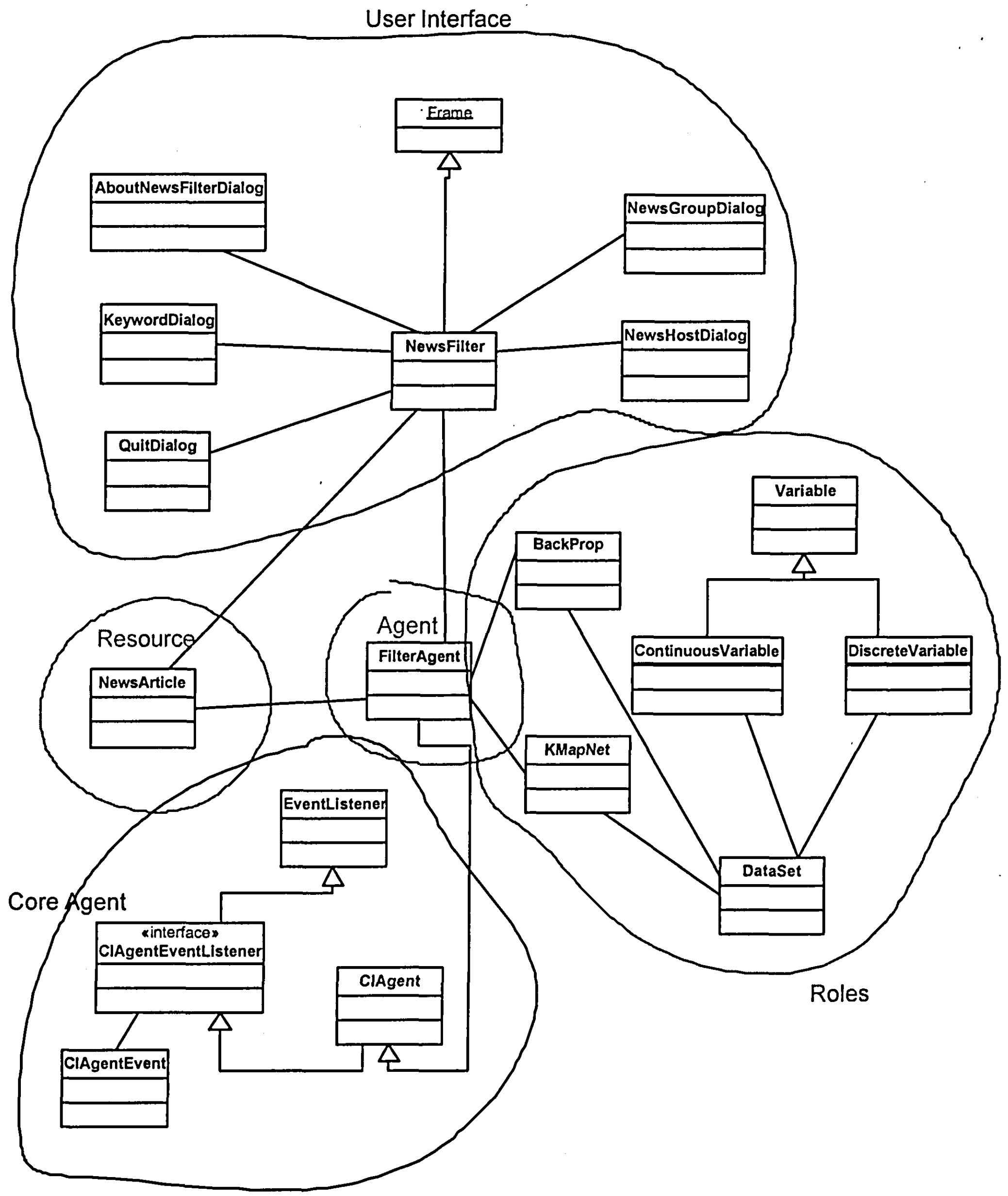


Figure B1.4: Static Class Diagram of the NewsFilter Application

The description of each component is discussed as below:

- Main user interface

The Newsfilter class is a main Java class that handles the user interaction of the whole system, which is associated with the five following classes that are used for dialogue between user and the system: AboutNewsFilterDialog, KeywordDialog, NewsGroupDialog, NewsHostDialog and QuitDialog. These four classes are defined as dialog objects because they interact with the user, while the Newsfilter class is defined as a frame object. It is used to present the graphic user interface as shown in Figure B1.2.

- NewsArticles

The NewsArticles class is used to define all information about a news article. NewsArticles uses the Java language components of java.awt.*, java.io.*, java.util.* and java.net.*. Java.net.* provides functionality for connecting to the Web and store the articles in Newsarticles files.

The FilterAgent class is responsible for providing all functions related to the management of user profile data, the keyword scoring and the construction of the neural network models used for the cluster and feedback filters.

- Agent

The FilterAgent Java class is portrayed as an agent. The next row associated to FilterAgent class is a group of classes that an abstraction of presenting the FilterAgent as an agent. The abstraction group of class is to present the autonomous and reactive behaviour as describes as core agent.

- Core Agent

The core agent is a component used to represent an object class as an agent. In this case, it is able to listen for any event coming from the user or other external objects and react accordingly. This mechanism is presented by defining the CIAgent as the interface with Evenlistener and EventObject. The FilterAgent is provided with autonomous behaviour, so it is able continuously to listen and capture events and react accordingly. The CIAgent class uses the following Java components: java.util.*, java.awt.* and java.beans.*. Java. beans.* provides functionality to the CIAgent be able to be run independently.

From here we can see only one agent is identified in this system, called FilterAgent. In this case the core agent is named CIAgent. This means the core agent is identified as the agent paradigm of the Newsfilter application solution. So, FilterAgent class inherits the CIAgent and is defined as a CIAgent type of agent.

- Roles

There are two roles assigned to FilterAgent. There are two abstraction of roles identified: BackProp and KmapNet. Each of them is defined as a class associated with FilterAgent to represent the neural network models of the learning algorithms of the back-propagation techniques and kohonen map techniques respectively, while the other classes shown in the next row in the table are used to define the neural network data abstraction for both techniques.

- Supportive

The rest of the classes as shown in the table that support the roles, are identified as supportive component. The ContinuousVariable class provides the necessary variables that take a continuous real value ranging from some predefined minimum to a maximum value, whereas the DiscreteVariable class takes a predefined set of numeric or symbolic values.

Both of these classes are used as a part of a DataSet class, which uses defined multiple data sets for the three learning techniques of Kohonen Map, BackProp and Decision Tree. It defines the data set to read data from a flat text file into memory and it is also used to display to the user, and for trace information. The Variable class is a base class for all variables that support the function of rule processing and learning in the BackProp and feedback methods.

A detailed explanation of the responsibility for each class, not relevant in this context, is given in the technical reports [Othm+99]. The static class is not enough to show the dynamic behaviour of the system. The dynamic behaviour of the system is portrayed using a sequence diagram. The UML sequence diagram is developed using the use case and state diagram, and proved by tracing the state or the value of the object during execution of the system as presented in [Othm+99].

*Design Agent Environment*

The Newsfilter system is executed based on a Java environment. Thus, no intermediator languages are developed to support the execution of the NewsFilter system. However, the 'core agent' is a reuse component to define the agent. It integrates with the Java environment which enables the FilterAgent to agent behave autonomously and reactively.

### B-1.2 Reverse Engineering the Filtering Application

The objective of the Filtering application is similar to the Newsfilter application and the design solution is also quite similar, that is to retrieve information from the Internet that most matches the user's interest. This is because both the amount and variety of information change rapidly over the time. Both systems, offer two stages of filtering: first filtering based on keywords while downloading articles from the Internet and second, filtering the downloaded articles based on a user profile. However, they are different in detailed requirements. The Newsfilter uses the kohonen map and back propagation learning techniques of machine learning in user profiling, whereas the Filtering application uses a genetic algorithm learning technique. The agent paradigm used for the Newsfilter solution and Filtering application are different. The Newsfilter system uses a single agent approach to solution. The Filtering application uses an autonomous reactive agent paradigm of solution, but the solution is based on a multi-agent system approach.

It provides the criteria of distribution, communication and element of openness for future direction. It is a multi-user based system and users from anywhere can use the same system as a real time basis. Therefore the reverse engineering result presents a different kind of modelling of needs.

This section discusses the result of reverse engineering of the Filtering Agent System. The result is analysed from the Filtering Application implementation code to produce user requirements, and design agent modelling. This section only discusses part of the reverse engineering result rather than providing the full result of reverse engineering. The complete reverse engineering is documented in the technical report [Othm00].

*The User Perspective.*

The user perspective is identified through the GUI presentation displayed while executing the system. The execution of the system provides an understanding of how to operate the system. We discovered that the GUI presentation clearly defines use cases: login system, provide keywords, display the article ranking and update the user model. The use cases as presented in the GUI interface as shown in Figure B1.5 This top use case is identified according to the GUI presented in Figure B1.6 and Figure B1.7.

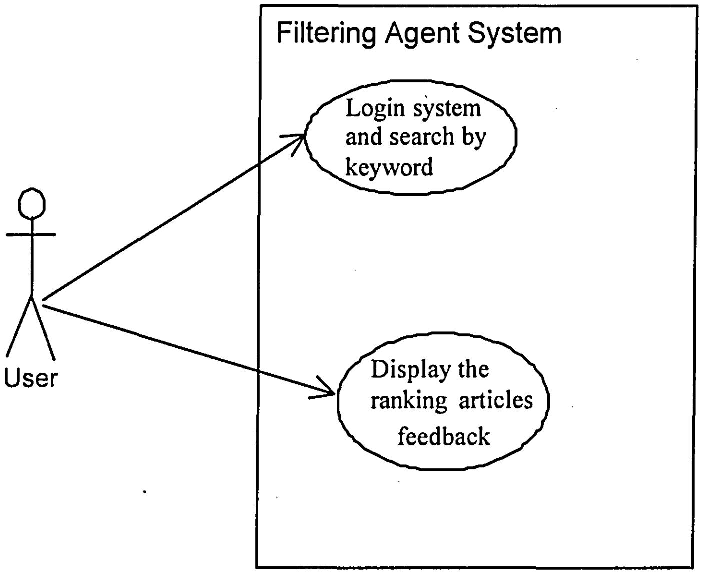

Filtering Agent System
Login system and search by keyword
User
Display the ranking articles feedback

Figure B1.5:The Use Case for Filtering Agent System.

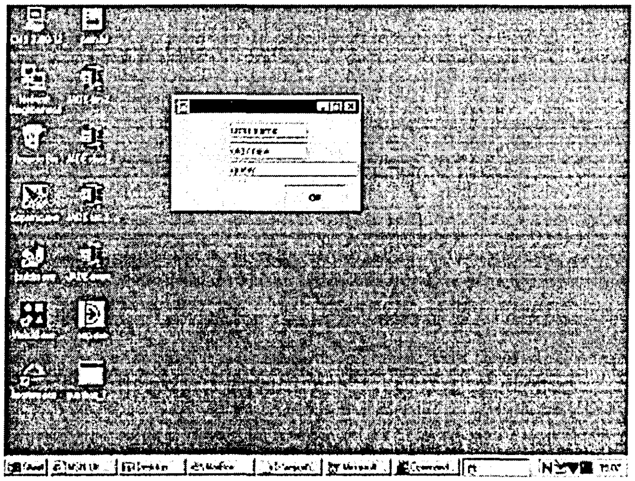
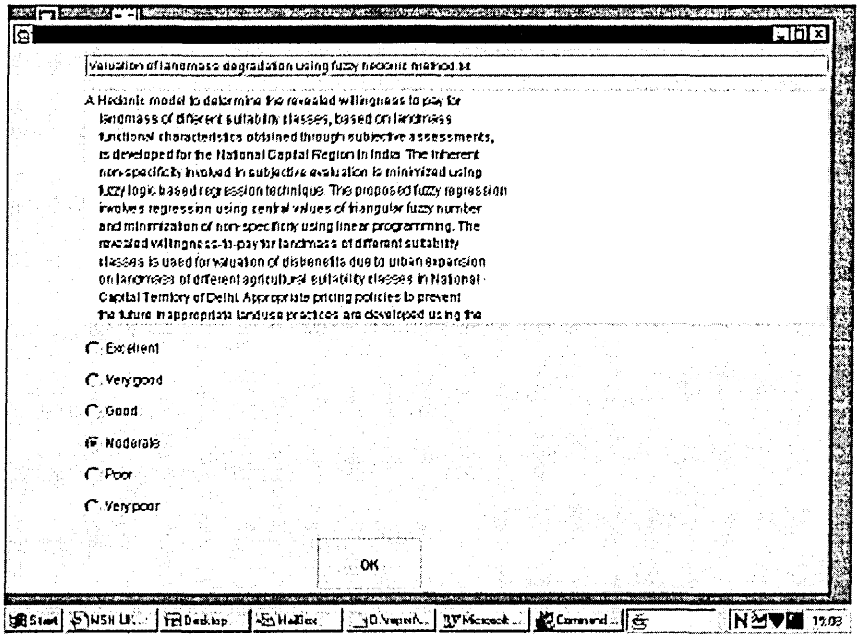


Figure B1.6: The GUI User Interaction for Login and Set Keyword Use Case

The GUI presentation clearly defines each use case for each GUI user interaction. For example, the first use case is for login on to the system and selecting keywords to search for articles, is associated with the first GUI user interaction.

Figure B1.7 shows how the GUI displays the title of ranked articles according to the user preferences. The first text box area displays the title of the articles and the second text box displays the content or abstract of the articles. The level of interest of articles according to the user preference is marked. This user interface allows the user to assess the level of interest of articles and update the user profile.



Valuation of landmass degradation using fuzzy hedonic method.tx

A Hedonic model to determine the revealed willingness to pay for landmass of different suitability classes, based on landmass functional characteristics obtained through subjective assessments, is developed for the National Capital Region in India. The inherent non-specificity involved in subjective evaluation is minimized using fuzzy logic based regression technique. The proposed fuzzy regression involves regression using central values of triangular fuzzy number and minimization of non-specificity using linear programming. The revealed willingness-to-pay for landmass of different suitability classes is used for valuation of disbenefits due to urban expansion on landmass of different agricultural suitability classes in National Capital Territory of Delhi. Appropriate pricing policies to prevent the future inappropriate landuse practices are developed using the

○ Excellent

○ Very good

○ Good

● Moderate

○ Poor

○ Very poor

Figure B1.7: The GUI User Interaction for Display Articles and Update User model.

*Design Perspective.*

The design perspective is divided into two phases: agent system design and agent environment design. The reverse engineering processes clearly shows the to separation of the two design process. In this section we focuses on the reverse engineering process using software abstraction, while in a later section we transform the reverse engineering abstraction result into the terms of the designing agent system and designing the agent environment.

*Agent System Design*

The first step is to distil the abstraction into design agent system stages and modelling in the stages through investigating the implementation codes of the Filtering application. The Filtering application consists of 27 Java files. These Java files are grouped into five groups as shown in Table B1.8.

| Agent Abstraction | Classes name |
|---|---|
| UserAgent | UserAgent |
| | UserLoginGUI, ShowingResultGUI, AbsoluteConstraints, AbsoluteLayouts<br>ShowingResultBehaviour, UserLoginBehaviour<br>Userlogin.dat |
| SearchAgent | SearchAgent<br>SearchBehaviour, Find_dir |
| DocumentAgent | DocumentAgent<br>IndexBehaviour, Parse, Document.dat |
| FilterAgent | FilterAgent<br>RankBehaviour, Sim_cal, myVector |
| UserModelAgent | UsermodelAgent.<br>UserprofileBehaviour, DispatchBehaviour, RFBehaviour, init_profile,<br>GABehaviour, fuzzy_main, Term_data, GAmain, R_feedback, Usermodel.dat |

Table B1.8: The Java classes of Filtering Agent System

The grouping criteria are based on dependency between the classes and how these classes are linked between each other. We identified two methods by which classes are linked with each other using *communication act* abstraction and not using *communication abstraction.* The *communication act abstraction* is the use *ACLMessage* objects for communication. We identified five Java classes linked together using abstraction. The Java classes are UserAgent, SearchAgent, DocumentAgent, FilterAgent and UserModelAgent. The other Java classes are linked using normal object-oriented dependency such as instantiate object or pass value using object method. The Java classes shown in Table B1.9 are grouped based on those five Java classes as identified above. The table shows the relationship between the abstraction name and the groups of the Java classes. The .dat type files are the data files.

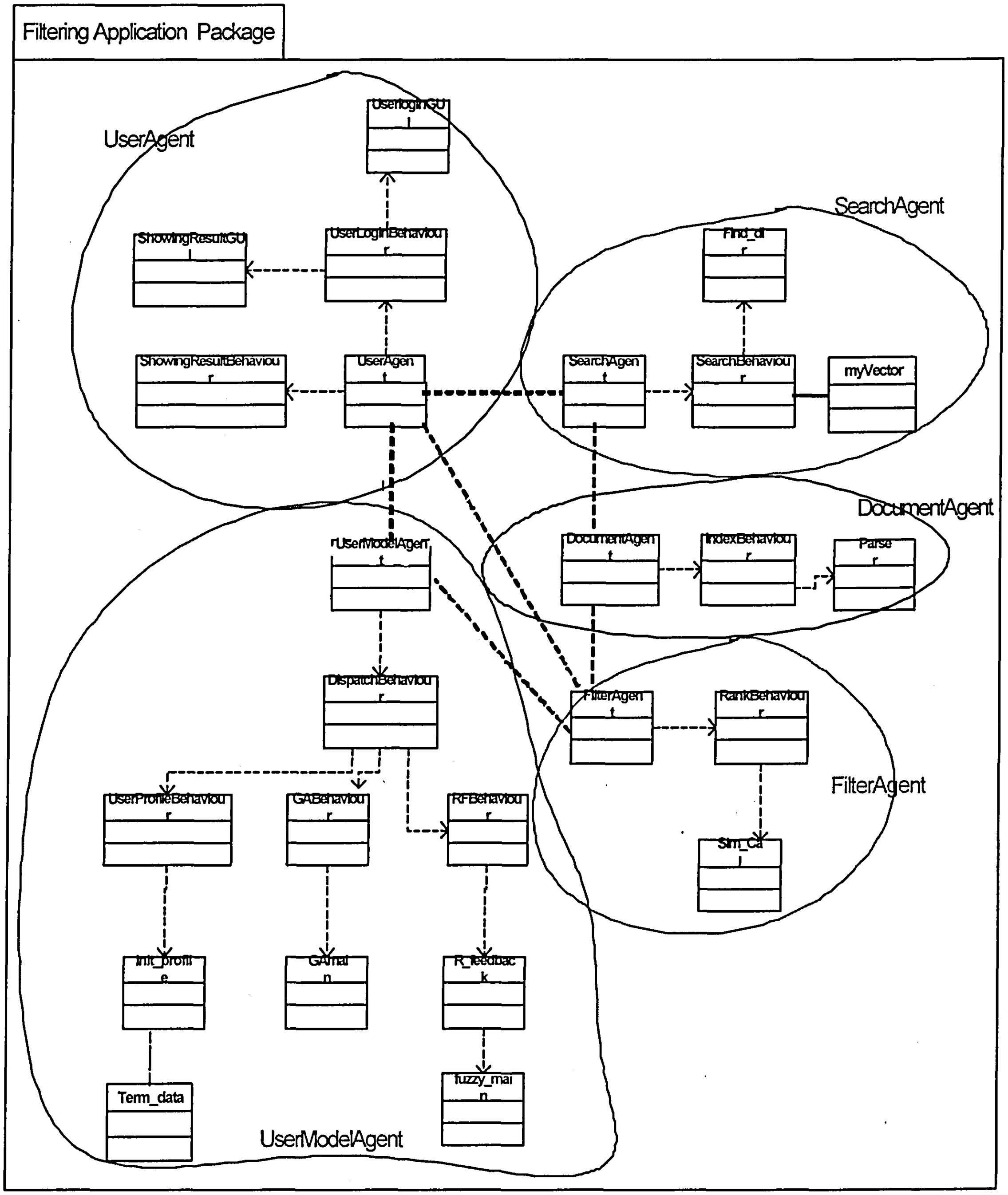


Figure B1.8: The Static Class Diagram for Filtering Agent.

After investigating the dependencies and links between all the Java classes, the process culminated in production of the static class diagram shown in Figure B1.9. The figure shows each group as an area bounded with a colour-line. The association among the classes in each group is represented as an agent. This means that there are five agents identified in the system. The five agents are UserAgent, SearchAgent, DocumentAgent, FilterAgent and UserModelAgent. The figure shows these five agents are linked with each other as shown by dark-dashed lines. The dark-dashed lines show the interactions between the agents.

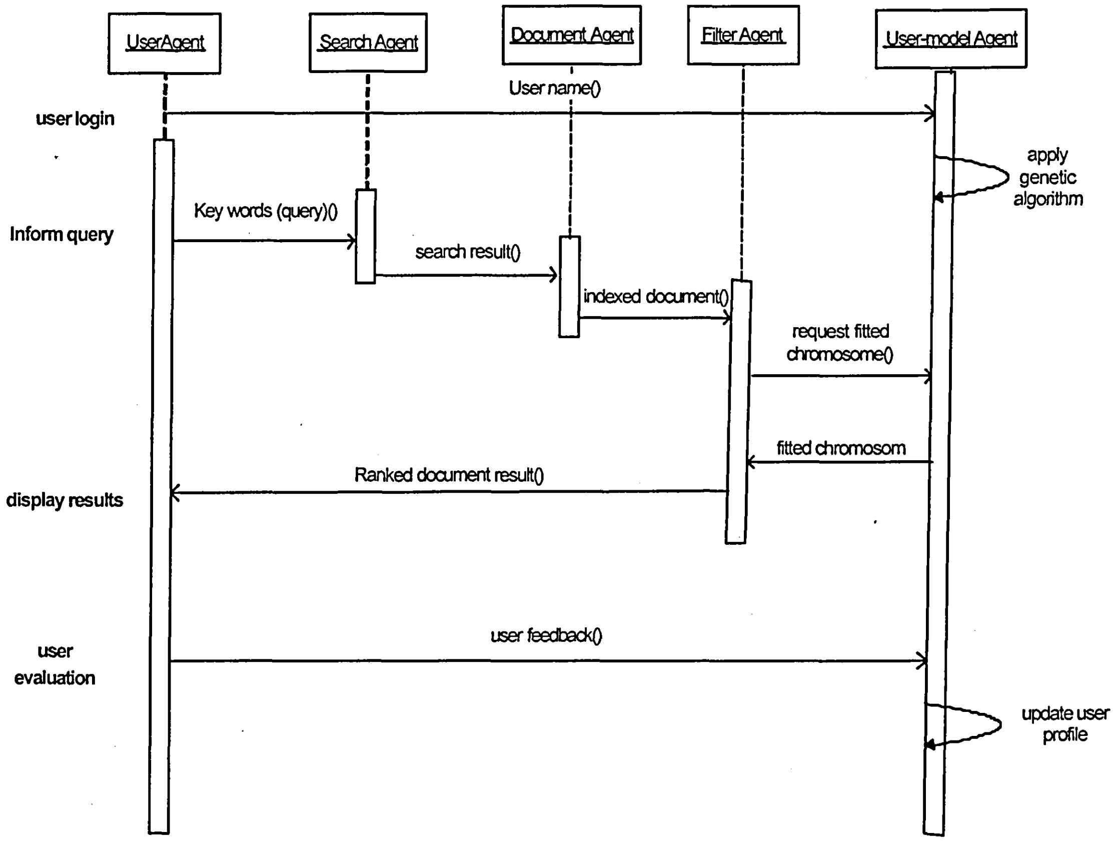


Figure B1.9: The Agent Interaction of Filtering Agent System

The second step is investigation through the implementation code of these five classes and debugging techniques while executing the system, to identify the dynamic diagram of the Filtering Agent System. The debugging mainly focuses on communication using the *communication act* abstraction only, which focused on interaction between agents. As a result, the sequence diagram of the system was produced, as shown in Figure B1.9. This sequence diagram shows the dynamic interaction between agents. It shows the flows of agent interaction when the UserAgent initiates login, make query and make feedback.

The above reverse engineering process shows the abstraction of the agent system design, while the next step is associated with agent design.

*Agent Design*

In the third step, the reverse engineering focused on each group of the agent abstraction. Referring back to Table B1.8, the classes in each group can be abstracted into five types of abstraction: user interface (dialogue), role, agent, resource and supportive. These abstractions are abstracted based on the types of objects identified as shown below:

- User interface – Jframe type of object.
- Role – CyclicBehaviour and OneShotBehaviour type of object.
- Agent – Agent type of object.
- Resource – Java class with .dat file type.
- Supportive – Java object type.

The class type is identified in the Java implementation code, in the form '*public class* UserAgent *extends* Agent'. For example the 'UserAgent' is the class name and 'Agent' is the object type. A similar process is performed to other types of objects as well. The resource abstraction is an abstraction of a data file associated with the group of abstraction.

The other classes that are linked with each other are defined as internal tasks or processes of each agent. The following shows the result of the abstraction of each group.

*The UserGroup.*

The UserAgent group consists the five abstractions as shown below.

| | |
|---|---|
| User Interface | – UserLoginGui, ShowingResultGui |
| Role | – UserLoginBehaviour, ShowingResultBehaviour |
| Agent | – UserAgent |
| Resource | – UserLogin.dat |
| Supportive | – AbsoluteConstrain, AbsoluteLayouts. |

*The SearchAgent.*

The SearchAgent group consists of three abstractions.

| | |
|---|---|
| Role | – SearchBehaviour |
| Agent | – SeachAgent |
| Resource | – The Web |
| Supportive | – Find-dir |

*The DocumentAgent*

The DocumentAgent group consists of four abstractions.

Role – IndexBehaviour  
Agent – DocumentAgent  
Resource – Document.dat  
Supportive – Parser

*The UserModelAgent*

The UserModelAgent group consists of four abstractions.

Role – UserProfileBehaviour,GABehaviour, RFBehaviour  
Agent – UserModelAgent  
Resource – UserProfile.dat  
Supportive – Init_profile, GAmain, Term_data, R-Feedback, Fuzzy_main.

The abstraction of each group can be illustrated using class diagrams. Figure B1.8 shows the abstraction of the UserModelAgent group. The four abstractions are shown by the colour-line (red label): agent (the top area), role (middle), supportive (the bottom) and resource (small round at the bottom). The resource abstraction is shown as a data type diagram in the diagram because it is not a class but rather a file data associated with the roles (shows in green colour). The UserInterface abstraction not shown is the figure because there is no user interface abstraction in the UserModelAgent, but is presented in the UserAgent abstraction. A similar process was performed for other groups as well, as presented in [Othm+00].

The fourth step was to identify the abstraction of each role. We continued by investigating the roles of UserModelAgent. This abstraction can be seen in Figure B1.10 as well. The diagram shows the relationship between the classes of roles abstraction and the classes of supportive abstraction as summarised below:

UserprofileBehaviour – init_profile.  
GABehaviour – GAmain.  
RFBehaviour – R_feedback, Fuzzy_main.

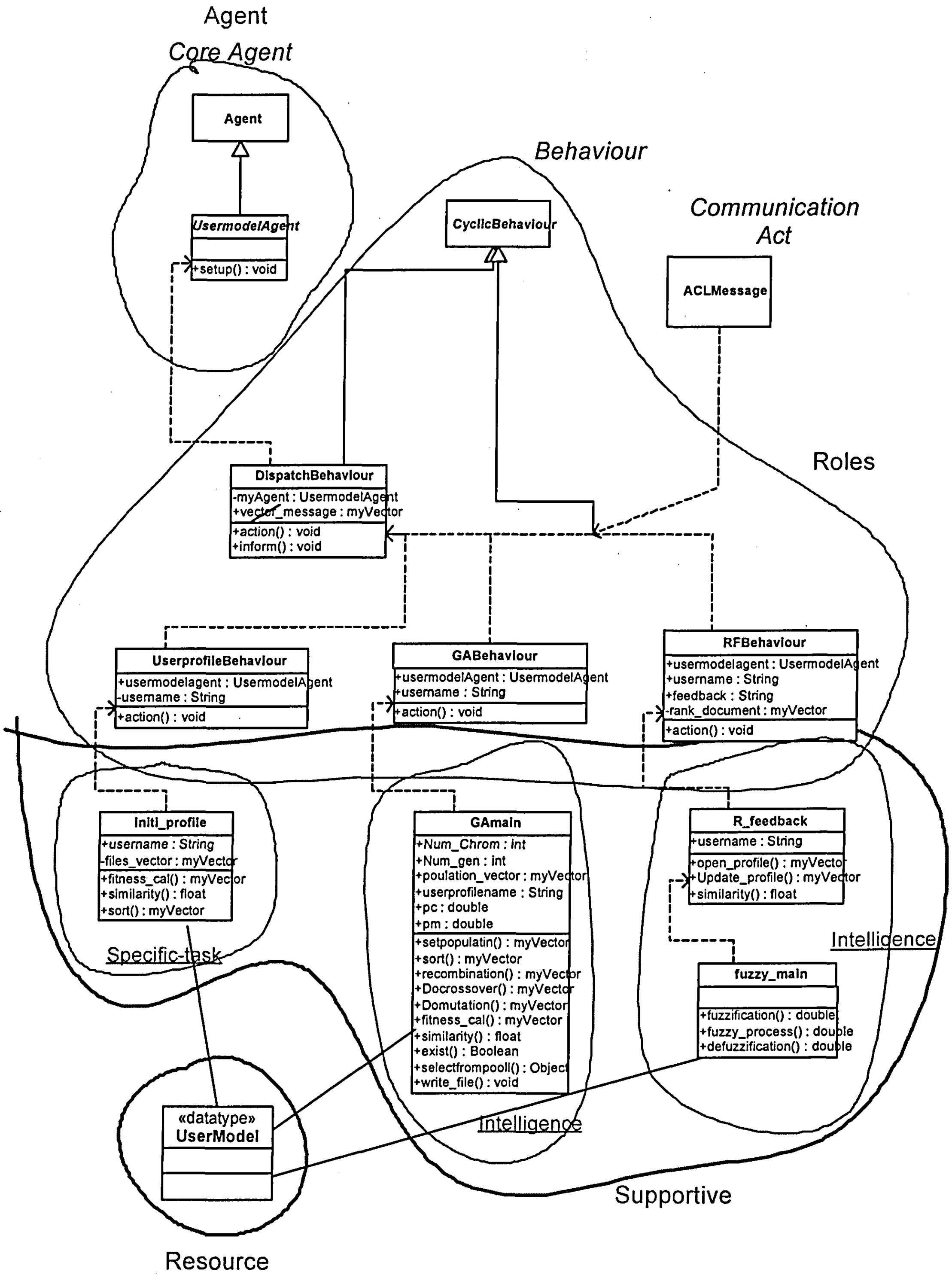


Figure B1.10: The UserModelAgent Class Diragram.

By investigation through the implementation code of the behaviour abstraction classes, we were able to abstract several types of abstractions. In these cases we identified two abstractions: *specific task* and *intelligence,* as the relationship as below:

Init-profile –presents as a specific task.

GAmain – presents as a specific task and intelligence together (provides a genetic algorithm).

R_Feedback – presents as a specific task while the user gives feedback.

Fuzzy_main – presents as an intelligence. It uses a fuzzy logic technique to identify the feedback relevant through the R_Feedback task.

These two abstractions are shown in the Figure B1.10 as well, in the area marked with a green underlined label.

There are several other abstractions that we captured such as external object, internal object, goal, and agent state. These abstractions are discussed in relation to the identification of an abstraction that was captured in the next step.

The five steps identified a communication abstraction. As a result of the abstraction process shows in step four, for all roles abstraction classes, we also identified three kinds of abstractions, which we defined in terms of communication, as described below:

- receiving event
- actions event

There are three types of receiving events: events from external objects, from other groups of abstraction ( i.e UserAgent, Search Agent) and from internal objects.

Examples of each type of receiving event are shown below.

- Event from external object :
  - event comes from the User i.e. in the UserLogin class of the UserAgent.
- Event from other agent abstraction group:
  - this type of event is captured in most role abstraction classes, for example GAbehaviour or Userprofilebehaviour in UserModelAgent receive events from UserAgent and FilterAgent respectively. This event is defined as receiving from another agent.
- Event from internal object:
  - changes of the state of agent as dead, alive, or busy, in all Agent abstraction.
  - changes of resource, e.g. mutation of the genetic algorithm changes the resource information and react to it, in UserModelUser.

On the other hand, two types of event actions are captured: internal action and using communication action. Examples of each type of action are shown below.

- Internal action:

  The internal action is associated with change in an internal object such as resource or agent's state or achieving agent goal. This internal action is performed by the specific task abstraction as discussed before.

- Communication Act action:

  The communication Act is mainly performed in two conditions. In the first, an action is performed as part of the achievement of the whole global goal or sub-goal of the system. This happens when the agent goal has been achieved, e.g. when all documents are indexed as the goal of DocumentAgent, it generates action to send the indexed documents to FilterAgent using communication act action.

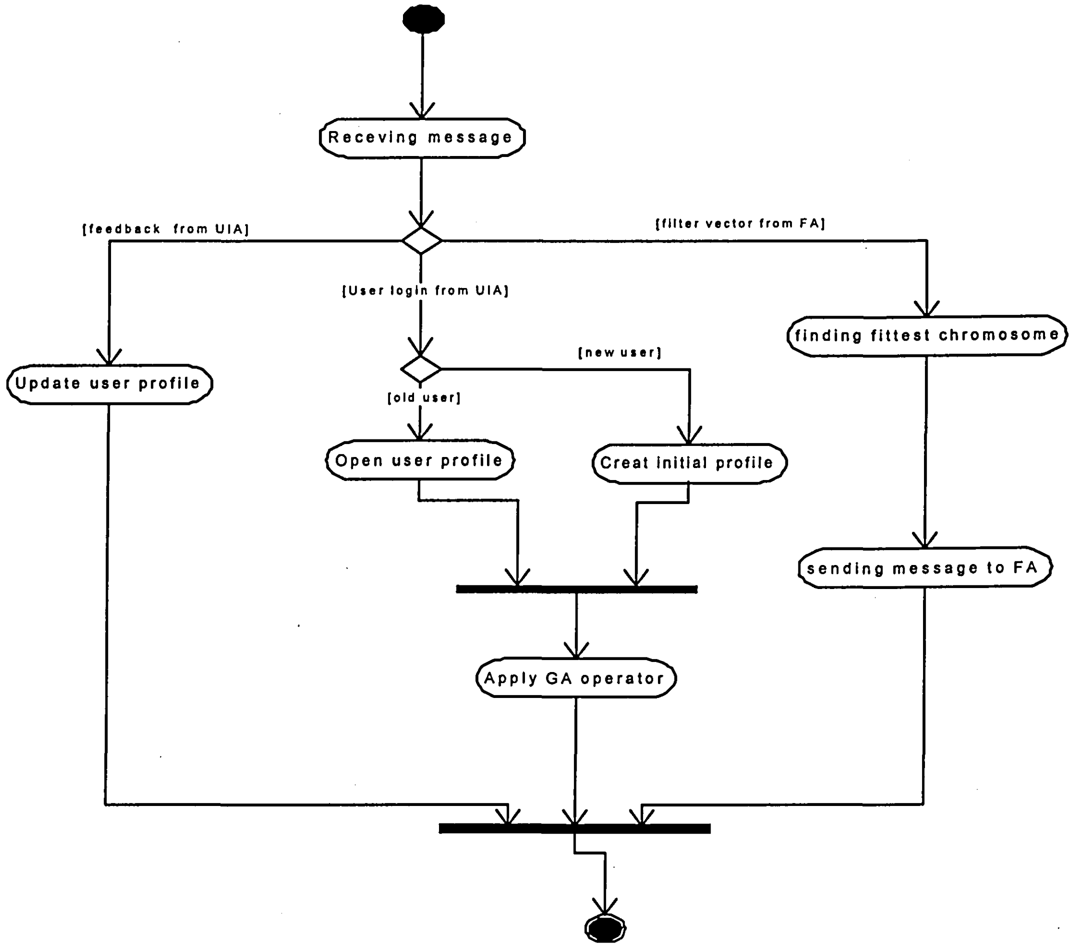


Figure B1.11: The UserModel State Diagram.

In the second, an action is performed in order to correspond to the previous receiving action. For example, UserModelAgent reacts when FilterAgent requests an accurate user profile.

The communication in an agent is associated with receiving events and actions can be modelled as shown in Figure B 1.11. The figure shows the UserModelAgent model and describes how each agent manages the receiving event associated with abstraction of roles, specific task and shows the actions are performed. The model uses the UML state diagrams notation. The model also shows the mechanisms for managing conflicting communications among other agents and shows mechanisms to perform the internal agent tasks.

*Design Agent Environment*

The end result of agent environment design is the identification of the components. The relationships among these components represent the architecture of the agent environment, which aims to provide the functionality needed to integrate with the agent system design into an executable agent system. This means the components are used in the implementation of the agent system. These components are reusable. Figure B1.10 shows some of the reusable components in the implementation of UserModelAgent, marked with a label blue italic, i.e. communication act, behaviours and core agent.

The investigation of the implementation code of each agent class shows the components are used to support agent system development, which are identified using the 'import' command. Two types of components are identified: components defined by the Jade framework and Java components as shown in Table B 1.3.

| Java Components | | |
|---|---|---|
| java.io.Reader; | java.io.StringReader; | java.io.StringWriter; |
| java.io.Serializable; | java.io.InputStream; | java.io.ObjectInput; |
| java.io.ObjectInputStream; | java.io.OutputStream; | java.io.ObjectOutput; |
| java.io.ObjectOutputStream; | java.io.IOException; | java.io.InterruptedIOException; |
| java.util.Date; | java.util.Iterator; | java.util.Map; |
| java.util.HashMap; | java.util.Vector; | |
| Jade Components | | |
| jade.core.behaviours.Behaviour; | jade.domain.FIPAException; | jade.onto.Name; |
| jade.lang.Codec; | jade.lang.acl.*; | jade.onto.Ontology; |
| jade.domain.AgentManagementOntology; | | |

Table B1.3: The Components used in the Agent class.

These components are associated with each other to present the Jade agent environment and are reused by the implementation agent. For example the DocumentAgent.java consists of the following components: jade.core.*; jade.core.behaviours.*; jade.domain.* and demo.MeetingScheduler.Clp.*, whereas the IndexBehaviour uses the following components: import jade.core.Agent; jade.lang.acl.ACLMessage; jade.core.behaviours.*; jade.lang.acl.*;

java.util.* and java.io.*. The Document Agent uses the jade.core.Agent component to declare it is an agent. It uses the ACLMessage component to communicate with other agents. The jade.lang.acl.* component is used to define all agent communication language commands and syntax (more detailed refer [Othm00]).

The Filtering application is executed using not only a Java environment but a combination of several environments, such as the communication environment and physical environment, as discussed later in chapter four.

The core agent, behaviour and communication act abstraction are linked with each other based on a specific requirement which describe the agent solution architecture (FIPA architecture). This means agent environment is developed based on FIPA architecture. The structure modelling as shown in Figure B1.10 also depends on the agent architecture.

In this section, the reverse engineering process identified elements for designing agent system and identified the needs of the agent environment design stage. The identification of the Jade components in the implementation code, shows that these components are needed to identify, design and implement it. These components are identified as an inter-mediator language that is used to implement the design of the agent system. The development of the agent environment is discussed in the next chapter. In the next section we will discuss what we learned from the reverse engineering process, leading to the identification of agent concepts, development stages and modelling.

## B-2: Reverse Engineering Process Towards Development Agent Environment

The reverse engineering process of the Newsfilter application and Filtering Agent System, as discussed in the previous chapter, shows how the components are used to develop an agent system. The NewsFilter application is developed based on the CIAgent framework while the Filtering Agent System is developed based on the JADE framework. We carried out reverse engineering for CIAgent to show the use of design patterns in the framework, while in the JADE framework, the reverse engineering was to identify the abstraction of components and relationship between the components as provided in the documentation. It is a part of the process of understanding the nature of agent environment development. Since the JADE framework provides documentation of the system development as well, the reverse engineering process was performed to gain more understanding of the implementation of the system and to identify the design patterns are used in the system. The following section describes each of the frameworks.

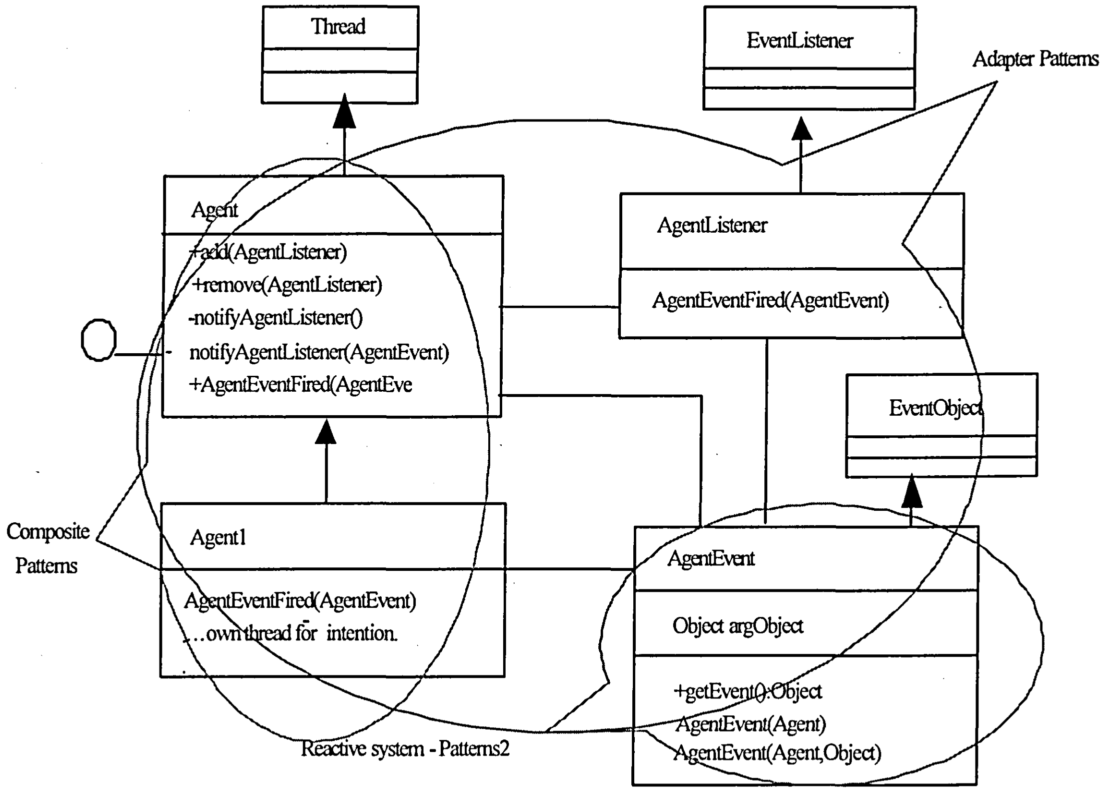
## B-2.1 CIAgent.

The reverse engineering of the CIAgent framework aims to show the use of design patterns in designing the CIAgent component as the implementation used the Java language. CIAgent is the name of a component that is reused in the NewsFilter application. The reverse engineering of CIAgent showed that the CIAgent component only presents the abstraction of the core agent as the agent paradigm for design of the Newsfilter application. If we refer back to the agent overview in Chapter three, this component is part of the agent capabilities. It does not present the criteria of distribution, communication and openness. This means CIAgent components use the Java language environment as the physical environment to implement the CIAgent architecture. Table B2.1 showed that the CIAgent architecture consists of three classes associated with each other to represent the behaviours: CIAgent, CIAgentListener and CIAgentEvent, integrated with the Java components. The following table shows the Java components used in the CIAgent framework.

| | |
|---|---|
| CIAgent.java | java. beans.*<br>java.awt.*<br>jawa.util.* |
| CIAgentListener.java | java.util.EvenListener |
| CIAgentEvent.java | java.util.EvenObject |

Table B2.1: Components use for Development CIAgent

CIAgent uses JavaBeans. The structure of generic CIAgent Javabeans is shown in Figure B2.1. JavaBeans is a component architecture for the Java programming language. JavaBeans is an OMG reusable software component that provides a specialised set of methods and fields that have generic methods through which the component can advertise the functionality it supports to the system into which it is loaded. This makes it possible to load complete dynamic objects. The CIAgent provides two components of CIAgentEventListener and CIAgentEvent. The CIAgentEventListener component is used to listen for any event that has occurred, whereas the CIAgentEvent captures the event object that has occurred that can react on it.



FigureB2.1: The CIAgent Component

The CIAgent architecture only provides the capability to capture the changes of the environment it is intended to observe. In the Newsfilter system case, *Textarea* is used as an object to identify any changes of agent stated and observed for any new articles that are downloaded. Development of a single agent environment follows the traditional development of a single object-oriented framework application, which consists of only one abstract class. This abstract class is used to define the capability of the agent. The Figure B2.2 shows how the abstract class of CIAgent is inherited by the FilterAgent class. The FilterAgent is an active object that inherits the CIAgent to present itself as an agent.

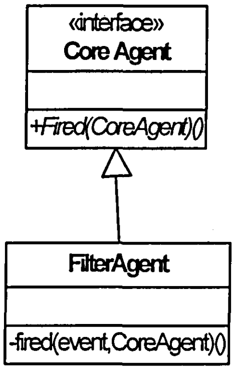


Figure B2.2: FilterAgent as an Active Object.

Figure B2.1 shows the class diagram of the CIAgent component to present the capability of an agent. It can be seen that it consists of several types of design patterns to provide capability of autonomous reactive behaviour. The adapter Pattern in [Gam+95] is identified as the external world, which is defined as an object that keeps changing and identifies the changes, but does not allow changes to other objects. The changes in the *textarea* are notified and this triggers performance of other actions.

The autonomous behaviour describes the agent as an active object. An Agent class is defined as a thread object. Adapter patterns and observer patterns [Gam+95] are used to provide the dynamic reactive behaviour. Pattern2 in the Reactive system by [Arid+98], describes how the agent event is objectified based on the location of the object. We can see here that the component to present the capability agent utilises the object-oriented design pattern.

Figure B2.2 shows a composite patterns of how an agent is created[Gam+95][Rieh97]. This method is a part of the deployment stage, where the components of the agent system design are deployed as the later stage of development process.

In the next section, we are able to identify other components built to present other capability and agent features for development of a multi-agent system environment.

### B-2.2 JADE

The reverse engineering of the Filtering Agent System reveals a similar process with the CIAgent, which resulted the identification of several components showed in Table B1.3. Two types of components were identified: Java based components and JADE components. The former components were identified based on the prefix, 'import java... ', while the JADE components show the prefix 'import jade... '. The JADE components represent the components defined in the agent environment.

The JADE framework is an executable semi-complete application to develop agents. It is an open source framework available from the Web. Figure B2.3 shows the organisation directory of the JADE framework as shown in the implementation code of JADE. This organisation directory is matched with the abstraction of JADE, as we captured in scrutiny of the JADE framework in the previous section.

The figure shows that the JADE framework is abstracted into four domains: core agent, agent management, communication and tools. In our view, the tools domain is not an aspect of the design of an agent environment but rather provides tools or utility for prototyping or testing the developed agent system based on the agent environment. Therefore, we focus only on the first three domains that influence the design of the agent environment.

The core agent describes an abstraction agent that is reused to create agents, so it mainly provides the functionality, as we identified in the agent overview, as shown in Chapter 3. The core agent abstraction consists of JADE core components and behaviour components. The JADE components are composed of several sub-components as shown in Figure B2.3.

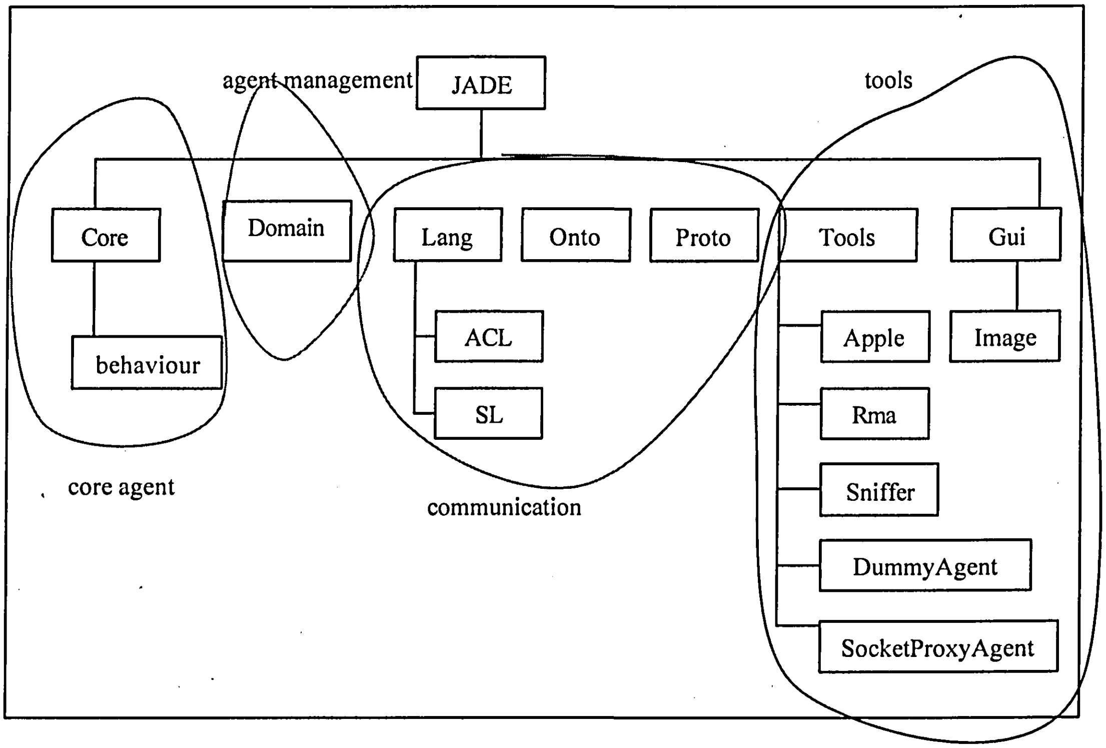


Figure B2.3: The Organisation of the JADE Framework

The Figure B2.4 shows the structure of class diagram of the JADE core components. It shows the sub-components that present the features of JADE agent and shows the relationship between the communication and agent management abstraction as well.

The JADE core components shown in the figure above are applied the design patterns as captured in the CIAgent components above. The serializeable components used allow many objects to run in parallel in their own thread, which allows several agents to run in parallel. The Commbroadcaster implements the Eventlistener, while the event is captured through the ACLmessage and AgentGroup, where all agents are subscribed.

The behaviour component consists of several types of behaviour components that can be reused to represent types of agent behaviour: *cyclicbehaviour, simplebehaviour, sequentialbehaviour, wakerbehaviour, senderbehaviour, receiverbehaviour, nondeterministicbehaviour* and *oneshotbehaviour.*

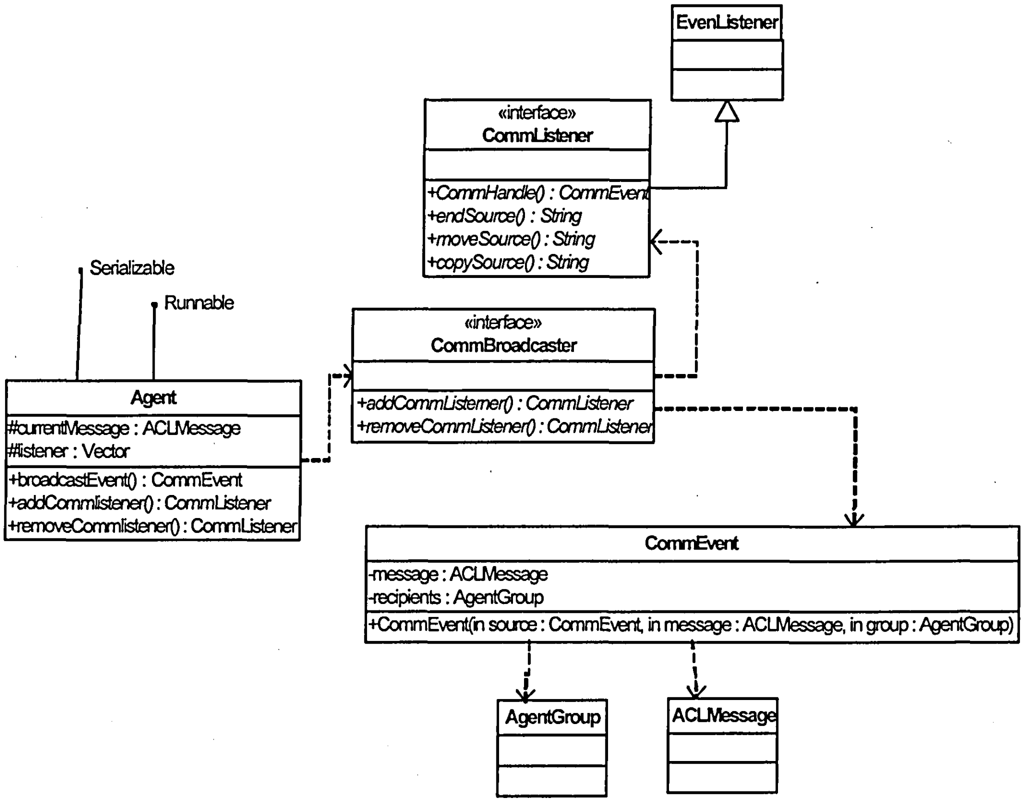

EvenListener
«interface»
CommListener
+CommHandle() : CommEvent
+endSource() : String
+moveSource() : String
+copySource() : String
Serializable
Runnable
«interface»
CommBroadcaster
+addCommListemer() : CommListener
+removeCommListener() : CommListener
Agent
#currentMessage : ACLMessage
#listener : Vector
+broadcastEvent() : CommEvent
+addCommlistener() : CommListener
+removeCommlistener() : CommListener
CommEvent
-message : ACLMessage
-recipients : AgentGroup
+CommEvent(in source : CommEvent, in message : ACLMessage, in group : AgentGroup)
AgentGroup
ACLMessage

Figure B2.4: The JADE Core Agent

The communication abstraction as shown in Figure B2.3 consists of sub-components of language, ontology and prototype and again the language component consists of sub-components of ACL (communication act) and SL(semantic language) components.

The agent management abstraction consists of domain component. The domain component is composed into several components that used to manage agent activities, state and interactions.

The scrutiny of JADE framework found that the JADE presented the criteria of distribution heterogeneous. The FIPA is the agent architecture for JADE. It is designed based on the architecture distribution pattern, which uses the CORBA architecture. CORBA services and CORBA management are used to present the criteria of openness in the agent environment. The directory service follows the domain service. Details of the CORBA reference model can be found in [Mow+97]. In the next section we will discuss how architecture pattern and design pattern play a role in designing an agent environment.

# Appendix C: The Scrutiny of Agent Frameworks.

## C-1: JAFMAS

JAFMAS was developed by Deepika Chauhan, University of Cincinnati, and the code implementation is available at [Chau97][Jaf97]. JAFMAS is a multi-agent environment. The architecture is based on COOL agent architecture [Cool99], but has limitations due to being implemented in Lisp language. JAFMAS is aimed at representing and developing cooperation knowledge and protocol in a multi-agent system.

### *Agent Environment Requirement Attributes.*

Based on understanding of the JAFMAS, we compare the JAFMAS attributes against the four required agent characteristics. These four characteristics describe the agent abstraction of the JAFMAS agent.

- The *agent abstraction* of JAFMAS is based on agent name, goal and services. The detailed agent abstraction is discussed later as agent architecture.
- *Agent Behaviour-* autonomous and reactive. This mean all agents created have these minimum behaviours. The intelligence of the system is provided by modelling the interaction between agents in the form of conversation as an agent's plan to achieve some goals.
- *Communication-* focuses only on communication with inter-agents through conversation using speech act language as basic communication.
- *Distribution-* provides distribution locations of homogeneous agents. This means that the agent interaction is in a similar environment. JAFMAS uses an organisational conceptual perspective in which high-level agents control agents below them.
- *Openness-* Even though it uses speech act for communication, it does not meet the openness criterion. On the other hand, the communication protocol is limited through UDP socket using RMI functionality exhibits in Java Languages.
- *Service Openness-* does not provide open services; rather it is used among agents in the agent environment community.

### *JAFMAS Physical Environment Requirement:*

- *Accessibility*- uses techniques subscribed through a group depending on agent services. So all agents are registered and disseminated by agent name and group.
- *Deterministic*- Agents determine the changes of environment through the communication event while communicating and depend on the agent stated.
- *Controllability*-none
- *Volatility*- no change in environment.
- *Temporality*- episode

### *The Three layers of the Physical Environment:*

- *Application layer*- provides all agent management including agent directory by subscription into listener list, security based on Java security, query using operator and conversation operator interface and defines the interaction protocol as described in agent communication environment requirements.
- *Communication and transportation layer*- provides two basic modes of communication: direct communication and broadcast mode. Direct communication uses an RMI mechanism through a TCP/IP transportation mechanism that supports transports and services, and multicast uses UDP datagrams.
- *Physical layer*- is the hardware communication and follows the standard OSI communication protocol.

JAFMAS execution is based on a Java JDK 1.1.5 environment.

### *JAFMAS Communication Environment.*

The following describes the requirements of the communication environment for JAFMAS.

- *Communication*-provides two basic modes of communication: direct mode communication and broadcast mode.
- *Interaction*- collection of actions where agent action or decisions will be influenced by the presence and knowledge of other agents. Agent messages can convey information about a sender agent using the knowledge base, concerning its goal, its intention or desire to the receiver. There are two types of messages: declaration messages, which allow agents to declare their presence and capability; and, content message; which send part of the agent knowledge-base.

- *Coordination* - using organisation level. Agent has plans containing provisions for interaction with other agents. Agents can plan conversation with other agents through the agent groups. During conversation, agents exchange messages according to mutually agreed conventions, change state and perform local action.
- *Interaction protocol*- provides contract-net and specification sharing. Communication protocol uses speech-act language. The developer has freedom to choose any set of performatives in applications development. For example in the N-queens problem it is necessary to use three basic performatives: propose, accept and reject.

The following describes the techniques used to provide efficient agent interaction.

- *Interaction Management*- not provided because there is only one type of interaction protocol provided.
- *Language processing and policy*- apply speech act but there is no mention using a parser.
- Social enforcement services control by the social group.
- Social differentiation based on groups of agents. Agents with similar services are grouped in the same group.
- No social order.

***Organisation of agent system***

The architecture of the agent system describes the pattern of agent interaction as shown below. JAFMAS is developed based on an organisational perspective. Each agent has services. Similar types of services are grouped the same agent group.

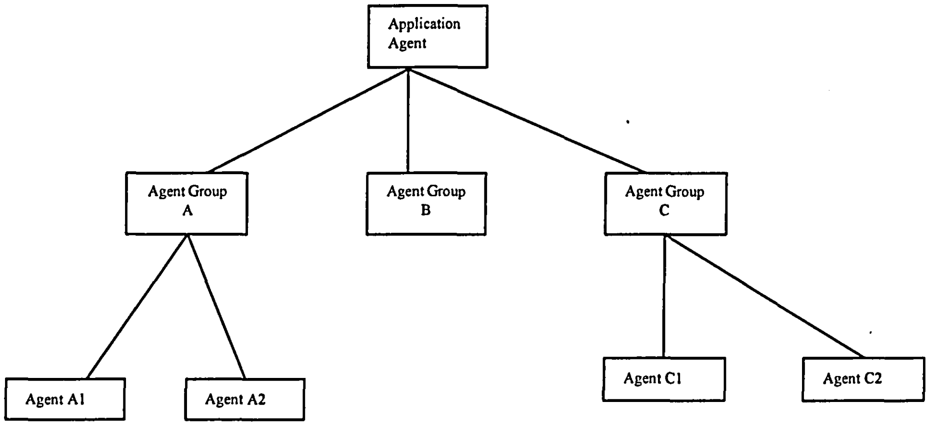


Figure C1.1: Organisation approach of JAFMAS agents system.

***Architecture Styles used in JAFMAS***

JAFMAS uses a combination of the following architecture styles :

- Client/ server architecture style, but the agent can change roles between of being a client and a server.
- Initiation of the system through multicast using multi-cast socket to UDP datagram (a specific communication protocol of an Internet standard network layer, transport layer and session layer protocols).
- Layered system architecture to present the communication environment. JAFMAS uses the standard layer communication protocol, in which the lowest level refers to hardware connections, while the upper layer consist of the JAFMAS operating system layer, as discussed below.

***JAFMAS Architecture Layer Approach.***

JAFMAS consists of seven layers of architecture. First, a multi-agent system application that describes the domain agent interaction among agents. Second, a social model which provides the functionality of the agent to socialise through coordination and conversation mechanisms. Third, a linguistic layer, which declares that speech act language is used. Fourth, provision of a basic communication protocol through TCP/IP using direct communication and multi-cast.

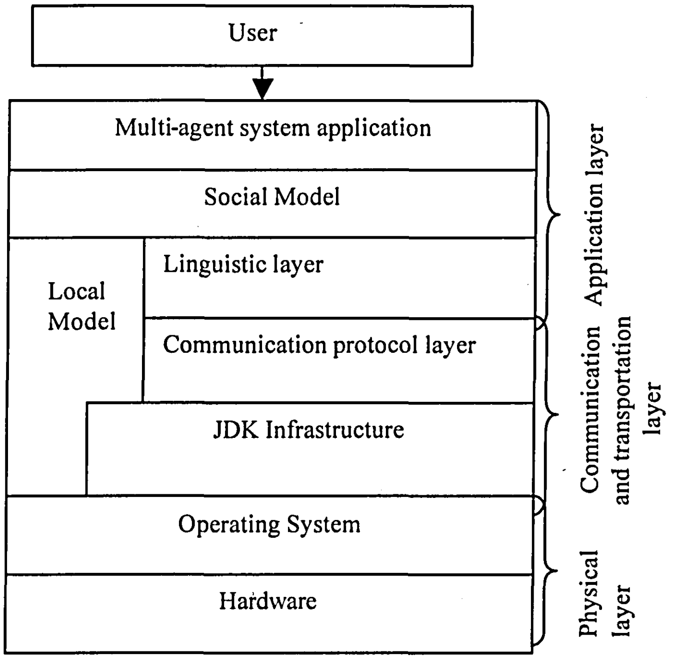


Figure C1.2: JAFMAS layer architecture

Fifth, the JDK infrastructure describes the requirements of the agent execution environment. Sixth, the operating system such as windows 95, windowNT, etc. and seventh, hardware which describes the hardware communication among agents.

***JAFMAS Internal Agent architecture.***

The internal agent architecture describes the architecture of each agent, in terms of how the social model and communication model are used for each agent in order to describe communication with the external world or other agents. The agent architecture shows how an agent communicates with other agents through direct communication and also through multi-cast using network UDP socket as shown in Figure Appendix C1-3 below.

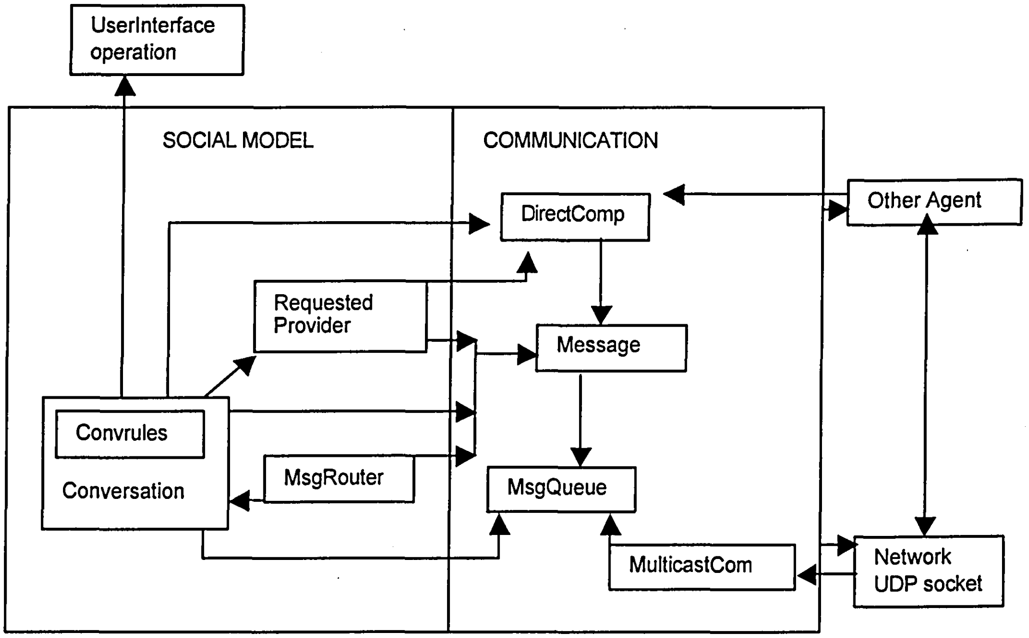


Figure C1.3: JAFMAS Internal Architecture.

***JAFMAS Component Structure.***

Figure Appendix B1-4: shows the structure of the JAFMAS environment. The JAFMAS agent class is the 'core abstraction' described agent, which has the ability to interact through direct and multi-cast communication. It also has a message router and requested resource provider that allows the agent to socialise and a conversation list that describes the plan for conversation between agents.

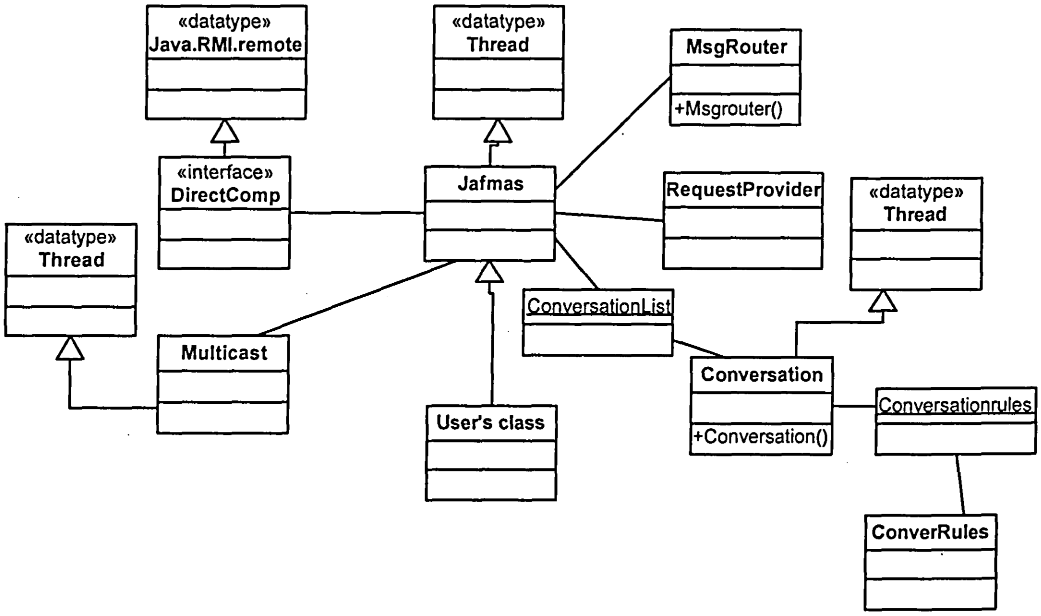


Figure C1.4: JAFMAS component structure.

## C-2: JATLITE

JATLITE was developed by the University of Stanford[Frost+96] and the implementation code of the JATLITE environment is available at [Jat97]. JATLITE is mainly used for internet agents. There are stand-alone agents (Java and C++) or applet agents exchanging information through a WWW browser. The JATLITE requirements are extracted and described according to the requiredattributes of the agent environment below.

*Agent abstraction-* The abstract agent describes agent as a Java applet or standalone agent( Java and C++), or wrapper agent for a legacy system. JATLITE framework only provides an agent communication environment. Its not specify any type of agent architecture, but the extension of JATLITE named JATLITEbean as discussed later is specifies the agent abstraction.

*Behaviour* - Agent behaviour only focuses on reactive behaviour. Autonomous behaviour is described in the statement that the agent stays connected to the server or keeps doing its task until it achieves its goal.

*Communication* - Only focuses on highly robust communication among agents, mainly for heterogeneous agents. JATLITE supports direct communication and broadcast through a message router. The communication is based on one to one communication using KQML agent communication language.

*Distribution*- provides distribution locations for agents anywhere in the similar agent environment.

*Openness*- agents have several choices of communication protocol to communicate with each other. The three choices of communication protocols are FTP, HTTP or SMTP.

*Service Openness* - JATLITE provides open services. Any agent can launch its services as green pages. Any agent can communicate or use the services as long as it has permission to access the agent launched using a password.

***Agent Physical Environment***

The following describes the requirements of the physical environment for Jaflite agents.

- *Accessibility*- all agents have access to subscribe into a router server as provider. While subscribing to the provider, all agents provide unique identification by name, address (based on where the agent is located) and password.
- *Deterministic*- It determines based on internal agent that initiate to communicate or initiate through communication from.other agents.
- *Controllability*-none
- *Volatility*- no change in the agent environment.
- *Temporality*- episode

***JATLITE Communication Environment.***

The communication environment for JATLITE consists of the requirements as discussed below:

- *Communication*- provides two basic modes of communication: direct mode communication and broadcast mode.
- *Interaction*- based on one to one interaction through exchange information via agent communication languages.
- *Coordination* - all coordination is through the agent router, as a centralised coordination. Even though coordination is centralised, agent communication is based on one to one interaction. The system is planned to have a sequence of agent interactions based on the state and condition of each agent. During conversation agents exchange messages according to mutually agreed methods of conversation.
- *Interaction protocol*- documentation does not discusses on interaction protocol, but rather uses performatives defined in KQML agent communication languages. Users have to define their own interaction protocol.

The following describes the techniques used to provide efficient agent interactions.

*Interaction management-* JATLITE provides an interaction management to manage the different channels of communication used by the agent.

Language processing and policing – based on the KQML languages.

*Co-ordination strategic service-* JATLITE uses directory services coordinate strategy. The directory services is the provider on which all agents are subscribed.

*Policy enforcement services-*not provided

*Social differentiation-* does not have a social differentiation.

*Social order-* does not specify any specific social patterns.

***Organisation of JATLITE based agents system.***

Figure Appendix B2-1 shows the general architecture of the agent system solution. It is based on centralised agent management. Any two agents communicating must go through the agent directory. It is presented as an agent that mediates all communication among agents.

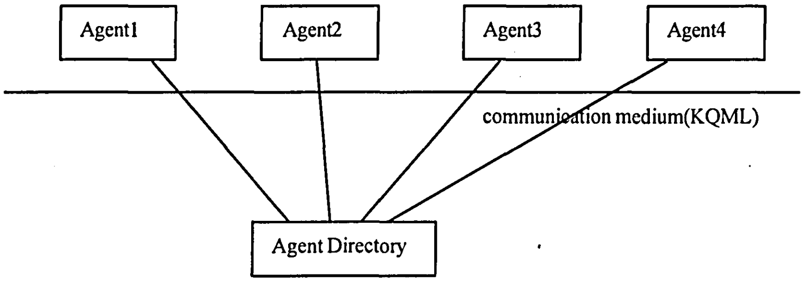


Figure C2.1: High level agent system architecture.

***The Three Layer of the Agent physical environment.***

- *Application layer* - Agent is described as Java applets. It describes the agent management by providing message routers for agent registration, connection, name, address, security (password) and services. It provides storage and queuing of messages for mobile and sporadic agents.
- *Communication and transportation-* basic communication among agents is through message routing rather than message polling. A router is used as directory services and provides a mechanism for communication with other agents using a centralised message repository. With such a method, communication messages are traced easily. Applet agents can run in any browser, communicating through HTTP server to send messages. The use of Agent Name Services(ANS) maintains all possible receiver addresses; thus it always keeps the latest address and the agents have no need to remember them. Communications are also supported using

SMTP and FTP protocol. The use of an internet service connector to extend these two protocol layers allows implementation of services other than SMTP and FTP especially for applets.

- *Physical layer-* is the hardware communication following the standard OSI TCP/IP communication protocol, supported by commonly used operating systems and extended to use other protocols.

The JATLITE environment is executed in any platform supporting Sun Java Development Kit JDK1.1 including Window95 and WindowNT, Solaris and MaxOS8. Applets agents can be run using a WWW browser such as Netscape or Internet Explorer or Sun's applet viewer.

***Architecture styles used in JATLITE***

JATLITE uses the Internet client/server architecture style, with a router as server and agents as clients. JATLITE is focused on communication. It is also focused on a layered system architecture for the web communication protocol. The lower levels define lower level interaction. The lowest level refers to hardware connections.

***JATLITE agent environment architecture.***

Figure Appendix B2-2 shows the JATLITE layer environment architecture. The user has access to all the layers. The user can select an appropriate layer to start building their system.

The user uses the TCP/IP communication but can only use KQML at Abstract layer and Base Layer.

*Abstract layer-* describes the components needed for agent communication, such as agent action, server thread, receiver thread, message buffer, address, security and address table.

*Base layer-* manage the communication using TCP/IP communication tools by specifying any higher level protocol connection by overriding the old address table, control connection table and security check.

*KQML layer-* enforces the communication protocol of connection and disconnection. It use the KQML syntax suggested in 1993.

*Router layer-* supports agents that are always on-line. The router layer is used for agent management of agent communication and as a directory for the agents. The client/server model is applied by defining the router as server and agents as router clients.

*Protocol layer-* supports other communication protocols such as SMTP and FTP using an Internet service connector.

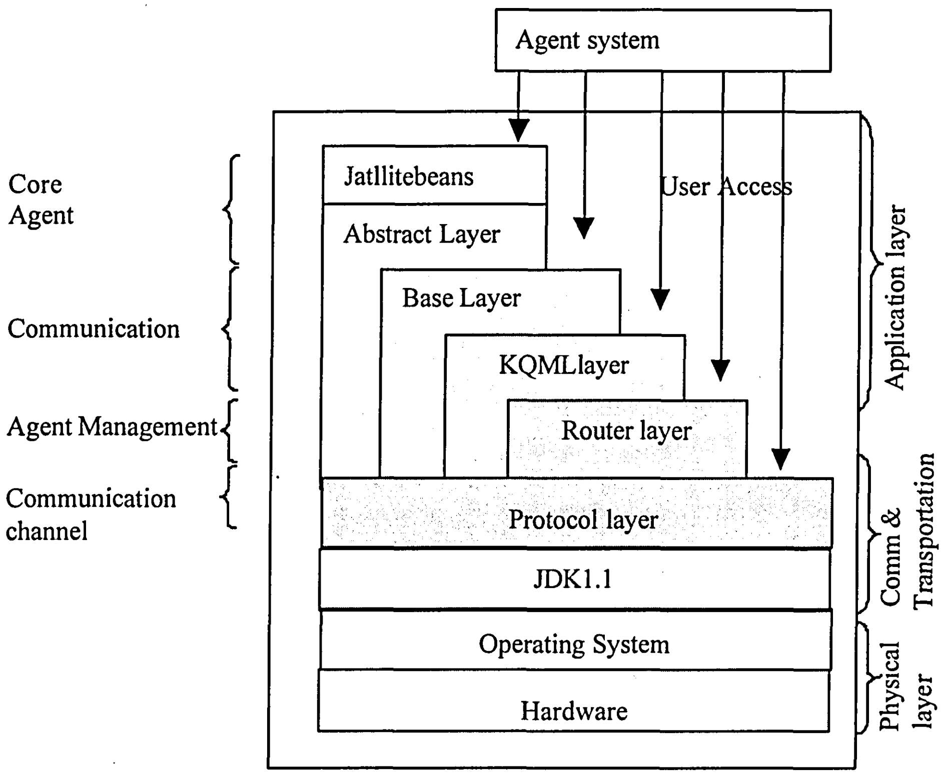


Figure C2.2: JATLITE Agent Environment Layer Architecture.

The client server architecture is mainly applied at the router layer. The server side uses RouterServer and the client side uses RouterClient. Both server and client components are shown in the Figure below. The RouterClient describes the internal agent architecture.

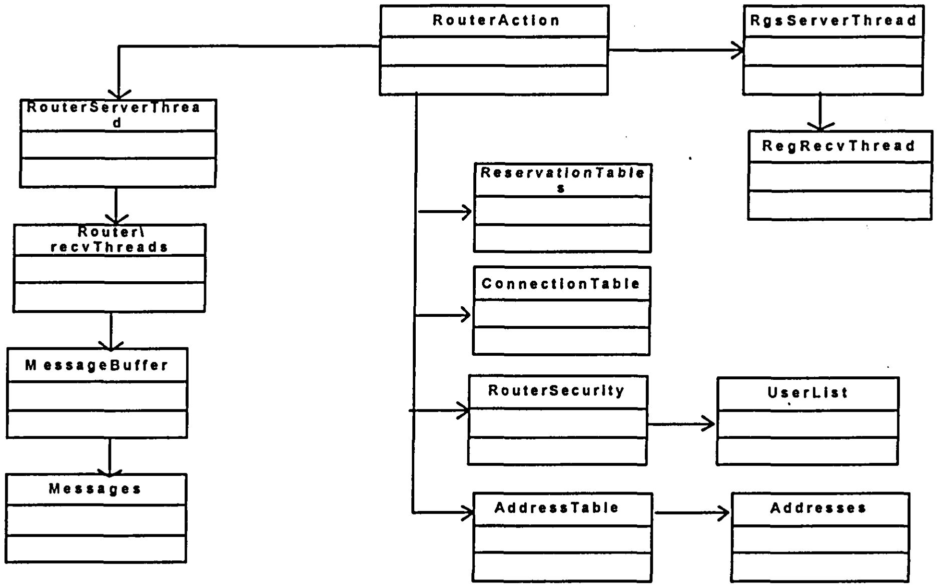


Figure C2.3: RouterServer Component Diagram.

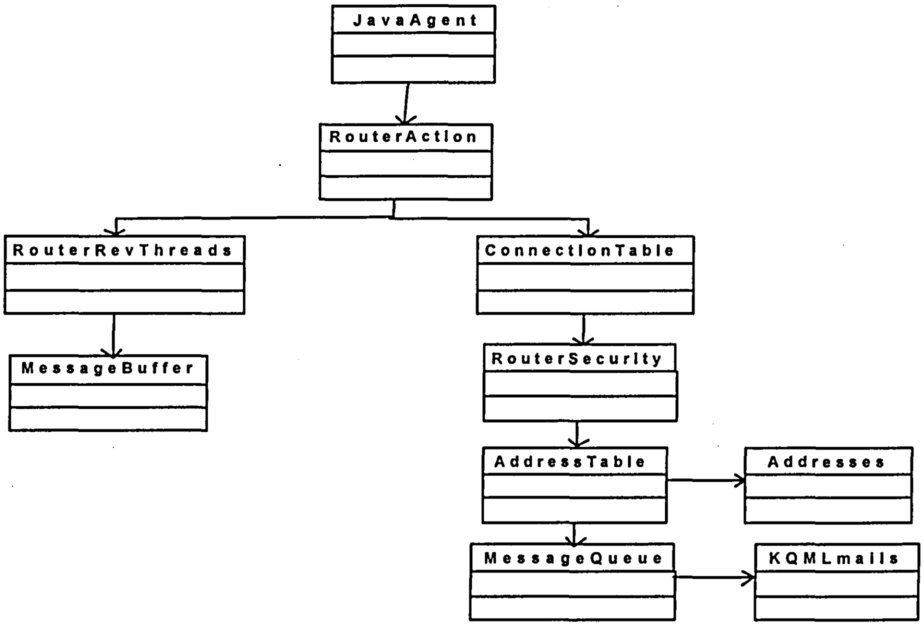


Figure C2.4: RouterCilentComponet Diagram

***Upgrading of JATLITE to JATLITEbean.***

JATLITE mainly provides an agent communication environment rather than providing autonomous behaviour. Therefore it was upgraded using the event-based architecture style. Each agent is provided with a thread that provides autonomous running in the environment. Any changes of agent state are captured through the eventlistener components. The remote agent is used to capture the event occuring for remote agent (different location). The following Figure shows the 'core agent' for JATLITEbean[Jat98] to describe the agent abstraction.

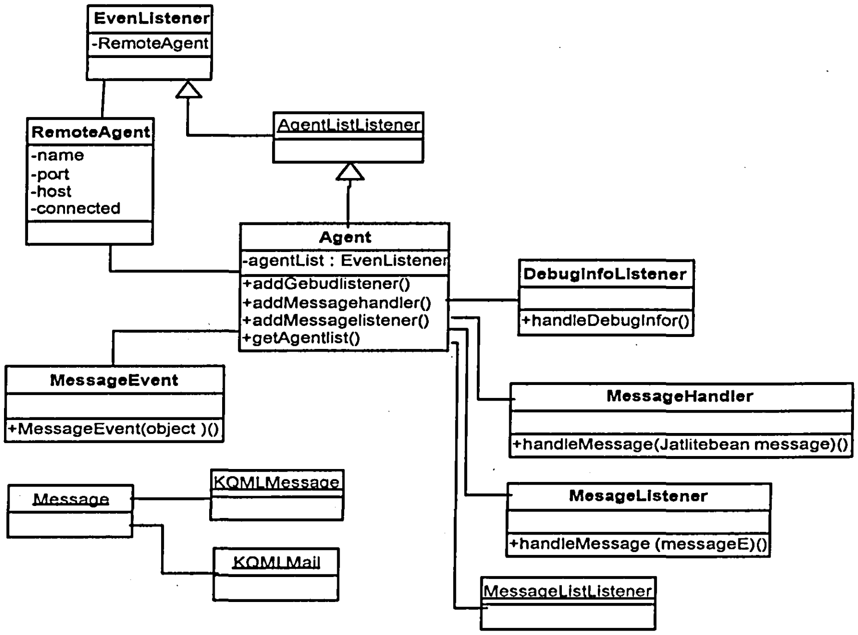


Figure C2.5: JATLITEbeans 'core agent'

## C-3: JADE.

Comparing Jade with JATLITE and JAFMAS we may note the following. Jade is different because it is developed based on FIPA architecture. Jade is an agent framework developed by Telecom Italia Lab in 1997 and the code implementation is available in [Jat97]. Jade is a multi-agent system based environment. FIPA is a non-profit association registered in Switzerland, in 1996. The aim of FIPA is to produce a software standards for heterogeneous and interacting agents and agent-based systems. FIPA specification comes with FIPA architecture. Jade simplifies the implementation of a multi-agent system environment through a middle-ware that claims to comply with the FIPA.

Jade only focuses on providing a minimum of agent abstraction. Jade is focused on providing interoperability of the agent environment for agents to openly socialise rather than the agent intelligence perspective. Agents can use any intelligence components such as Jess[Jess00]. Jess focuses on communicative, intelligent, rational and possible intention entities. This is similar to mobile agents, which present different functionality of the agent environment. Thus, the mobile agent environment can be reused and integrated with the Jade agent. The mobile agent environment is called Leap[Leap02]. However, in this section discussion we only focus on the Jade framework.

*Agent abstraction-* The agent is described as an active object with the following agent behaviours.

*Agent behaviour*- the abstract agent behaviour consists of autonomous behaviour and communal interaction (reactive behaviour). The *beliefs, desires* and *intentions* are an explicit set of beliefs about the agent and the rest of the world, a set of desires for goals to be achieved and intentions for carrying out actions to achieve those goals. Given that an act of communication is an action (known within the community as a communicative act for that very reason), the semantics of such an action are given in terms of what the agent believes to be true, what it believes about the state of knowledge of another agent and its desire to make another agent come to believe some fact of mutual importance. That is, the meaning of communicative acts between agents is explained in cognitive terms, such as what an agent believes to be true and what it desires to bring about. However, other agent behaviours can add to these minimum behaviours. For example, Jade supports a mobile agent environment on the top of these behaviours.

- *Communication* - Jade supports inter-agents communications and supports communication with different agent environments.
- *Distribution*- Jade presents conceptual and locational distribution. It provides a heterogeneous agent environment. This means agents in different environments are able to communicate with agents in the agent environment.
- *Openness*- Jade provides both openness criteria: Openness of the communication channel used for communication, and open services in, whereby an agent that exists in the Jade environment is open to access by other agents from anywhere.

*Agent Physical environment*

- *Accessibility*- all agents have to subscribe into directory services. Each agent should have permission to access the server. Each agent is provided with a name, address and status. Agents under the same directory services or agent management have the ability to interact with each other, but agents registered in agent directory A cannot access agents in agent directory B. Agents which other agents can access are registered in green pages.
- *Deterministic*- Determinism is based on the agent's status as connected, so that the other agents are able to interact with it.
- *Controllability*-not mentioned
- *Volatility*- not mentioned
- *Temporality*- the agent is continuously actives
- *Locality* - distributed location in different agent environment.

***The Three Layers of the physical environment.***

- *Application layer* - The application layer consists of agent management of agent registration, connection, name, status and address. The agent provides a query function to find agents, advertise the agent services, defines the ontology and interaction protocols and provides a *directoryfacilitator* that provides agent openness.
- *Communication and transportation*- Jade consists of two layers of agent communication. The upper level is the application layer, which defines the agent communication language, the interaction protocol and ontology, while the lower layer uses an agent communication channel that allows agents to communicate physically. The use of message transportation services allows any agent to support openness to use any agent communication channel. The message delivery service is able to determine the correct transport mechanism (TCP/IP, SMTP, HTTP, etc).
- *Physical layer*- is the hardware communication that supports the ISO/IEC 2022 standard.

The Jade environment is executed in any platform supporting Sun Java Development Kit JDK1.2 above, including Window95 and WindowNT.

***Jade Communication Environment.***

Communication between Jade agents is based on one to one communication. However, it supports broadcast using domain services. Domain services are defined as a group of agents with similar services. An agent may be assigned more then one domain service.

The Jade agents' interaction, cooperation or negotiation is also based on one to one interactions. The pattern of the series of interactions between two aspects in order to achieve a goal is called the interaction protocol.

Jade environment supports the use of KQML or FIPA agent communication languages. The information contained in the content languages defines as text and identifies the meaning according to the type of ontology used.

***Organisation of Agents System.***

The figure below shows the general architecture of agent system solution. It is based on decentralised agent management, however it can also use centralised agent management. The architecture of Jade agent system is generalised as shown below. Any two agents can directly communicate with each other. The agent management is defined at lower level to identify the cycle of agent, creating and removing agent, and status of agent. This agent management is identified as white pages of *directory services*. It provides a list of agents are registered, the status and address. Despite this Jade environment provides optional yellow pages so called *directory facilitator*. Any agents that want the service openly access to any agents can be registered to the green pages. The use of green pages provides open characteristics.

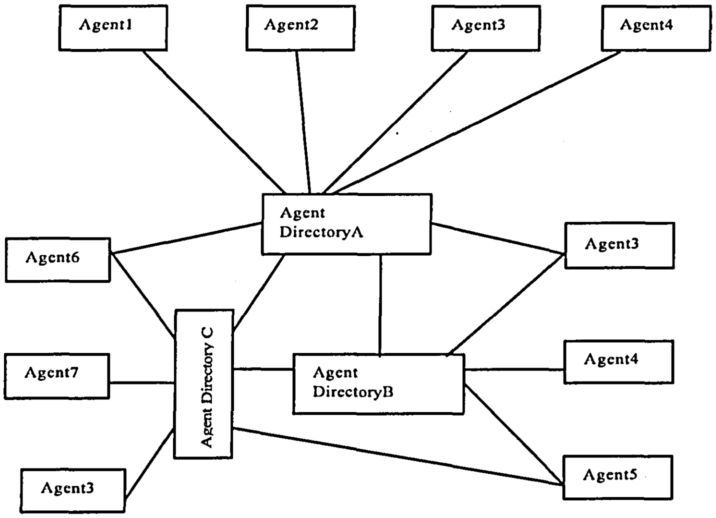


Figure C3.1: Jade Coordination patterns

### *Architecture styles used in Jade*

Jade mainly developed based of Jade agent architecture on top of CORBA architecture. CORBA architecture has presented a highly heterogeneous distributed system and application. It is as a middle-ware of applications system and standard communication. Thus, the architecture design is based on CORBA as well. The next section discusses the Jade agent environment architecture.

### *Jade agent environment architecture.*

Figure Appendix C3-2 below shows that Jade environment is built according to CORBA architecture. The application objects are defined as agents. The common facilities are identified: agent management system and directory facilitator. The Object request broker becomes the message transport system which is responsible for providing transportation services between the two agent communication channels in different physical environment (agent platform).

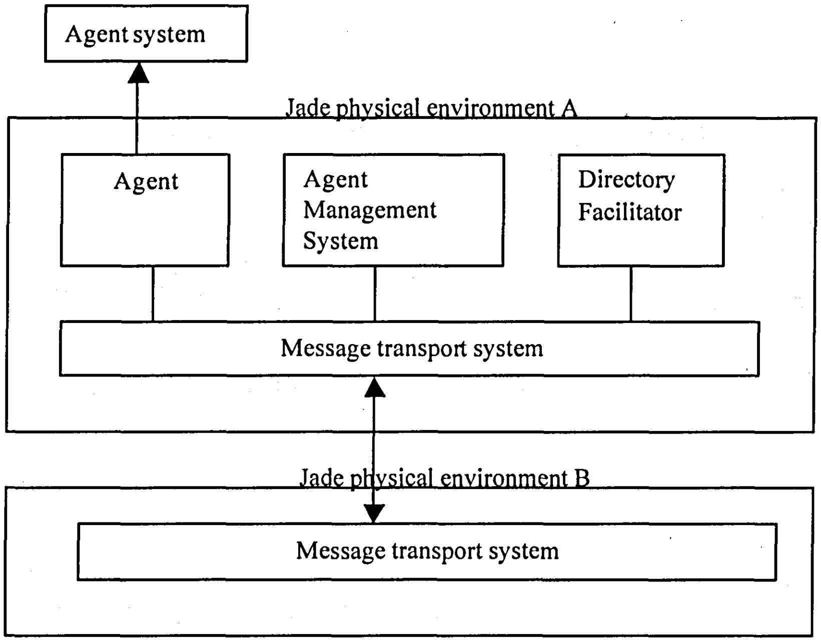


Figure C3.2: Jade Agent Environment architecture.

Agent Components- The agent component is identified as a core agent for the system. It presents the agent abstraction and the agent architecture of the agent environment. The agent architecture describes the behaviours of it self, capability to access external software, human users and communications facilities. An agent may have certain resource brokering capabilities for accessing software (see FIPA00079 in www.fipa.org).

The agent may support several notions of identity. An Agent Identifier (AID) labels an agent so that it may be distinguished unambiguously within the Agent Universe. An agent may be registered at a number of transport addresses at which it can be contacted and it may have certain resource brokering capabilities for accessing software.

Directory facilitator (DF) - provides yellow pages services to other agents. Agents may register their services with the DF or query the DF to find out what services are offered by other agents. Multiple DFs may exist within an agent physical environment and may be federated.

Agent Management System (AMS) - exerts supervisory control over access to and use of the AP. Only one AMS will exist in a single agent physical environment. The AMS maintains a directory of AIDs which contain transport addresses (amongst other things) for agents registered with the AP. The AMS offers white pages services to other agents. Each agent must register with an AMS in order to get a valid AID.

Message Transport Service (MTS) is the default communication method between agents on different APs (see [FIPA00067] in www.fipa.org).

# Appendix D: Software Abstraction in Agent Environment.

This appendix examines the four types of software abstraction i.e framework, software architecture, architecture pattern and design patterns in the four existing agent environments as discusses in Appendix C. This appendix presents the used of software abstraction in the existing agent frameworks.

## Appendix D-1: The use of Software Architecture and Architecture Patterns Existing Agent Environment.

The SEAF and RE processes found the use of architecture patterns to express fundamental organisation schema for structuring software systems. It provides a set of predefined subsystems, specifies their responsibilities, include rules and guidelines for organising the relationships between them.

Through the investigation of the software architecture development approaches, we found that all of them presented a particular architecture to express fundamental organisation schema for structuring software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organising the relationships between them.

We use the Busshmann [Buss+00] architecture pattern to show the process. Busshmann has identified eight architecture patterns: layers, pipes and filters, Blackboard, Broker, Model-View-Controller, Presentation-Abstraction-Controller, Microkernel and Reflection, that can be classified according to the following context:

- Define architecture from mud to structure - The Layer, Pipes and Filters, and Blackboard architecture are the high level architectures chosen for system architecture design.
- Distribution systems- The Microkernel, Broker, and Pipes and Filters are the architecture patterns presenting the criteria for distributed systems.
- Interactive systems- The two patterns of Model-View-Controller and Presentation-Abstraction-Control are used to present the criteria of interactive systems.
- Adaptable Systems- if the system meets the criteria of adaptable system, the Reflection and Microkernel patterns are strongly used to support extension of applications and adaptation to new technology and changing functional requirements.

Architecture patterns need to be understood based on the context of the system. For example, if system presents a distribution system criterion, by understanding the distributed architecture pattern, the designer knows what components are needed and how they interact with each other to present a distributed system. The architecture pattern provides the high-level architecture for a distributed system. It shows the components used and the inter-relationships. It also provides several scenarios for applying the patterns and describes the relationship with other patterns. The patterns also provide examples of different alternative dstributed architecture based on CORBA, International Business Machines(IBM) (Component Object Model/Distributed Component Object Model)COM/DCOM, ntier client/server architecture and WWW, ATM-P[Busch+96]. With such alternatives, the user can choose and apply the architecture. The architecture also may present a combination with other architectures. By learning from the example of the architecture, the developer also can learn how the architecture is designed and how it can be integrated with other types of architecture.

There are several patterns that can be used to design the high-level abstraction architecture of an agent environment which have the features distributed design. These patterns involve infrastructure, which we define as the physical environment. The patterns can be incorporated into the architectural designs such as using of-the-self reusable components. [Grand01] has identified several distributed architecture patterns that can be used for agent environment design: Share Object, Object Request Broker, Object Replication, Redundant Independent Objects, Prompt Repair, Mobile Agent, Demilitarise Zone and Process Repair. The selection of patterns depends whether the design centralised or decentralised, conceptual or locational distribution, heterogeneity of the system and also include aspects of security as well. For example JATLITEbeans is based on centralised design, which influences the decision on use client/server architecture that assigns agent management to the server, while JADE applies a decentralised design, which creates additional components of agent management to handle several client/server.

Based on the issues that influence agent environment designing, we found all the scrutinised agent frameworks use layers to structure the abstraction of architecture environment, as shown in Appendix C3-1(JADE), Figure Appendix C2-1 (JATLITE) and Figure Appendix C1-1(JAFMAS).

All agent frameworks present distributed system and all the agent frameworks applied the Broker architecture pattern, which it provides a complete infrastructure for distributed system, which is a standardised distribution for OMG [OMG92]. Figure D-1.1 shows the use of architecture patterns in each agent framework.

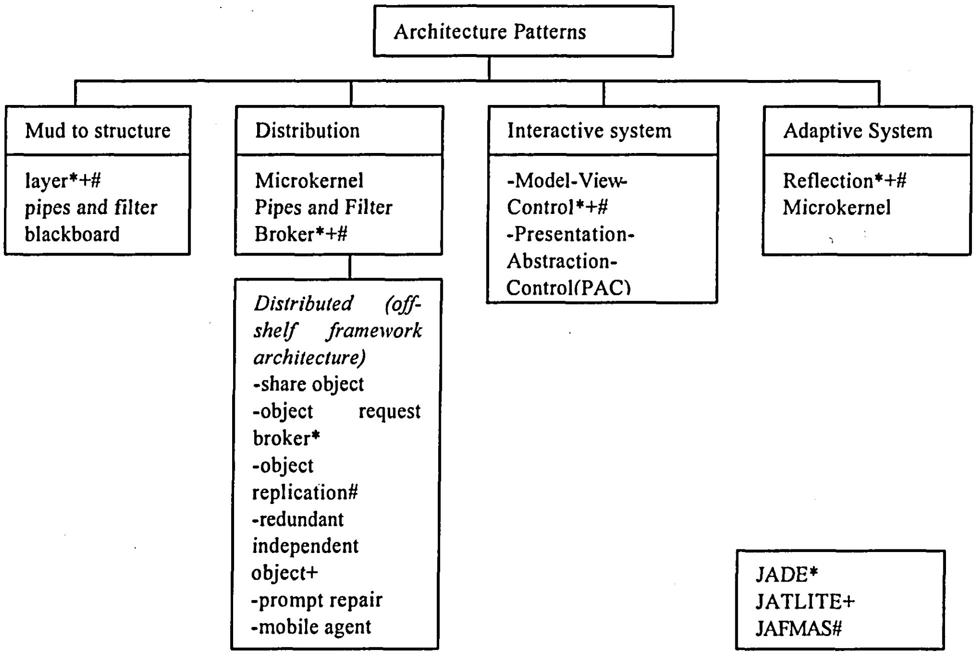


Figure D1.1: The used of Architecture Patterns in Agent Frameworks.

The Broker architectural pattern comprises six participation components: *clients, server, brokers, bridges, client-side-proxies* and *server-side proxies*. However, the choice of techniques for the solution, influences the identification of components to present the agent environment architecture. Figure D1.2 shows the components and their relationships in a distributed architecture pattern. Based on these basic components we then show how the components are abstracted in relation to the abstraction that we captured in the scrutiny process. According to the architecture pattern the diagram of the broker system is applied in three scenarios:

**Scenario 1** illustrates the behaviour when a server registers itself with the local broker components. It consists of server and broker components only.

**Scenario 2** illustrates the behaviour when a client sends a request to a local server. This scenario describes synchronous invocation, in which the client blocks until it gets a response from a server. The broker may also support asynchronous invocations, allowing clients to execute further tasks without having to wait for a response.

**Scenario 3** illustrates the interaction of different brokers via bridge components.

Through scrutiny of the three agent environments, we identified three components used to develop the agent environment as *Agent, Inter-agent communication* and *Agent management.*

These components are developed based on the Broker Architecture Patterns as shown in Figure D1.2. The agent is a client representative, which is provided with certain behaviour. The agent communication component presents a high level communication used for agent communication (ACL, content message and ontology). The communication component is an additional component that defines the agent communication components. At lower levels of agent communication, components are associated as with the broker component's responsibility, which is to establish the communication. The agent management here is on the sever side, where the agents are registered, make query on services, etc. The bridge component is needed when the agent environment is heterogeneous, meaning that agents use different communication channel to communicate with each other.

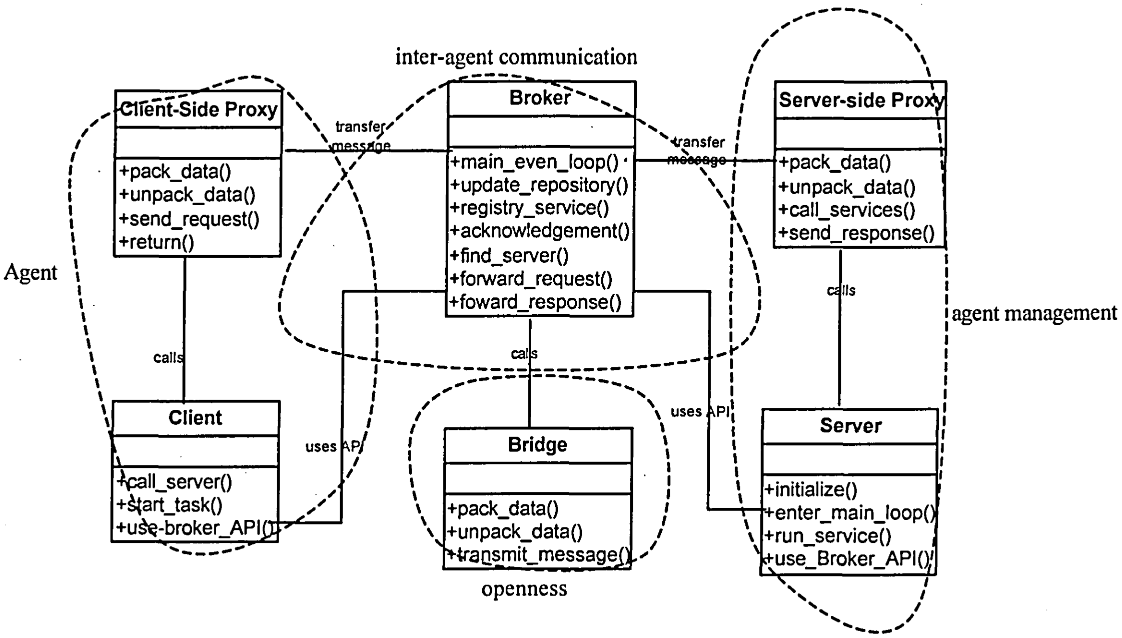


Figure D1.2: Broker Architecture Patterns.

In spite of these three scenarios, several kinds of broker system can be applied, that present different agent environment architecture.

*First*, Direct Communication Broker System. This method is chosen when the agent can only make requests through a local broker for reasons of efficiency. So, the broker tells what communication channel the server provides. The agent can establish a direct link to the directory facilitator (server). In this condition the proxies take over the broker's responsibility for handling all communication activities. So the client (agent) can communicate directly because it knows the client address directly. This method also allows use of the bridges when agents are located in different locations.

*Second*, is the Trader system. In the Trader System, agents make queries to only one uniquely-identified server. The broker knows which server provides services. The agent client-side proxies use service identifiers instead of server identifiers to access server functionality. But the same request can be forwarded to more then one server by implementing the same service.

*Third,* the Adapter Broker System uses an additional layer to hide the interface of the broker component to the servers to enhance flexibility. The adapter layer is a part of the broker and is responsible for registering servers and interacting with servers.

*Fourth*, the Call-back Broker System uses a reactive model rather than implementing an active communication model in which clients produce requests and servers consume them. The reactive model is event-driven and makes no distinction between clients and servers. Whenever an event arrives, the broker invokes the call-back method of the component that is registered to react to the event. In certain conditions, the execution of the method may generate new events that cause the broker to trigger new call-back method invocation.

There are several ways to present the agent environment architecture by combining the above variants. For example, a combination of the Direct Communication Broker system and Trader system will result in an agent environment that allows an agent request communication to cause the broker to select one server from among those that provides the requested service. The broker then establishes a direct link between the client and selected server.

The JADE environment applies a variant of Direct Communication Broker system. It also provides a heterogeneous agent environment and applied the CORBA. The communication component uses an interface definition language that is available to support the interoperability of agents and server. JADE also applies the reactive model being event-driven. So, when any event comes from agents, the broker invokes the call-back method from where it is registered and then reacts to the event.

The JAFMAS and JATLITE also apply this reactive model, but the JATLITE applies the Direct Communication Broker system combined with the Trader system. JATLITE creates a router layer to allow direct communication.

Based on the discussion above we can see how architecture patterns are used in designing agent environments. The use of the architecture patterns is able to abstract away the complexity of development at high level design, while design patterns, as will be discussed in the next section are able to abstract away the complexity at lower design stages.

The scrutiny of existing agent frameworks revealed the three layers of the physical environment. These three layers are used to present the basic layer architecture for agent environment development. Table D1.1 shows how those three layers are presented, while Figure D1.3 shows the detail description of layer pattern for development agent environment[1].

| | JADE | JAFMAS | JATLITE |
|---|---|---|---|
| **Application layer** | Agent<br>Agent Mgmt System<br>Directory Facilitator | Social Model<br>Linguistic Layer<br>Local Model | JATLITE Agent<br>Abstract layer<br>Base Layer<br>KQMLlayer<br>Routerlayer |
| **Communication Layer** | Message transport system | Comm. Protocol layer | Protocol layer |
| **Physical layer** | http, smtp, MS series, tcp/ip | UDP socket, TCP/IP | ftp, http, smtp. |

Table D1.1: The three layers of agent environment.

Layer 1 encapsulates the agent's ability to interact with external objects, application and databases through the mediator, sensor and adaptor patterns respectively. This type of communication is known as non-agent communication.

Layer 2 describes the reasoning layer that encapsulates the interpreter and strategy patterns. An automated reasoning for each agent is created by instantiating an automated reasoning action in a thread using active object patterns. Layer 3 is the collaborative layer model to show the agent's ability to communicate and coordinate with other agents through standard message parsing or ACL message and coordination protocols use the mediator and proxy patterns.

Layer 4 shows the ability to physically transmit messages across threads and processes to other agents, as it incorporates mediator, active object and broker patterns.

The analysis result shows that agent environment development uses distributed architecture patterns. The choice of different kinds of distribution influences the components identified. For example, JATLITE is based on internet client- server architecture while JADE is based on CORBA architecture. But design patterns are used in the solution of smaller contexts of architecture design as presented as adapter pattern, sensor, etc.

[1] modification of the layer pattern proposed by [Kend+95]

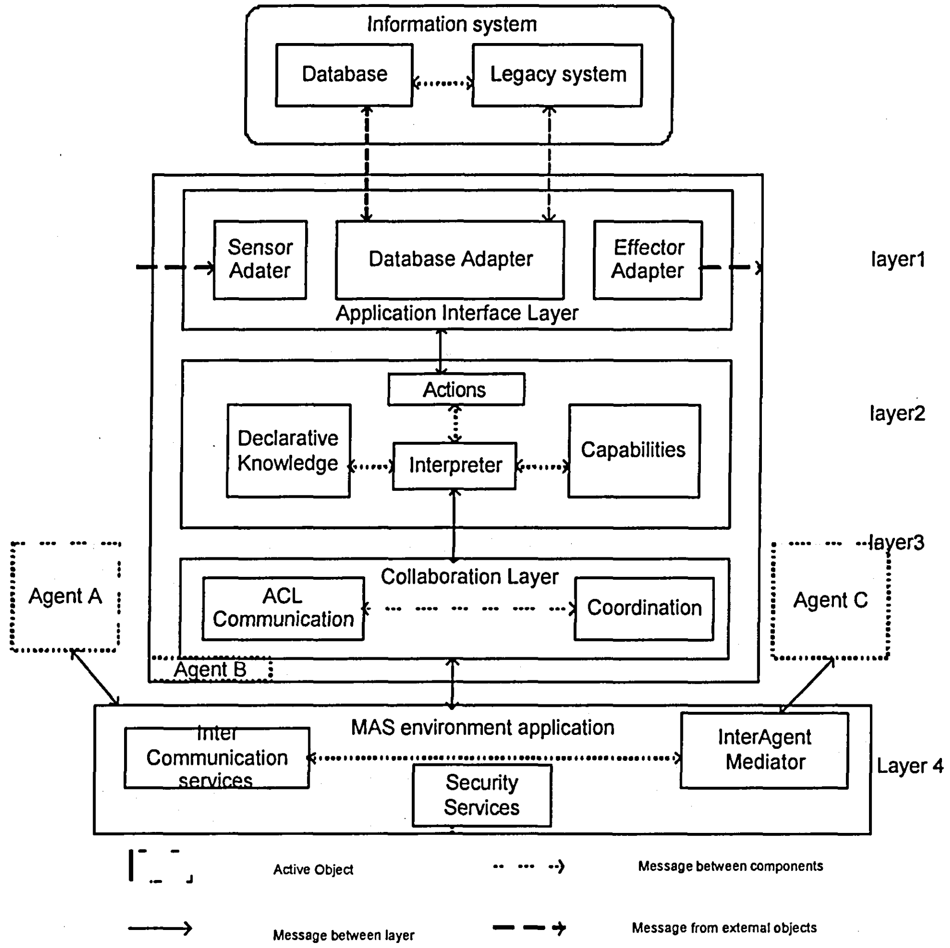


Figure D1.3: Agent Environment Architecture Layer Patterns

## Appendix D-2: The used of Design Patterns in Existing Agent Frameworks

In the previous section, we discussed the use of architecture patterns to develop agent environment architecture. We also discussed that the architecture consists of components and their inter-relationships. At this stage, design activity is focused on the design of each component. However, in a small-scale agent environment, such as JAFMAS, the broker component is not applied, because it is rather slow. Some agent environments focus on a good performance of agent interactions, so they depend directly on a concrete mechanism for inter-process communication. Broker patterns introduce an indirect layer that focuses on providing a portability, flexibility and changeability. Broker patterns are usually applied for large-scale of agent environments. In a small scale agent environment, inter-process communications can be applied using design patterns, which do not have a broker component.

There are several design patterns that can be applied for the inter-process communication between two agents.

First, the *Forward- Receiver* pattern [Busch+96] encapsulates inter-process communication between two agents (components). An agent as a client-side forwarder receives a request and addressee from the agent using the IPC (inter-process communication) facility. The receiver on the server side unpacks and delivers the message to the server.

Second, the *Proxy* pattern [Gam+95] are usually applied for remote cases. The remote proxy is often used in junction with a forwarder. The proxy encapsulates the interface and remote address of the server. The forwarder takes the message and transforms it into IPC-level code.

Third, the *Mediator* patterns [Gam+95] is a centralised agent management. It is like a broker pattern but it uses a hub of communication. In applying the mediator pattern it is necessary to investigate the nature of the request in order to respond appropriately.

The first design pattern focuses on peer-to-peer interaction, whereas the second design pattern focuses on the intermediation between client and server. The third helps to synchronise of cooperating agents.

JAFMAS is a small agent environment. It does not apply the broker pattern, but uses proxy patterns in designing the agent environment. It uses msgRouter as forwarder. It takes the message to transform into inter-process communication level through the UPD socket.

On the other hand, the design pattern is used to design each component of the agent environment architecture. The broker components are designed dependent on the requirements of the agent environment, which may consist of additional services such as name services or marshalling support, which are integrated into the broker. This is where the openness agent criterion is applied. JADE, as an example, provides two types of openness in the agent environment. First, open agents services apply green pages and second, agents are allowed to use several types of communication channels. This means the agents are free to use HTTP or SMTP for communication. The client components are identified as agents in a multi-agent system. The client components design is based on agent architecture.

The client-side proxy is a component presenting a layer between clients and server and vice versa. The components design is based on inter-process communication mechanism used for message transfer between clients and brokers, the creation and deleting the memory blocks and the marshalling of parameters and results. In many cases, the components are translated as an object model that specifies the broker pattern to object model of the programming languages used to implement the client.

The Server-side proxy component is generally analogous to client-side proxies. The difference is that it is responsible for receiving requests, unpacking incoming messages, unmarshalling the parameters and calling the appropriate services. For example in JADE, the HTTP server or SMTP server built can communicate.

The last element is the Bridge component, which is able to present a heterogeneous agent environment. Bridge is esponsible for encapsulating network-specific functionality and as a mediator between the local broker and the bridge of a remote broker. The Bridge component is not applied in the JATLITEbeans but is applied in JADE, which enables the openness of a heterogeneous agent environment to be provided.

There are other more detailed design pattern aspects which are related to the provisions of concurrency, distributed computing patterns, database patterns for internal agent interaction with external objects or legacy system. These are detailed in [Gam+95][Grand01][Gran97] [Gran99][Mow+97]. Here, we summarise their use in the agent environment development.

After identifying the components, the next step is to develop each component. We take the client component as an example. The client component is identified as a 'core agent'. Figure Appendix B2.4 shows the core Jade agent of an agent frameworks. The core agent is described as the clients' components design. The figure shows the use of composite patterns [Gam+95], adapter/observer patterns [Gam+95] and reactive patterns [Aare+99] similar to core agent for CIAgent shows in Figure B2.1 (see Appendix B).

## Appendix D-3: Identifying Framework or Component in Existing Agent Environment

As a result of scrutiny of the existing agent frameworks, we have found that the development of an agent environment represents a similar pattern to a framework application process, which focuses on the reuse aspect of development of a family type of software applications. However, in the development agent system, the development of an agent environment is becomes necessary.

This is because agent system consists of several agents, which provide at least a similarity basic capabilities, properties or way to interact among agents that can be reused among the agents.

The derivation process shows that the core element of development of an agent environment is identifying components and sub-components that are inter-related each other for producing the architectural design. The nature of this development process is similar to the software architecture process approach proposed by [Shaw+96]. It is concerned with producing a high quality architecture. The development process of Framework application plays a role in development of an agent environment is because it provides a method to produce reusable components based on the architectural design.

The main concept, which the Frameworks approach offers reusable component is the identification of the concrete and abstract classes, and how they are integrated according to the architectural design. The scrutiny of agent frameworks covered in this thesis was only applied to a similar type of agent architecture in a multi-agent environment. Therefore we can see that the existing agent frameworks only presented one type of 'core agent' abstract component. The RE of the JADE environment has captured the organisation of the reused components as shown in Figure D-3.1

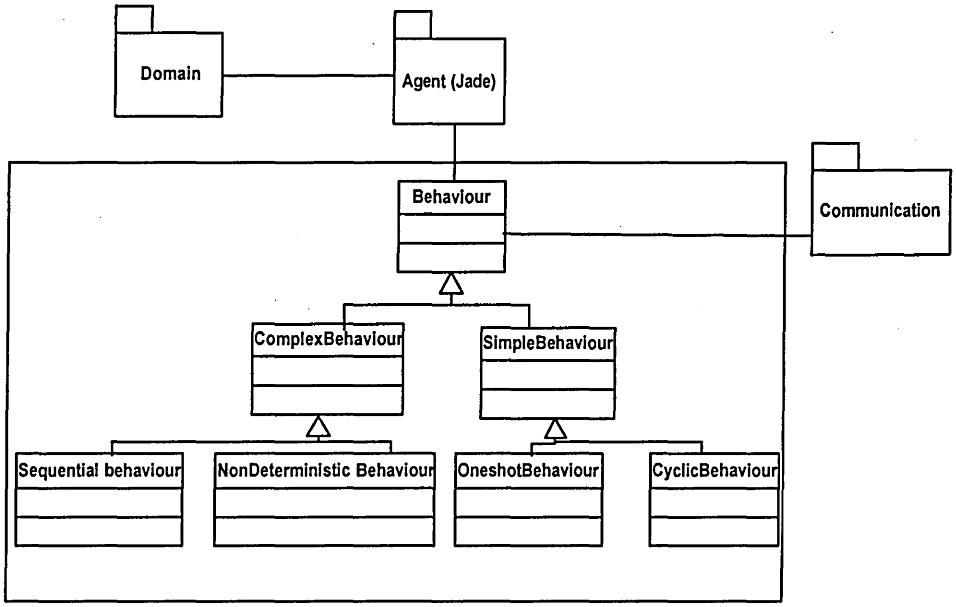


Figure D3.1: The Organisation Class of Agent Behaviours in JADE

The figure shows several types of abstract components of Agent and behaviours and their relationships to present agents and their roles as discussed in agent overview in Chapter three. The communication component is defined as a concrete class, while the agent component is defined as abstract class. The communication component consists of several sub-components. ACL consists of message content, ACL performatives and ontology as we showed in the diagram. There are several ways that agents interact with other agents, which can be reused, such as request, inform and bidding, also described as patterns of agent interaction. Agent interaction patterns can be defined as reused communication components.

We can see that these concepts are applied within the development of object-oriented application frameworks and object-oriented patterns used as guidance to structure the organisation of the components.

# Appendix E: Agent Modelling and Notation.

## E-1: Notation Definition.

This section presents the symbol of notation used to present the analysis and design concept.

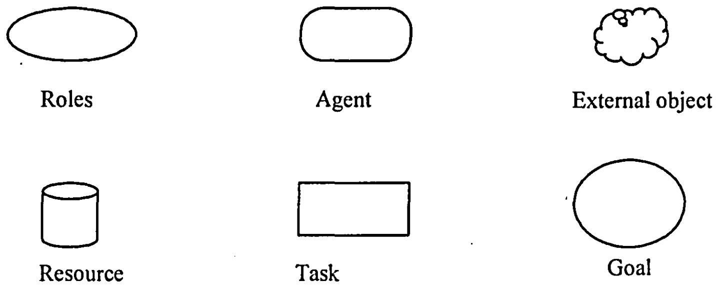


Figure E-1: Concept Symbol

The Figure bellow shows the Relation symbol use in the methodology.

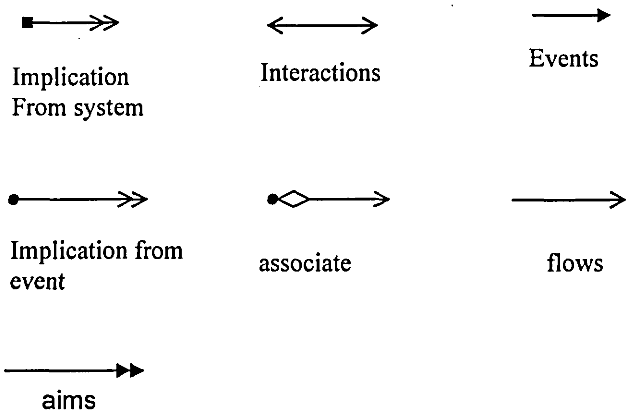


Figure E-2: Relation Symbol

## E-2: Agent Model.

The notation used in modelling and the figure below shows the notation use to model agent.

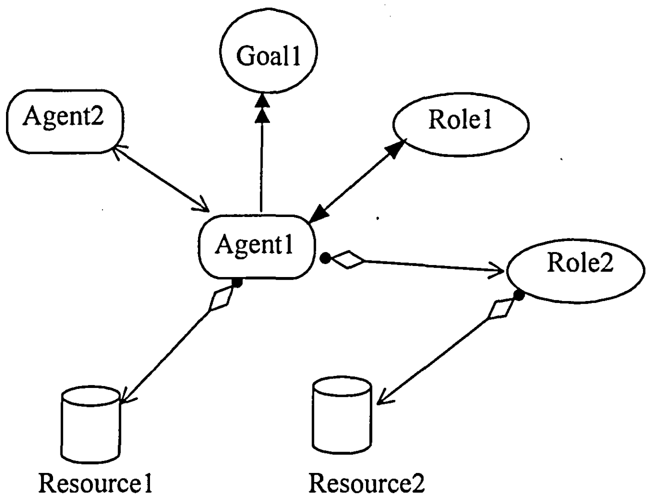


Figure E2-1: Agent Model

## E-3: Goal Modelling.

The notation used to model the system goal. The Goal 2 is goal capture from the system, while Goal3 is depends on event.

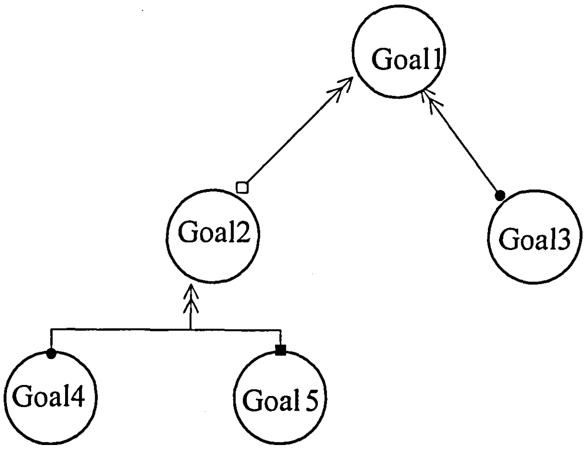


Figure E3-1: Goal Modelling

## E-4: Scenario Model.

| **Scenario Number** | | |
|---|---|---|
| Event | **Event Name** | |
| Source | **Source of Event** | |
| Activity | | |
| 1. | Activity 1 | Role 1 |
| 2. | Activity 2 | Services |
| 3. | Activity 3 | Role 2 |
| 4. | Activity 4 | Role 3 |
| 5. | Activity 5 | Task2 |
| 6. | Activity 6 | Role 4 |
| 7. | Activity 7 | Role 5 |
| 8. | Activity 8 | Task 3 |
| 9. | Activity 9 | Task4 |
| 10. | Activity 10 | Role 6 |
| 11. | Activity 11 | Description |
| 12. | Activity 12 | Role 7 |
| 13. | Activity 13 | Action |
| 14. | Activity 14 | Internal Action |
| 15. | Activity 15 | Role 8 |

Figure E-4: Scenario Model

## E-5: Use Case Modelling

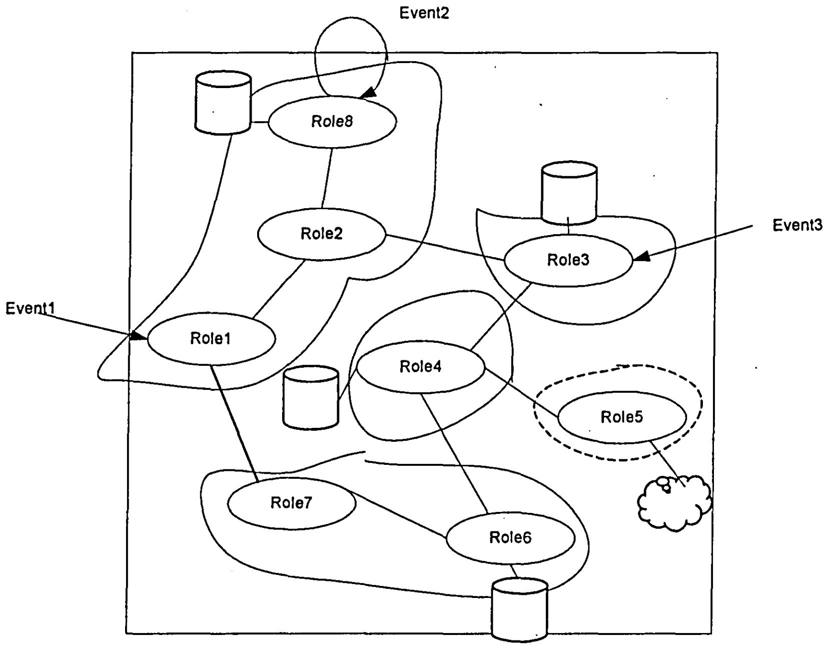
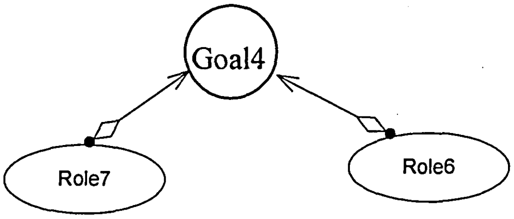
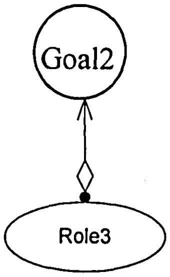


Figure E-5.1: The role based Use Case



Goal 4 achieve based on the Role7 and Role 6



The Role3 performs to achieve the Goal 2 but the Role3 is dependence with Goal 3 and Goal 4.

Figure E-5.2: The relationship Goal and Role

## E-6: Agent system Architecture Modelling

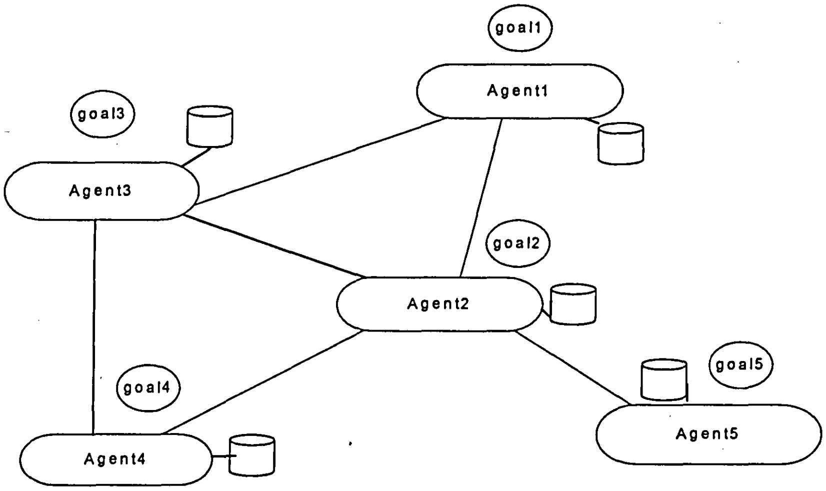


Figure E-5.3 : The Agent System Level Architecture Design.

The other modelling as state in Chapter five are use extension of the object oriented modelling as following models. However, the differences of the concepts are discusses in Chapter three and five:

Agent Plan Sequence Diagram- Sequence diagram
Agent Plan Activity Diagram – Activity diagram
Agent Level Plan – State Diagram
Agent Deployment- class diagram, collaboration diagram