I am the author of the thesis entitled *An Agent-Based Hybrid Framework for Decision Making on Complex Problems* submitted for the degree of DOCTOR OF PHILOSOPHY.

This thesis may be made available for consultation, loan and limited copying in accordance with the Copyright Act 1968.

Full Name: Zili Zhang

Signed .................................................................

Date ..............................................................

29 January, 2002

Signature Redacted by Library
# Consultation of Thesis

Please sign this form to indicate that you have used this thesis in accordance with the *Access to Thesis* form signed by the author of this thesis.

<table>
<thead>
<tr>
<th>NAME (please print)</th>
<th>SIGNATURE</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gillian Corkeron</td>
<td></td>
<td>28/10/03</td>
</tr>
<tr>
<td>Doug Congdon</td>
<td>Signature Redacted by Library</td>
<td>29/10/04</td>
</tr>
</tbody>
</table>
AN AGENT-BASED HYBRID FRAMEWORK FOR
DEcision making on complex PROBLEMS

By

Zili Zhang
BSc (SiChuan University, China)
MEng (Harbin Institute of Technology, China)

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy
of Deakin University
September 2001
DEAKIN UNIVERSITY
CANDIDATE DECLARATION

I certify that the thesis entitled:

An Agent-Based Hybrid Framework for Decision Making on Complex Problems

submitted for the degree of:

DOCTOR OF PHILOSOPHY

is the result of my own research, except where otherwise acknowledged, and that this thesis in whole or in part has not been accepted for an award, including a higher degree, to any other university or institution.

Full Name: Zili Zhang

Signature Redacted by Library

Date: 25/01/2002
To My Wife Qiong and Our Lovely Daughter Liqi
Abstract

Electronic commerce and the Internet have created demand for automated systems that can make complex decisions utilizing information from multiple sources. Because the information is uncertain, dynamic, distributed, and heterogeneous in nature, these systems require a great diversity of intelligent techniques including expert systems, fuzzy logic, neural networks, and genetic algorithms. However, in complex decision making, many different components or sub-tasks are involved, each of which requires different types of processing. Thus multiple such techniques are required resulting in systems called hybrid intelligent systems. That is, hybrid solutions are crucial for complex problem solving and decision making. There is a growing demand for these systems in many areas including financial investment planning, engineering design, medical diagnosis, and cognitive simulation. However, the design and development of these systems is difficult because they have a large number of parts or components that have many interactions. From a multi-agent perspective, agents in multi-agent systems (MAS) are autonomous and can engage in flexible, high-level interactions. MASs are good at complex, dynamic interactions. Thus a multi-agent perspective is suitable for modeling, design, and construction of hybrid intelligent systems. The aim of this thesis is to develop an agent-based framework for constructing hybrid intelligent systems which are mainly used for complex problem solving and decision making.

Existing software development techniques (typically, object-oriented) are inadequate for modeling agent-based hybrid intelligent systems. There is a fundamental mismatch between the concepts used by object-oriented developers and the agent-oriented view. Although there are some agent-oriented methodologies such as the Gaia methodology, there is still no specifically tailored methodology available for
analyzing and designing agent-based hybrid intelligent systems. To this end, a methodology is proposed, which is specifically tailored to the analysis and design of agent-based hybrid intelligent systems. The methodology consists of six models – role model, interaction model, agent model, skill model, knowledge model, and organizational model. This methodology differs from other agent-oriented methodologies in its skill and knowledge models. As good decisions and problem solutions are mainly based on adequate information, rich knowledge, and appropriate skills to use knowledge and information, these two models are of paramount importance in modeling complex problem solving and decision making.

Follow the methodology, an agent-based framework for hybrid intelligent system construction used in complex problem solving and decision making was developed. The framework has several crucial characteristics that differentiate this research from others. Four important issues relating to the framework are also investigated. These cover the building of an ontology for financial investment, matchmaking in middle agents, reasoning in problem solving and decision making, and decision aggregation in MASs. The thesis demonstrates how to build a domain-specific ontology and how to access it in a MAS by building a financial ontology. It is argued that the practical performance of service provider agents has a significant impact on the matchmaking outcomes of middle agents. It is proposed to consider service provider agents’ track records in matchmaking. A way to provide initial values for the track records of service provider agents is also suggested. The concept of "reasoning with multimedia information" is introduced, and reasoning with still image information using symbolic projection theory is proposed. How to choose suitable aggregation operations is demonstrated through financial investment application and three approaches are proposed – the stationary agent approach, the token-passing approach, and the mobile agent approach to implementing decision aggregation in MASs.

Based on the framework, a prototype was built and applied to financial investment planning. This prototype consists of one serving agent, one interface agent, one decision aggregation agent, one planning agent, four decision making agents, and five service provider agents.

Experiments were conducted on the prototype. The experimental results show
the framework is flexible, robust, and fully workable. All agents derived from the methodology exhibit their behaviors correctly as specified.
Acknowledgments

First, thanks to my supervisor and friend, Associate Professor, Dr. Chengqi Zhang for his guidance and stimulating discussions during the course of this work. He always gives me timely and constructive feedback for my research ideas, papers and thesis drafts. His suggestions are helpful, and his comments and questions are very insightful. I am lucky and happy to have been able to work with him during my PhD program. Thanks, Chengqi!

I am grateful to the School of Computing and Mathematics at Deakin University, which has provided an excellent environment for learning and experiencing, and sponsored my participation in conferences. The conferences I have attended both domestically and overseas have broadened my knowledge and mind enormously.

Thanks to all those who helped me in one way or another during my PhD study: Professor Andrzej Goscinski, Professor Brian Garner, Professor Geoffrey Webb, Dr. Bingbing Zhou, Dr. Yuefeng Li, Ms Li Liu, Mr. Shichao Zhang, and Dr. Songmao Zhang. Thanks also to the anonymous examiners for their valuable comments and suggestions.

Finally, to my wife Qiong and our daughter Liqi, who mean everything to me: thank you.
List of Publications

The following is a list of my research papers published in referred international conference proceedings or journals during my PhD study at Deakin University:


Contents

Abstract iii
Acknowledgments vi
List of Publications vii

1 Introduction 1
  1.1 Basic Concepts of Decision Making and Complex Decision Making 3
  1.2 Complex Decision Making Needs Hybrids 7
  1.3 Suitability of the Multi-Agent Perspective for Hybrids 9
  1.4 An Agent-Based Hybrid Framework for Complex Decision Making 12
  1.5 Key Research Issues in the Agent-Based Hybrid Framework 15
  1.6 Principal Contributions of the Thesis 16
  1.7 Thesis Outline 18

2 Literature Review and Related Work 22
  2.1 Typical Intelligent Techniques for Complex Decision Making 23
  2.2 Agent-Oriented Software Engineering 26
    2.2.1 Traditional Methodologies 27
    2.2.2 The Gaia Methodology 28
    2.2.3 The Coordination-Oriented Methodology 28
  2.3 Approaches to Incorporating Intelligent Techniques into Agents 29
  2.4 Approaches to Agentification 33
  2.5 Multi-Agent Systems in Finance 35
  2.6 Summary 37
3 An Agent-Oriented Methodology for Hybrid Intelligent Systems 39
3.1 Hybrids are Complex .................................................. 40
3.2 Current Practice in Typical Hybrid Intelligent System Development 42
3.3 Agent-Oriented Approaches Suitable for Hybrids ..................... 44
  3.3.1 Agent-Oriented Decomposition, Abstraction, and Organization 44
  3.3.2 The Advantages of Using Middle Agents ....................... 47
  3.3.3 Object-Oriented Vs Agent-Oriented ............................ 48
3.4 Methodology for Analysis and Design Agent-Based Hybrids .......... 50
  3.4.1 Outline of the Methodology .................................. 50
  3.4.2 The Role Model ............................................... 53
  3.4.3 The Interaction Model ...................................... 56
  3.4.4 Organizational Rules ....................................... 58
  3.4.5 The Agent Model ............................................ 59
  3.4.6 The Skill Model ............................................ 59
  3.4.7 The Knowledge Model ...................................... 60
  3.4.8 Organizational Structures and Patterns ....................... 60
3.5 Summary and Discussion ........................................... 62

4 An Agent-Based Intelligent Technique Society for Decision Making 63
4.1 Requirement Statements ............................................ 65
4.2 Analysis of Agent-Based Intelligent Technique Society ............... 66
4.3 Design of the Society ............................................. 72
4.4 Architecture of the Society ....................................... 74
4.5 Implementation of the Society .................................... 77
  4.5.1 Technology Platform ...................................... 77
  4.5.2 Internal Structures of Agents ............................... 80
4.6 Case Study – Determining a Client’s Investment Policy .............. 83
  4.6.1 Description of the Problem .................................. 84
  4.6.2 More about KQML .......................................... 85
  4.6.3 Converting Legacy Programs into Agents ..................... 86
  4.6.4 Experiments and Results .................................. 87
4.7 Summary ............................................................ 91
7.4 Reasoning with Multimedia Information Using Symbolic Projection 154
  7.4.1 Description of the Problem ........................................ 155
  7.4.2 A Brief Introduction to Symbolic Projection .................. 158
  7.4.3 2D String Representation of Moving Average Charts .......... 160
  7.4.4 Special Considerations for the Problem ....................... 161
  7.4.5 Discussion .......................................................... 163
  7.5 Summary .............................................................. 166

8 Decision Aggregation in Multi-Agent Systems 167
  8.1 Description of the Aggregation Problem .......................... 168
  8.2 Choosing Appropriate Aggregation Operations .................... 169
    8.2.1 Fuzzy Averaging .................................................. 171
    8.2.2 Ordered Weighted Averaging .................................. 174
    8.2.3 Discussion ...................................................... 177
  8.3 Approaches to Implementing Decision Aggregation in Multi-Agent Systems ..................................................... 178
    8.3.1 The Stationary Agent Approach ................................ 178
    8.3.2 The Token Passing Approach .................................. 180
    8.3.3 The Mobile Agent Approach ................................... 182
    8.3.4 Comparison of the Three Approaches .......................... 184
  8.4 Case Study: Financial Portfolio Selection ........................ 185
    8.4.1 Portfolio Selection Models ................................. 186
    8.4.2 Agent Architecture for Portfolio Selection ............... 189
    8.4.3 Experimental Results .......................................... 190
    8.4.4 Empirical Evaluation of the Aggregated Results .......... 192
  8.5 Summary .............................................................. 195

9 Concluding Remarks 197
  9.1 The Prototype System: Pulling It All Together ................. 197
    9.1.1 A Typical Scenario for Investment .......................... 198
    9.1.2 Example ......................................................... 199
  9.2 A Comparison of the Framework with Related Work ............. 203
  9.3 Evaluation .......................................................... 205
9.4 Conclusions ......................................................... 207
9.5 Future Work ......................................................... 209

Bibliography .......................................................... 211
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Property Assessment of Typical Intelligent Techniques</td>
<td>26</td>
</tr>
<tr>
<td>3.1</td>
<td>Operators for Liveness Expressions</td>
<td>56</td>
</tr>
<tr>
<td>4.1</td>
<td>Services in Skill Model</td>
<td>73</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of Agent Development Tools</td>
<td>80</td>
</tr>
<tr>
<td>4.3</td>
<td>Agents in Determining Client's Investment Policy Example</td>
<td>90</td>
</tr>
<tr>
<td>6.1</td>
<td>Description of Benchmark Problems</td>
<td>118</td>
</tr>
<tr>
<td>6.2</td>
<td>Mapping Results between Distance and Satisfactory Degree</td>
<td>119</td>
</tr>
<tr>
<td>6.3</td>
<td>Track Records with $n = 40$</td>
<td>124</td>
</tr>
<tr>
<td>6.4</td>
<td>Agents' Scores with Different Track Records</td>
<td>125</td>
</tr>
<tr>
<td>6.5</td>
<td>Change in Consumer Price Index</td>
<td>126</td>
</tr>
<tr>
<td>6.6</td>
<td>Change in Gross National Product in 1982 US Dollars (Billions)</td>
<td>127</td>
</tr>
<tr>
<td>6.7</td>
<td>Change in M2 Money Supply in 1982 US Dollars (Billions)</td>
<td>127</td>
</tr>
<tr>
<td>6.8</td>
<td>Personal Wealth in 1982 US Dollars (Billions)</td>
<td>128</td>
</tr>
<tr>
<td>6.9</td>
<td>Two-Quarter Moving Average Change in T-Bill Discount Rate</td>
<td>128</td>
</tr>
<tr>
<td>6.10</td>
<td>Predicting Results on Benchmark Problems</td>
<td>129</td>
</tr>
<tr>
<td>8.1</td>
<td>Comparison of the Three Approaches</td>
<td>185</td>
</tr>
<tr>
<td>8.2</td>
<td>Returns on Nine Securities</td>
<td>191</td>
</tr>
<tr>
<td>8.3</td>
<td>Portfolios Based on Three Models and Aggregated Portfolio</td>
<td>192</td>
</tr>
<tr>
<td>8.4</td>
<td>Portfolios and Variances Based on 12 Years Return Data</td>
<td>193</td>
</tr>
<tr>
<td>8.5</td>
<td>Realized Average Returns of the Portfolios (%)</td>
<td>193</td>
</tr>
<tr>
<td>8.6</td>
<td>Returns on Twelve Securities from ASX</td>
<td>194</td>
</tr>
<tr>
<td>8.7</td>
<td>Portfolios and Variances Based on ASX 8 Years Return Data</td>
<td>194</td>
</tr>
<tr>
<td>8.8</td>
<td>Realized Average Returns of the Portfolios Based on ASX Data (%)</td>
<td>195</td>
</tr>
</tbody>
</table>
List of Figures

1.1 View of a Canonical Complex System ............................................. 11
1.2 Canonical View of a Multi-Agent System ........................................ 11
1.3 An Overall Agent-Based Framework for Complex Problems .................. 12
1.4 Agent-Based Hybrid Framework for Complex Decision Making .......... 14
2.1 Architecture of PREDICTOR ............................................................ 31
2.2 Agent Building Blocks of IMARDA .................................................. 32
3.1 Hybrid Intelligent System Development Cycle ................................. 43
3.2 Relationships between Models ....................................................... 53
3.3 Template for Role Schemata ............................................................ 54
3.4 Schema for Role PRICEWATCHER .................................................. 57
3.5 The GetPrice Protocol Definition .................................................. 58
4.1 Schema for Role USERHANDLER .................................................... 67
4.2 Schema for Role WORKPLANNER ................................................... 67
4.3 Schema for Role CAPABILITYRECORDER ....................................... 68
4.4 Schema for Role CAPABILITYMATCHER ........................................ 68
4.5 Schema for Role DECISIONMAKER ............................................... 69
4.6 Schema for Role HELP PROVIDER ............................................... 69
4.7 Schema for Role DECISIONAGGREGATOR ....................................... 70
4.8 Schema for Role USER ................................................................. 70
4.9 Definition of Protocols Associated with the DECISIONMAKER Role:
   (a) ReceiveTask, (b) GetInformation, (c) AskforHelp, and (d) Inform-
   DecisionAggregator ......................................................................... 71
4.10 The Agent Model of the Society ..................................................... 72
4.11 Architecture of Agent-Based Intelligent Technique Society ............ 76
4.12 Decision Making Agent Structure ........................................ 81
4.13 Serving Agent Structure .................................................. 81
4.14 Intelligent Technique Agent Structure .............................. 81
4.15 Aggregation Agent Structure ............................................ 82
4.16 Planning Agent Structure ................................................ 82
4.17 Interface Agent Structure ............................................... 82
4.18 Practical Architecture for Determining Investment Policy ...... 88
4.19 Intelligent Technique Agents Advertise Capabilities to Serving Agent .................................................. 90
5.1 Financial Ontology Conceptual Structure (Example) .......... 98
6.1 Curves of Benchmark Values, SC-Agent_NN, and SC-Agent_FLGA ... 130
6.2 Distances with Benchmark Values .................................... 130
7.1 Terms of Input (a) annual income, (b) total net-worth, and (c) risk tolerance .................................................. 139
7.2 An Example of a Two-Dimensional FAM Matrix ................. 140
7.3 Speedups on the PVM ...................................................... 143
7.4 Three Levels of Multimedia Information Processing .......... 146
7.5 A Framework for Multimedia Information Fusion (Based on [21], pp. 107-109) .................................................. 150
7.6 Example Moving Average Chart ........................................ 155
7.7 Moving Averages of Security Price .................................... 157
7.8 Retrieved Moving Average of a Specific Security ............... 157
7.9 (a) Moving Average Chart Segmentation Using Cutting Lines; (b) Moving Average with Benchmark Object; (c) Final Moving Average Segmentation with Benchmark Object .................................................. 160
7.10 Moving Average with Different Slopes .............................. 161
7.11 Revenue of a Company .................................................. 163
7.12 Trading Volumes of Securities ......................................... 164
7.13 Boundary Lines in a Pie Chart ........................................ 164
8.1 Three Conflicting Agents' Decisions for Investment Policy ...... 173
8.2 Aggregation by Aggregation Agent .................................... 179
8.3 Aggregation by Token Passing ......................................... 180
8.4 Mobile Aggregation Agent Model ..................................... 183
8.5 Architecture for Portfolio Selection .......................... 190
8.6 Practical Architecture for Portfolio Selection .................. 190
9.1 Example Input Screen of the Prototype ......................... 199
9.2 Example Output Screen of the Prototype ........................ 200
Chapter 1

Introduction

Making decisions effectively has far-reaching influence on a nation, an organization, or a person. Electronic commerce and the Internet have created demand for automated systems that can make complex decisions utilizing information from multiple sources. A decision is an allocation of resources. It is made by the decision maker, who has authority over the resources being allocated. The decision maker should have alternative courses of action that he or she might take available at the time of the decision. Evaluating and choosing among alternative actions is usually called decision making [85]. Complex decision making is decision making on complex problems that are uncertain, dynamic, distributed, and heterogeneous in nature.

Good decisions are mainly based on adequate information, rich knowledge, and appropriate skills to use knowledge and information. For many complex problems, to make good and timely decisions is not an easy task to accomplish. In the contemporary information society, there is too much information available for one problem, and it changes rapidly. This makes manually gathering relevant information for a problem nearly impossible. On the other hand, one decision maker's knowledge is very limited. To deal with complex problems, the efforts and opinions of many decision makers are required. Furthermore, some knowledge that is crucial for decision making lies in large databases scattered on the Internet. Such knowledge is hard to discover manually. For these different components or sub-tasks, different types of processing are required. With all these observations in mind, it is
apparent that a great diversity of intelligent techniques and their combinations are needed for complex problem solving and decision making. That is, hybrid solutions are crucial for complex problem solving and decision making. However, the design and development of hybrid intelligent systems is difficult because they have a large number of parts or components that have many interactions. Existing software development techniques (for example, object-oriented analysis and design) cannot manage these complex interactions efficiently as these interactions may occur at unpredictable times, for unpredictable reasons, between unpredictable components.

An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [1]. Agent-oriented techniques represent an exciting new means of analyzing, designing and building complex software systems. They have the potential to significantly improve current practice in software engineering and to extend the range of applications that can feasibly be tackled [73]. A multi-agent system (MAS) is a collection of autonomous agents that work together to solve problems that are beyond the capabilities of individual agents. MASs are good at complex, dynamic interactions. Thus the multi-agent perspective is suitable for modeling, designing, and constructing hybrid intelligent systems. To this end, an overall framework for complex problem solving and decision making is proposed (see Section 1.4).

While agent-based computing is rapidly emerging as a powerful technology for the development of complex software systems, only a few complete and well-grounded methodologies for the analysis and design of multi-agent systems have been proposed so far.

With this observation in mind, we start from tailoring a methodology for analyzing and designing agent-based hybrid intelligent systems.

Follow the methodology, an agent-based hybrid framework for complex decision making (called agent-based intelligent technique society) was proposed and then applied to financial investment planning. Some related problems were solved (see Section 1.5).

Under this framework (see Section 1.4), there are three main types of agents: decision making agents, intelligent technique agents, and serving agents.
decision making agents are agents without embedding intelligent techniques. They are at the front end of a MAS. Intelligent technique agents are at the back end of a MAS. They provide decision making agents with intelligent technique capabilities. The serving agent is a matchmaker — one kind of middle agent. The ontology is the foundation for agent communication. All agents in the society interpret the content of received messages based on the ontology.

This thesis attempts to:

- justify why the multi-agent perspective is suitable for complex decision making such as financial investment planning;
- tailor a methodology for the analysis and design of agent-based hybrid intelligent systems from a few available MAS analysis and design approaches;
- develop an agent-based hybrid framework for complex decision making according to the methodology;
- build an ontology for finance to facilitate the communication among agents as well as the matchmaking when applying the framework to financial investment planning;
- find out new ways to improve the performance of matchmaking in middle agents;
- use symbolic projection theory [25] to solve a sub-problem of reasoning with multimedia information; and
- choose appropriate aggregation algorithms to aggregate different decisions and implement them in MASs.

1.1 Basic Concepts of Decision Making and Complex Decision Making

A decision is an allocation of resources. It can be likened to buying some securities and trading them in stock market. It is irrevocable, except that a new decision may reverse it.
A decision is made by the decision maker, who has authority over the resources being allocated. Presumably, he or she makes the decision in order to further some objective, which is what he or she hopes to achieve by allocating the resources.

At the time of the decision, the decision maker has available some alternatives (at least two), which are the courses of action that he or she might take. When choosing an alternative and committing to it (i.e., buying and trading the securities), he or she has made the decision.

The work of managers, of scientists, of engineers, of lawyers—the work that steers the course of society and its economic and governmental organizations—is largely the work of making decisions and solving problems. It is work of choosing issues that require attention, setting goals, finding or designing suitable courses of action, and evaluating and choosing among alternative actions. The first three of these activities—fixing agendas, setting goals, and designing actions—are usually called problem solving; the last, evaluating and choosing, is usually called decision making [85]. The discussions in this thesis are focused on decision making.

Nothing is more important for the well-being of society that the work of making decisions and solving problems be performed effectively. Many problems requiring attention at the national level (the budget and trade deficits, AIDS, national security, the mitigation of earthquake damage) must be addressed successfully, similarly at the level of business organizations (product improvement, efficiency of production, choice of investments), and at the level of individual lives (choosing a career or a school, buying a house) [85].

To make a decision efficiently and effectively, the decision maker must have rich knowledge and adequate information about the problems, as well as proficient skills to use knowledge and information. Financial investment planning is a good example [53].

Financial planning is a profession which requires specialized knowledge across diverse areas including investments, risk management, taxation, estate planning, social security, and retirement planning. Financial planners (decision makers) can provide advice only if they have a comprehensive understanding of the needs and goals of their clients and the attitudes of their clients toward risk, investments, insurance, retirement planning, estate issues, social security issues, and taxation.
Thus financial planners need to collect clients' personal information.

To prepare a set of investment recommendations, financial planners also need to gather other financial information such as stock technical analysis data. They need to have a strong working knowledge of fundamental mathematical concepts that relate to investment. The skills required include a basic understanding of the nature of compounding and the time value of money. An understanding of these concepts will enable financial planners to determine both present and future values of capital amounts and to access alternative income streams so that different investment options can be compared.

From the above analysis, it is clear that the financial planning task is not easy to accomplish. It requires financial planners to have expertise about investment, sufficient information about clients and financial markets, and related skills using the knowledge and information to give advice.

In short, rich knowledge and adequate information about the problems as well as proficient skills to use knowledge and information are key factors in decision making and problem solving. To improve the decision making and problem solving capabilities, decision makers and problem solvers must do their best to possess rich knowledge of the problems, gather adequate information about the problems, and use relevant knowledge and information flexibly. This is not an easy thing to do, especially for complex problems.

Many problems related to open environments such as the Internet are difficult to deal with – either because no existing technology can be used to solve the problems, or because it is considered too expensive, difficult, time-consuming, or risky to develop solutions using existing technology. In this context these are called complex problems. In order to identify the principal characteristics of complex problems, it is necessary to take a closer look at the typical open environment – the Internet. There are three dominant characteristics of the Internet [12, 13]:

- Information available from the Internet is unorganized, multi-modal, and distributed on server sites all over the world;

- The number and variety of data sources and services increase dramatically every day. Furthermore, the availability, type, and reliability of information services are constantly changing; and
• Information is ambiguous and possibly erroneous due to the dynamic nature of the information sources and potential information updating and maintenance problems.

Therefore, information is becoming increasingly difficult for a person or computer system to collect, filter, evaluate, and use in decision making and problem solving. Meanwhile, with the knowledge and information explosion, the capabilities of individual decision makers or agents are limited. Complex decision making requires the efforts and opinions of many decision makers (agents).

Based on these observations, the typical characteristics of complex problems can be identified. They are: uncertain (e.g., there is much uncertain in knowledge, information, and decision), dynamic (e.g., information and decision keep changing), distributed (e.g., information and knowledge scatter in different sources), and heterogeneous (e.g., information and knowledge may have different representations and in different media forms). Complex decision making is decision making on complex problems.

Financial investment planning is a typically complex problem. Take one sub-task of financial planning — financial portfolio management — as an example. The task environment has many interesting features, including [4]:

• the enormous amount of continually changing, and generally unorganized, information available;

• the variety of kind of information that can and should be brought to bear on the task (market data, financial report data, technical models, analysts' reports, breaking news, etc.); and

• the many sources of uncertainty and dynamic change in the environment.

Thus financial planning problem is uncertain, dynamic, distributed, and heterogeneous in nature. Due to the characteristics of complex problems such as financial investment planning, it is very difficult to make timely, good decisions on complex problems.

"Rapidly changing" may best describe today's network computing environment. An ever-increasing number of individuals and businesses are being added to the Internet daily. But, as new data and services are added, others change, move, or
disappear. In this dynamic, unstable environment, gathering the relevant information for a complex problem manually and timely is impractical, if not impossible. On the other hand, one decision maker's knowledge is very limited. To deal with complex problems, the efforts and opinions of many decision makers (agents) are required. They must work as a group or a team, coordinate or cooperate with each other to achieve goals. Furthermore, some knowledge crucial to decision making lies in large databases scattered on the Internet. It is only possible to discover such knowledge (association rules etc.) by data mining techniques. Analyzing these databases manually is an ordeal for most people.

With all these observations in mind, obviously, a great diversity of intelligent techniques are needed to perform complex decision making automatically or semi-automatically.

1.2 Complex Decision Making Needs Hybrids

In complex decision making, many different components or sub-tasks are involved, each of which requires different types of processing. Because of this, many techniques were developed for complex decision making. These techniques can be divided into two categories: (1) traditional hard computing techniques including operations research, system science/engineering, expert systems, and (2) soft computing techniques including fuzzy logic (FL), neural networks (NN), and genetic algorithms (GA). The techniques in both categories are called intelligent techniques in this thesis.

While there is now an array of different types of intelligent techniques, each technique has particular strengths and limitations and cannot be successfully applied to every type of problem (refer to Section 2.1). For example, in a decision making task that requires explicit explanations, NNs are less applicable than a rule induction approach. Similarly, for tasks that require constant adaptation and learning from the operating environment, a static expert system is far less useful than an adaptive technique such as a NN. These techniques and methodologies are complementary rather than competitive and thus must be used in combination and not exclusively. This allows us to take advantage of their respective component
strengths and compensate for each other's weaknesses.

Within soft computing, each of the constituent techniques has a set of capabilities to offer. In the case of FL, it is a body of concepts and techniques for dealing with imprecision, information granularity, approximate reasoning and, most importantly, computing with words. In the case of NN, it has the capability for learning, adaptation and identification. In the case of GA, it has the capability to employ systematized random search and achieve optimal performance, and so on.

Hybrid intelligent systems (hybrids in short) are computational systems that integrate different intelligent techniques in these two categories. These systems are now being used to support decision making in a wide variety of tasks [99, 149]. Hybrid intelligent systems allow the representation and manipulation of different types and forms of data and knowledge which may come from various sources. Refined system knowledge is used during reasoning and decision making processes producing more effective results.

As pointed out in Section 1.1, the decision makers (agents) must have great skills to use the knowledge related to a problem and process the relevant information. This includes the capability to deal with imprecise, uncertain or vague information. Dealing with real-world uncertainty plays a very important role in decision making. In order to make good decisions, the agents must have the ability to deal with imprecision information, achieve optimal performance, and be adaptive, that is, they should have high a MIQ (machine intelligence quotient [10]). In Zadeh's view [10], most high MIQ systems are hybrid intelligent systems that use soft computing techniques such as FL, NN, and GA etc. in some combination.

From the discussions of this section, it is concluded that hybrid intelligent systems are required for complex decision making.

However, the design and development of hybrid intelligent systems is difficult because they have a large number of parts or components that have many interactions. Existing software development techniques (for example, object-oriented analysis and design) cannot manage these complex interactions efficiently as these interactions may occur at unpredictable times, for unpredictable reasons, between unpredictable components. At this stage, agent techniques are very promising to take on the responsibilities.
1.3 Suitability of the Multi-Agent Perspective for Hybrids

Agent techniques represent an exciting new means of analyzing, designing and building complex software systems. They have the potential to significantly improve current practice in software engineering and to extend the range of applications that can feasibly be tackled [73].

Although a precise definition of an intelligent agent is still forthcoming, an increasing number of researchers find the following characterization useful [1, 2]:

"An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives."

When adopting an agent-oriented view of the world, it soon becomes apparent that most problems require or involve multiple agents; to represent the decentralized nature of the problem, the multiple loci of control, the multiple perspectives or the competing interests. Moreover, the agents will need to interact with one another, either to achieve their individual objectives or to manage the dependencies that ensue from being situated in a common environment. A multi-agent system can be defined as a loosely coupled network of entities that work together to make decisions or solve problems that are beyond the individual capabilities or knowledge of each entity [14]. These entities - agents - are autonomous and may be heterogeneous in nature. The characteristics of multi-agent systems are [7]:

- each agent has incomplete information, or capabilities for making a decision or solving the problem, thus each agent has a limited viewpoint;
- there is no global system control;
- data is decentralized; and
- computation is asynchronous.

In this thesis, the terms "agents" or "software agents" are used to indicate "intelligent agents that can interact". "Intelligent" indicates that the agents pursue their goals and execute their tasks such that they optimize some given performance
measures ([8], pp. 2-3). To say that agents are intelligent does not mean that they are omniscient or omnipotent, nor does it mean that they never fail. Rather, it means that they operate flexibly and rationally in a variety of environmental circumstances, given the information they have and their perceptual and effectual capabilities.

From a multi-agent perspective, agents in multi-agent systems are autonomous and can engage in flexible, high-level interactions. Considering the autonomous nature of agents, autonomy means that the agents have their own persistent thread of control (i.e., they are active) and that they decide for themselves which actions they should perform at what time. The fact that agents are active means they know for themselves when they should be acting and when they should update their state. The flexible nature of interactions means that agents can make decisions about the nature and scope of interactions at run-time rather than design time.

Multi-agent systems are ideally suited to representing problems that have multiple problem solving methods, multiple perspectives and/or multiple problem solving entities. Such systems have the traditional advantages of distributed and concurrent problem solving, but have the additional advantage of sophisticated patterns of interactions. Examples of common types of interactions include cooperation, coordination, and negotiation. It is the flexibility and high-level nature of these interactions which distinguishes multi-agent systems from other forms of software and which provides the underlying power of the paradigm.

Furthermore, N. Jennings defined the canonical views of a complex system and a multi-agent system [74]. In the canonical view of a complex system (see Figure 1.1), the system’s hierarchical nature is expressed through the “related to” links, components within a subsystem are connected through “frequent interaction” links, and interactions between components are expressed through “infrequent interaction” links.

In the canonical view of a multi-agent system (see Figure 1.2), it can be seen that adopting an agent-oriented approach to software engineering means decomposing the problem into multiple, autonomous components that can act and interact in flexible ways to achieve their set objectives. The key abstraction models that define the agent-oriented mind-set are agents, interactions, and organizations. Finally,
explicit structures and mechanisms are often used to describe and manage the complex and changing web of organizational relationships that exist between the agents.

From this discussion, it is apparent that multi-agent perspectives are well-suited for modeling hybrid intelligent systems for complex decision making.

In this thesis, agent-based systems are also used to refer to systems that are both designed and implemented as several interacting agents, i.e., multi-agent systems. When discussing agent techniques from software engineering point of view and comparing them with object-oriented techniques, agent-oriented is used.
1.4 An Agent-Based Hybrid Framework for Complex Decision Making

From the previous discussions, rich knowledge, adequate information, and proficient skills are needed for complex problem solving and decision making. The multi-agent perspective is well suited for modeling complex problems. To this end, an overall multi-agent framework for complex problem solving and decision making is proposed (see Figure 1.3).

![Diagram](image)

Figure 1.3: An Overall Agent-Based Framework for Complex Problems

In this framework, there is stress on harmonizing and integrating knowledge, information, and skills for complex problem solving and decision making. There are four principal components in this system. Information gathering agents are responsible for gathering relevant information on the Internet. Data mining agents are in charge of discovering knowledge from retrieved information as well as other relevant databases. Decision making agents can effectively use available knowledge and appropriate information to make reasonable decisions. User agents interact with users. Information gathering agents, data mining agents, and decision making agents themselves constitute three different MASs. A scenario goes as follows [24]: the user queries the system through the user interface module of a user agent.
CHAPTER 1. INTRODUCTION

The user interface module converts the user's inquiry to some kind of internal representation. The decision making agents then ask information gathering agents to collect some information according to the user's inquiry on the Internet. The retrieved information is stored in a database. Data mining agents can extract some knowledge (positive and/or negative association rules etc.) from the retrieved information as well as other databases and add the knowledge to knowledge bases of decision making agents. Data mining agents can work on-line or off-line. The decision making agents make decisions according to the user's inquiry, the collected information, and their own domain knowledge. The final alternative decisions are provided to the user by the result visualization module of user agents. Communications among different agents are based on some kinds of ontologies.

Compared with related work such as Retsina [12, 13] and InfoSleuth [20] [21] (pp. 11-14) frameworks, attention is paid to not only information but also knowledge and skills for complex problem solving and decision making. Both Retsina and InfoSleuth were focused on information. As many research topics are involved in such a framework, this thesis focuses on problems related to decision making agents (the dash line part in Figure 1.3). Compared with task agents in Retsina and task execution agents in InfoSleuth, decision making agents in our framework put more emphasis on knowledge and skills, especially the skills to use knowledge and information.

According to the discussions in Section 1.2, employing synergism of intelligent techniques is crucial for complex decision making. It is vital to integrate different intelligent techniques into the decision making subsystem in the overall framework for complex problems (refer to Figure 1.3), while multi-agent perspectives are well suited for managing the integration. This results in an agent-based hybrid framework for complex decision making (Figure 1.4). Figure 1.4 is the detailed result of the decision making subsystem in Figure 1.3. Hybrid framework means different intelligent techniques are combined together in the framework. How to develop the framework and solve the related problems are the key topics of this thesis. Under this framework, there are three principal types of agents: decision making agents, intelligent technique agents, and serving agents.

Sycara et al. distinguish three general agent categories: service provider agents,
service requester agents, and middle agents [138]. Service provider agents provide some types of services, such as finding information, or performing some particular domain specific problem solving. Requester agents need provider agents to perform some services for them. Agents that match the requested services with the provided services are called middle agents [39]. Based on this classification, decision making agents here are service requester agents without intelligent technique capability. They are at the front end of a MAS. Intelligent technique agents are service provider agents, which are at the back end of a MAS. They provide decision making agents with intelligent technique capabilities. The serving agent is a matchmaker – one kind of middle agent. The ontology is the foundation for agent communication. All agents in the framework interpret the content of received messages based on the ontology.
1.5 Key Research Issues in the Agent-Based Hybrid Framework

The quality of decisions (the outcomes of decision making agents) is highly reliant on the knowledge, information, and skills that decision making agents possess for a specific problem. The smarter the decision making agents, the better the decisions. It is possible to increase the degree of "intelligence" (L. A. Zadeh called it MIQ) of decision making agents to make them smarter. As stated in Section 1.2, one efficient and most promising way to achieve high MIQ is to employ a combination of different intelligent techniques such as expert systems, FL, NNs, and GAs. On the other hand, many different components or sub-tasks are involved in complex decision making, each of which may require different types of processing. For example, in a financial planning application, a NN can be used as a pattern watcher for stock market; a GA can be used to predict interest rate; and the approximate reasoning based on FL can be used to evaluate the client's financial risk tolerance ability. Furthermore, hybrid intelligent systems allow the representation and manipulation of different types and forms of data and knowledge which may come from various sources. Refined system knowledge is used during reasoning and decision making processes producing more effective results. Hence in order to make good decisions, agent-based hybrid intelligent systems are essential for complex decision making. How to incorporate intelligent techniques into MASs (decision making agents) is one problem that needs a solution. There is a need to extract and tailor a methodology suitable for the analysis and design of agent-based hybrid intelligent systems, which is based on the available agent-oriented analysis and design approaches. Also needed is to provide a framework that can facilitate the construction of agent-based hybrid intelligent systems following the methodology.

To facilitate the communication among agents in the agent-based intelligent technique society (a multi-agent system) as well as the matchmaking of middle agents in the society, a domain-specific ontology – financial ontology – has been built.

To increase the flexibility and robustness of the society, a middle agent is employed to assist in locating and connecting the ultimate service provider with the
ultimate service requester. The performance of middle agents heavily relies on the matchmaking algorithms used. How to improve the performance of *matchmaking in middle agents* must be dealt with.

Adequate information is crucial for making good decisions. When making decisions, decision making agents will utilize the information gathered by information gathering agents. The gathered information is not only in text form, but also in image, audio, or video etc. multimedia forms. Therefore, decision making agents should be equipped with multimedia information processing abilities. Actually, decision making agents will use the gathered multimedia information as “known facts”. They will then do some reasoning to obtain further conclusions based on these multimedia form “known facts”. This *reasoning with multimedia information* problem also needs to be tackled. Reasoning with still image information is solved by using symbolic projection theory [25].

In the decision making MAS, each agent may only have a limited amount of domain knowledge or information – just like our human experts, someone can be an expert in certain fields, but not all. An agent can make decisions based on its existing knowledge and its inherent subjectivity. Thus, it is unlikely that agents’ opinions in a multi-agent system are identical. Usually these opinions are either close or conflicting to various degrees. They have to be combined or reconciled in order to produce one decision. This multi-agent decision-making procedure is called aggregation. Meanwhile, analysis of complex problems requires the efforts and opinions of many experts (agents). In such cases, there is a need to aggregate the different decisions from different decision making agents to produce one final decision, that is, decision aggregation is an important issue in MASs. Thus *decision aggregation* problems must be investigated. We focus on choosing and evaluating appropriate aggregation algorithms to aggregate different decisions and implementing them in MASs. These are some of the key issues addressed in this thesis.

### 1.6 Principal Contributions of the Thesis

The major contributions of this thesis include:
• A methodology suitable for the analysis and design of agent-based hybrid intelligent systems is extracted and tailored from available agent-oriented analysis and design approaches;

• A general framework, called agent-based intelligent technique society, is proposed to facilitate the construction of agent-based hybrid intelligent systems. The framework has four crucial characteristics that differentiate this work from other hybrid intelligent systems:

  – The ability to exchange comprehensible communications (interactions at knowledge level [54]);

  – Each service requester (decision making) agent can easily access all the intelligent techniques available in the system;

  – The presence of the serving agent in the framework allows adaptive system organization. For example, if it is clear that one hybrid soft computing agent (one kind of intelligent technique agents) can do a better job than a single technology soft computing agent, the single soft computing technology agent can be deleted and the hybrid agent added simply by adding or deleting a record in the database of the serving agent;

  – Overall system robustness is facilitated through the use of the serving agent. For example, if a particular intelligent technique service provider (intelligent technique agent) disappears, a requester agent (problem solving agent) can find another one with the same or similar capabilities by interrogating the serving agent.

• To verify the framework, a prototype of the intelligent technique society is implemented and applied to financial investment planning. A financial ontology is built to ensure the agents in the prototype communicate effectively, and to support the matchmaking of the middle agent in the framework;

• The ways to improve the performance of middle agents are investigated. These ways consist of the acquisition of agents’ track records and the use
of track records. It is highlighted that the practical performance of service provider agents has a significant impact on the matchmaking outcomes of middle agents;

- The parallelism in approximate reasoning based on fuzzy logic is explored. A fast parallel approximate reasoning algorithm is implemented and incorporated into the intelligent technique society;

- Three levels of multimedia information processing are identified. Reasoning with still image information problem is solved by using symbolic projection theory;

- How to choose appropriate aggregation algorithms for a specific application is demonstrated and the aggregated results are evaluated. Three approaches to implementing decision aggregation algorithms in MASs are investigated. Their pros and cons and applicability are analyzed.

1.7 Thesis Outline

The rest of the thesis consists of 8 chapters. All the chapters in the thesis can be divided into two levels. The upper level is the work toward the generic methodology and framework, which contains Chapters 2, 3, and 4. Chapter 2 is literature review and related work. Chapter 3 addresses agent-oriented methodology for the analysis and design of agent-based hybrid intelligent systems. Based on the methodology proposed in Chapter 3, Chapter 4 discusses an agent-based hybrid framework to construct agent-based hybrid intelligent systems for complex decision making. The lower level is the work applying the methodology and framework proposed in upper level to financial investment planning. Some problems including ontology building, matchmaking in middle agents, reasoning in decision making, and aggregating decisions, which are related to the agent-based hybrid framework for complex decision making, are investigated in Chapters 5, 6, 7, and 8, respectively. Chapter 9 consists of concluding remarks. An overview of each chapter is as follows:

- Chapter 2, Literature review and related work, briefly presents the advantages and disadvantages of typical hard computing and soft computing techniques
for complex decision making. Agent-oriented analysis and design methodologies in current practice of agent-oriented software engineering are summarized. A brief survey of typical agent-based hybrid intelligent systems, approaches to incorporating intelligent techniques into agents, and agent technology in finance is provided. Typical approaches to converting legacy intelligent technique software packages into agents are also given.

- Chapter 3, An agent-oriented methodology for hybrid intelligent systems, argues that agent-oriented approaches are well suited for constructing hybrid intelligent systems. A methodology suitable for analysis and design agent-based hybrid intelligent systems is extracted and tailored based on the current practice of agent-oriented software engineering. Some material in this chapter is also reported in [29].

- Chapter 4, An agent-based intelligent technique society for decision making, discusses the analysis, design, and implementation of a general framework of agent-based hybrid intelligent systems according to the methodology proposed in Chapter 3. This framework is called the agent-based intelligent technique society. Such a society has four crucial characteristics that differentiate this work from others. The work is also reported in [26, 27]. A case study is presented to demonstrate how to determine a client’s investment policy using the society, and to show how to convert a program written in C/C++ programming language into an agent.

- Chapter 5, Building an ontology for financial investment, explains why there is a need to construct an ontology used in finance and how to construct it. Three problems involving the construction of financial ontology – acquiring knowledge about the ontology, coding the ontology, and accessing the ontology – are discussed. Ontology construction is difficult and time consuming and is a major barrier to the building of intelligent systems and intelligent agents. Based on our ontology building experience and a relatively profound analysis of the current development of ontologies, it is proposed that the next generation of ontology construction tools should include nine capabilities. The material in this chapter is also reported in [28].
• Chapter 6, *Matchmaking in middle agents*, addresses some novel matchmaking algorithms that can overcome the drawbacks of most currently used matchmaking algorithms. Matchmaking in middle agents is a crucial issue in MASs, especially those used in open environments such as the Internet. Currently, there are some efficient matchmaking algorithms, but their matchmaking quality falls far behind the needs of real applications. Thus it is of paramount importance to improve the matchmaking quality of middle agents (while remaining efficient) so that they can pick up the "right" service provider agents. It is argued that the practical performance of service provider agents has a significant impact on the matchmaking outcomes of middle agents. It is suggested to consider the track records of agents in accomplishing delegated tasks. This includes the acquisition of track records and the use of track records. Some results in this chapter are also reported in [100].

• Chapter 7, *Reasoning in decision making*, deals with the reasoning problems for reaching conclusions (decisions) using information only in text form and in multimedia forms. The parallel implementation of approximate reasoning is discussed. The emphasis of this chapter is on how to utilize information in multimedia forms. Three levels of multimedia information processing – storage, retrieval, and post-processing of multimedia information – are presented first. One of the most important topics in multimedia post-processing – reasoning with multimedia information, especially reasoning with still image information – is detailed. Symbolic projection theory is employed to solve this problem. To our knowledge, this problem has not previously been addressed.

• Chapter 8, *Decision aggregation in multi-agent systems*, elaborates why ordered weighted averaging (OWA) aggregation algorithm was chosen and how to implement it in multi-agent systems. Three approaches to implementing decision aggregation in multi-agent systems – the stationary agent approach, the token passing approach, and the mobile agent approach – are presented. The pros and cons of the three approaches are described. The material in this
Chapter 1. Introduction

Chapter is also reported in [30, 31, 32]. A case study for portfolio selection is given to show the decision aggregation process.

- Chapter 9, Concluding remarks, first describes how to pull all case studies together to constitute a working prototype. Comparison with other frameworks is given followed by the evaluation of the framework. The contributions of the presented work and the conclusions obtained are summarized. Some interesting directions for future research are also suggested.
Chapter 2

Literature Review and Related Work

While there is now an array of different types of intelligent techniques for complex decision making, each technique has particular strengths and limitations and cannot be successfully applied to every type of problem. To reveal why hybrid intelligent systems are required in complex decision making, the advantages and disadvantages of typical intelligent techniques in both traditional hard computing and soft computing are outlined in Section 2.1.

Intelligent agents and multi-agent systems represent a new way of analyzing, designing, and implementing complex software systems such as hybrid intelligent systems. The agent-based perspective offers a powerful repertoire of tools, techniques, and metaphors that have the potential to considerably improve the way in which people conceptualize and implement many types of software. However, the development of complex multi-agent systems requires not only new models and technologies, but also new methodologies to support developers in an engineered approach to the analysis and design of such systems. Therefore, the current practice of agent-oriented software engineering will be discussed in Section 2.2.

As stated in Chapter 1, hybrid intelligent systems are essential for complex decision making. Meanwhile, the design and development of these systems is difficult
because they have a large number of parts or components that have many interactions, while MASs are good at complex, dynamic interactions. Hence incorporating intelligent techniques into agents to construct agent-based hybrid intelligent systems is of paramount importance to make good decisions. In Section 2.3 the approaches to integrate intelligent techniques and agents are summarized.

There are many legacy software packages of intelligent techniques available for different applications. When constructing hybrid intelligent systems from the agent-oriented viewpoint, this implies that some techniques should be available for converting such legacy programs into agents. Thus typical approaches to agentification will be presented in Section 2.4.

Financial investment planning is a typical complex problem that is difficult to deal with. Intelligent agent technology holds great promise for such problems. Some typical multi-agent systems used in finance will be examined in Section 2.5. Finally in Section 2.6, the limitations of current research will be summarized and the approaches to overcoming these limitations will be outlined.

2.1 Typical Intelligent Techniques for Complex Decision Making

As pointed out in Section 1.2, there are two main categories of intelligent techniques—traditional hard computing techniques and soft computing techniques. One typical hard computing technique is expert systems, while the principal members of soft computing techniques are fuzzy logic (FL), neural networks (NN), and genetic algorithms (GA). In this section, the four typical intelligent techniques are compared and contrasted on three key information processing capabilities—knowledge acquisition, brittleness, and explanation. This section is mainly based on the discussions in [68] (pp.3-5).

Knowledge Acquisition. Knowledge acquisition is a crucial stage in the development of intelligent systems. As a process, it involves eliciting, interpreting and representing the knowledge from a given domain. Knowledge acquisition for expert systems (from domain experts) is time consuming, expensive and potentially unreliable. Furthermore, expert systems do not have mechanisms to deal with any
changes in their decision making environment – they cannot adapt and learn from changes in their operating environment. Thus the maintenance of knowledge in expert systems is also time consuming and expensive.

Due to these problems, intelligent techniques such as NNs and GAs, which can learn from domain data, have certain advantages. In expert systems, the decision boundaries – the bounds used to make particular decisions – are specified by a domain expert, while in NNs and GAs these decision boundaries are learned. Changes in the operating environment cause the decision boundaries to be shifted or changed. Systems that learn can detect and adapt to these changes.

**Brittleness.** Although there are notable successes of expert systems, many of these systems operate in very narrow domains under limited operational conditions. This phenomena in expert systems is referred as brittleness. The systems are brittle in the sense that they respond appropriately only in narrow domains and require substantial human intervention to compensate for even slight shifts in domain.

An operational view of the brittleness problem can be seen as the inability of an intelligent system to cope with inexact, incomplete or inconsistent knowledge. Causes of this brittleness problem are twofold – inadequate representation structures and reasoning mechanisms. In expert systems, knowledge is represented as discrete symbols and reasoning consists of logical operations on these constructs.

In contrast, reasoning in NNs involves the numeric aggregation of representation over the whole network. This distributed representation and reasoning allows these systems to deal with incomplete and inconsistent data and also allow the systems to gracefully degrade. That is, even if some parts of a NN are made non-operational, the rest of the NN will function and attempt to give an answer. This type of inherent fault tolerance contrasts strongly with expert systems which usually fail to function even if a single processing part is non-operational.

FL deals with the problem of brittleness by adopting novel knowledge representation and reasoning methods. Fuzzy sets, the form in which knowledge is represented, diffuse the boundaries between concepts. There are no sharp divisions where one concept ends and the next begins. This fuzzy data representation, in conjunction with fuzzy reasoning (approximate reasoning) mechanisms, allows the processing of data which are inexact or partially correct.
CHAPTER 2. LITERATURE REVIEW AND RELATED WORK

GAs are able to cope with brittleness. It is the maintenance of a population of solutions which makes GAs and classifier systems non-brittle. Each rule in the classifier system population contains a relationship describing the system being modeled. The system's flexibility arises from the rules representing a wide range of competing, conflicting hypotheses. The selection of the appropriate rule to fire is dependent on its past performance - a statistical aggregation of its correct performance. Similar to NNs, it is this statistical reasoning property, based on the past performance that gives GAs their ability to cope with brittleness.

**Explanation.** The ability to provide users with explanations of the reasoning process is important for complex decision making. Explanation facilities are required both for user acceptance of the decisions made by a system, and for the purpose of understanding whether the reasoning procedure is sound. Good examples of this requirement can be found in medical diagnosis, loan granting, and legal reasoning. There have been fairly successful solutions to the explanation problem by expert systems, symbolic machine learning and case-based reasoning systems. In expert systems, explanations are typically provided by tracing the chain of inference during the reasoning process.

In a FL system the final decision is generated by aggregating the decisions of all the different rules contained in the fuzzy rule base. In these systems a chain of inference cannot be easily obtained, but the rules are in a simple to understand "IF-THEN" format which users can easily inspect.

GAs, especially in the form of classifier systems, can build reasoning models in the form of rules. As in the case of expert systems, it is possible to trace a chain of inference and provide some degree of explanation of the reasoning process.

In contrast, in NNs it is difficult to provide adequate explanation facilities. This is due to NNs not having explicit, declarative knowledge representation structures but instead having knowledge encoded as weights distributed over the whole network. It is therefore more difficult to find a chain of inference which can be used for producing explanations.

Table 2.1 summaries the above computational properties with respect to the four typical intelligent techniques. It is clear that these intelligent techniques are complementary and should be used in combinations for complex decision making.
Table 2.1: Property Assessment of Typical Intelligent Techniques

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Automated Knowledge Acquisition</td>
</tr>
<tr>
<td>Expert Systems</td>
<td>⋆</td>
</tr>
<tr>
<td>Fuzzy Logic</td>
<td>⋆</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>4</td>
</tr>
<tr>
<td>Genetic Algorithms</td>
<td>4</td>
</tr>
</tbody>
</table>

(⋆-weak, 4-strong)

2.2 Agent-Oriented Software Engineering

Building high-quality, industrial strength software to solve complex problems is difficult. To provide software developers with a number of significant advantages over contemporary methods, agents are being advocated as a next generation model for engineering complex, distributed software systems [1, 73]. Some researchers in this field gave a qualitative analysis to provide the intellectual justification of precisely why agent-based systems are well suited to engineering complex software systems [2, 73, 74, 76, 77]. Based on the analysis, they argued that:

- Agent-oriented approaches can significantly enhance our ability to model, design and build complex, distributed software systems;

- As well as being suitable for designing and building complex systems, the agent-oriented approach will succeed as a mainstream software engineering paradigm.

Developing applications in terms of autonomous software agents that exhibit proactive and intelligent behavior, and that interact with one another in terms of high-level protocols and languages, leads to a new programming paradigm. By dint of being a new programming paradigm, the development of agent-based applications implies new programming abstractions, techniques, as well as new methodologies. This section provides a brief overview of the state of the art in the area of software engineering methodologies for multi-agent systems.
2.2.1 Traditional Methodologies

A number of different methodologies have been proposed in recent years for modeling and engineering agents and multi-agent systems [78, 79, 90, 92]. Traditional methodologies for analysis and design are poorly suited to multi-agent systems because of the fundamental mismatch between the respective levels of abstraction. Despite this mismatch, however, several proposals do take object-oriented modeling techniques or methodologies as their basis. On the one hand, some proposals directly extend the applicability of object-oriented methodologies and techniques to the design of agent systems. However, these proposals fail to capture the autonomous and proactive behavior of agents, as well as the richness of their interactions. On the other hand, some proposals seek to extend and adapt object-oriented models and techniques to define a methodology for use in multi-agent systems. This can lead, for example, to extended models for representing agent behavior and their interactions [80, 94], as well as to agent-tuned extensions of UML (Unified Modeling Language) [80]. However, although these proposals can sometimes achieve a good modeling of the autonomous behavior of agents and of their interactions, they lack the conceptual mechanisms for adequately dealing with organizations and agent societies.

A different set of proposals build upon and extend methodologies and modeling techniques from knowledge engineering [81]. These techniques provide formal and compositional modeling languages for the verification of system structure and function. These approaches are well-suited to modeling knowledge- and information-oriented agents. However, since these approaches usually assume a centralized view of knowledge-based systems, they fail to provide adequate models and support for the societal view of multi-agent systems.

Other models and approaches attempt to model and implement multi-agent systems from an "organization-oriented" point of view [82]. These help pave the way toward agent-oriented methodologies by explicitly conceiving multi-agent systems as organizations or as societies. However, these proposals define an organization merely as a collection of interacting roles, thus failing, again, to deal with the key point of social tasks.
2.2.2 The Gaia Methodology

The Gaia methodology [83, 84] is one of the few attempts that is specifically tailored to the analysis and design of multi-agent systems and that deals with both the micro (intra-agent) level and the macro (inter-agent) level of analysis and design. Gaia makes explicit use of an organizational point of view. In Gaia, analysis and design are well-separated phases. Analysis aims to develop an understanding of the system and its structure, in terms of the roles that have to be played in the agent organization and of their interactions, without any reference to implementation details. The design phase aims to define the actual structure of the agent system, in terms of the agent classes and instances composing the system, of the services to be provided by each agent, and of the acquaintances' structure. However, Gaia, as it presently stands, is not a general methodology for all kinds of multi-agent systems. Rather, it is intended to support the development of distributed problem solvers in which the system's constituent components are known at design time (i.e., a closed system) and in which all agents are expected to cooperate toward the achievement of a global goal. For these reasons, Gaia is not suitable for modeling open systems and for controlling the behavior of self-interested agents.

Similar shortcomings also affect most of the recently proposed organization-oriented methodologies. For example, the MASE (Multi-Agent Systems Engineering) methodology [93, 90, 92] provides clean guidelines for developing multi-agent systems, based on a well-defined seven-step process (capturing goals, applying use cases, refining roles, creating agent classes, constructing conversations, assembling agent classes, and system design). This process drives developers from analysis to implementation. However, once again, the design process fails to identify any organizational abstraction other than the role model.

2.2.3 The Coordination-Oriented Methodology

In [78], the authors broaden the scope of Gaia, and indicate that insights from the area of coordination models can be incorporated in order to make it more suitable for developing Internet-based applications. The adoption of a coordination model
as the conceptual abstraction to be exploited in the analysis and design of multi-agent systems for the Internet enables open systems, self-interested agents, and social laws, to find a suitable accommodation. On this basis, the methodological concepts introduced by Gaia can be effectively complemented by the concepts of social laws and coordination media, leading to the definition of a coordination-oriented methodology suitable for multi-agent Internet systems. Yet, to date, the coordination-oriented methodology is far from being well-defined.

To date, the organizational concepts of agent roles and role models have become an important research area in the field of agent-based systems. In [91], Zambonelli, Jennings, and Wooldridge introduced three further organizational abstractions: organizational rules, organizational structures, and organizational patterns. They sketched some general guidelines for a new methodology for the analysis and design of multi-agent systems that is centered around organizational abstractions.

2.3 Approaches to Incorporating Intelligent Techniques into Agents

There are many ways one can incorporate different intelligent techniques such as FL, NNs, and GAs into agents. The applicability of any method depends heavily on one's selection of the agent development language and delivery platforms as well as one's overall agent architecture and, to a lesser extent, the infrastructure over or within which the agent system exists. All these approaches can be divided into two categories: loosely coupled and tightly coupled.

There are three principal methods to incorporate intelligent techniques into individual agents from the implementation point of view [27]:

- Via .DLL (Dynamically Linked Libraries) or other callable APIs (Application Programming Interface);
- Through specific interface agents and stand-alone intelligent systems;
- Intelligent technique components as O-O (Object-Oriented) classes [34].
The first two methods are usually used in the loose-coupling models, whereas the third one is usually used in the tight-coupling model.

Each intelligent technique has particular strengths and weaknesses and they cannot be applied universally to every problem. Furthermore, a collection of agents are needed for complex decision making. Hence integrating two or more intelligent techniques with multiple agents is very important. Thus far, there is some research work involved in this topic.

One of such attempts is the MIX multi-agent platform [35, 87]. The focus of MIX is the development of strategies and tools for integrating neural and symbolic technologies. This test-bed is a distributed system of multiple cooperating heterogeneous agents. The system includes a multi-agent toolkit with generic agent structures, services, and communication protocols. Specific agents have been developed for different types of neural networks and for other functions such as fuzzy inference and case-based reasoning.

Another such attempt is the PREDICTOR system [36]. In [36] Scherer and Schlager detail the use of distributed artificial intelligence approaches for combining neural networks and expert systems. The approach is based on a blackboard architecture and is demonstrated in the domain of economic forecasting. The authors have implemented a system called PREDICTOR using this approach to solve prediction tasks in economics. The architecture of the PREDICTOR system is shown in Figure 2.1.

In the architecture, each problem solver has specific knowledge about the domain and the ability to react to messages that are distributed via the communication facility. To handle the communication aspects of this system, each problem solver has a front end processor which is responsible for managing the co-operative aspects of the problem solver. The communication facility distributes messages all over the system (scheduler, problem solvers). The blackboard manager provides other nodes with blackboard operations like search, read, write, and update. The scheduler is responsible for the task analysis, task allocation and task synthesis. Domain dependent knowledge, specific analysis and allocation strategies are stored at a meta level within a knowledge base. This meta knowledge controls the problem solving abilities of the scheduler.
In [37], Khosla and Dillon introduce a computational architecture called IMAHDA (Intelligent Multi-Agent Hybrid Distributed Architecture). The role and knowledge content of IMAHDA consists of four layers, namely, object, software agent, intelligent agent, and the problem solving agent respectively. The IMAHDA can be seen as being constructed from generic software agents (distributed processing, distributed communication, belief base, and relational software agents), generic intelligent agents (expert/knowledge based system, supervised neural network, unsupervised neural network, fuzzy logic, genetic algorithm intelligent agents), and problem solving agents (global preprocessing, decomposition, control, decision, and post processing agents) as shown in Figure 2.2.

A more recent attempt is the multi-agent architecture for fuzzy modeling [38]. Delgado et al. proposed a hybrid learning system that combines different fuzzy modeling techniques by means of a multi-agent architecture. The proposed multi-agent architecture, involving agents which embody the different problem solving methods, is a flexible tool to be used in the fuzzy modeling process. The system consists of four kinds of agents: service agent or facilitator acting as a Yellow Page to other agents, task agents (including clustering, rule generation, tuning, and evaluator agents), resource agents, and control agents (containing planner, decisor,
and error control agents).

In [148], Jacobsen proposed a generic architecture for hybrid intelligent systems, which is based on the conceptual learning agent architecture according to Russell and Norvig [3]. They have presented two instantiations of the architecture – reinforcement-driven fuzzy-relation-adaptation architecture and an expert-guided hybrid neuro-fuzzy system and have experimentally validated their designs.

Among the above five agent-based hybrid systems, the MIX, PREDICTOR, and the architecture for fuzzy modeling only integrated very limited soft computing techniques. Both the MIX and PREDICTOR systems are focused on the integration of neural networks and symbolic technologies such as expert systems. The multi-agent architecture of Delgado et al. concentrated on the integration of different fuzzy modeling techniques such as fuzzy clustering, fuzzy rule generation, and fuzzy rule tuning techniques. In MIX and PREDICTOR systems, the way for integrating intelligent techniques into multi-agent systems is to embed the intelligent techniques in each individual agent. The MIX and IMAHDA architectures
are inflexible as no middle agent [39] was used. The work in [148] is focused on the micro (intra-agent) level of agents, i.e., the integration and interaction of different components within one agent. The macro (inter-agent) level integration and interaction are ignored.

In summarizing, the approaches used in the above systems have the following limitations:

- It is impossible to embed many intelligent techniques within a single agent. Otherwise, the agents will be overloaded. In many applications, the agents in multi-agent systems should be kept simple for ease of maintenance, initialization, and customization;

- It is not flexible to add more intelligent techniques to or delete some unwanted one from the multi-agent systems. For example, one software agent may be equipped with fuzzy logic, the other with neural network etc. In such a way, one agent can only have one soft computing capability. If one wants the agent to possess two or more soft computing or hard computing capabilities, the implementations must be modified;

- The agents in these systems are difficult to inter-operate as they did not use some type of common or standard agent communication language.

In addition to the above limitations, all these systems did not follow any agent-oriented analysis and design methodology. (In IMAHDA, object-oriented analysis and design approaches were adopted). Therefore it is essential to find a new way for integration intelligent techniques with multi-agent systems that can overcome the drawbacks of currently used approaches. Moreover, it is vital to tailor an agent-oriented methodology for agent based hybrid intelligent system construction, which is based on the currently available agent-oriented approaches.

2.4 Approaches to Agentification

There are many legacy software packages of intelligent techniques available for different applications. When constructing hybrid intelligent systems, it is of paramount
importance to reuse these software packages. On the other hand, for hybrid intelligent systems to be fully accepted in real-world applications, they must be able to integrate and communicate with conventional legacy computing systems. When constructing hybrid intelligent systems from the agent-oriented viewpoint, this implies that some techniques should be available for converting such legacy programs into agents. In work thus far, a number of different approaches have been taken [40].

One approach is to implement a transducer that mediates between an existing program and other agents. The transducer accepts messages from other agents, translates them into the program’s native communication protocol, and passes those messages to the program. It accepts the program’s responses, translates into agent communication language (ACL) such as Knowledge Query and Manipulation Language (KQML) [42, 44], and sends the resulting messages on to other agents. This approach has the advantage that it requires no knowledge of the program other than its communication behavior. It is, therefore, especially useful for situations in which the code for the program is unavailable or too delicate to modify. This approach also works for other types of resources, such as files and people. It is a simple matter to write a program to read or modify an existing file with a specialized format, thereby providing access to that file via ACL. Similarly, it is possible to provide a graphical user interface for a person, allowing one to interact with the system in a specialized graphical language, which is then converted into ACL, and vice versa.

A second approach to dealing with legacy software is to implement a wrapper; i.e., inject code into a program to allow it to communicate in ACL. The wrapper can directly examine the data structures of the program and can modify those data structures. Furthermore, it may be possible to inject calls out of the program so it can take advantage of externally available information and services. This approach has the advantage of greater efficiency than the transduction approach, since there is less serial communication. It also works for cases having no interprocess communication ability in the original program. However, it requires the code for the program be available.

Of course, the third and most drastic approach to dealing with legacy software
2.5 Multi-Agent Systems in Finance

In financial applications, multiplicity of information and different expertise must be brought together to produce a quality recommendation. This has given rise to the exploration of agent technologies for analyzing data and making critical decisions.

Although intelligent agent technology holds great promise for the financial services and investment industries, up to now, there are not many papers published or products announced in the financial field. Here are some typical multi-agent systems (by no means all) in finance:

- **The Warren System** This system is for financial portfolio management and was developed by the intelligent agent group at Carnegie Mellon University [4, 12, 16]. The Warren system consists of five main agents in the portfolio management task. They are portfolio manager agent, fundamental analysis agent, technical analysis agent, breaking news agent, and analyst tracking agent.

The portfolio manager agent is an interface agent that interacts graphically and textually with the user to acquire information about the user's profile and goals. The fundamental analysis agent is a task assistant that acquires and interprets information about a stock from the viewpoint of a stock's fundamental value. Calculating fundamental value takes into consideration information such as a company's finance, forecasts of sales, earnings, and expansion plans. The technical analysis agent uses numerical techniques such as moving averages, curve fitting, complex stochastic models, and neural nets to try to predict the near future in the stock market. The breaking news agent tracks and filters news stories and decides if they are so important that the user needs to know about them immediately, in that the stock price might be
immediately affected. The analyst tracking agent tries to gather intelligence about what human analysts are thinking about a company.

- **Banker and Investor Agent System** This intelligent agent-based portfolio management system was developed by Krishna and Ramesh at Illinois Institute of Technology [22]. It can be used by the financial service industry to provide inexpensive Internet-based “self serve” offerings to small investors. Banker agents in this system assist mutual fund managers in devising a global efficient frontier from the individual risk-return characteristics of each of the funds from which a portfolio can be constructed. Investor agents offer personalized advice to each individual investor regarding the choice of a portfolio. The various investor agents also learn from each other through “forums” hosted by the concerned financial services firm. Such a system requires minimal human intervention on both sides (the banker and the investor) thereby reducing costs without sacrificing service quality.

- **Distributed Financial Computing System** A prototype of the distributed financial computing has been implemented to illustrate the idea of multi-agent computing on workstation clusters [23]. This multi-agent system consists of 40 software agents to run on the cluster at the University of Hong Kong. This system is able to simultaneously monitor or process 30 economic indexes. It is also used to capture the structure of Hong Kong economic activities and calculate risks and returns based on more than 30 financial indicators that cover major business activities in Hong Kong.

- **Multi-Agent Financial Trading System (MAFTS)** MAFTS can help manage a financial portfolio by monitoring the variation of stock price [150]. A financial adviser agent is used in MAFTS. This agent can alert its owner, or take autonomous action when the information being monitored satisfies user-established criteria. It collaborates with other agents to retrieve financial information, performs research on it and takes appropriate action to optimize the client's portfolio.

Among the above three systems, only WARREN is a fully functional multi-agent system. It focuses on the information gathering part (not the decision making part)
of financial portfolio management. It is a working example of applying the Retsina framework [75]. The MAFTS system focuses on monitoring after portfolios have been selected.

In this thesis, financial planning problems will be presented as illustrative examples. One reason is that financial planning is a typical complex problem. Another is that multi-agent systems are lacking in the finance industry. The goal is to demonstrate how multi-agent technologies contribute toward complex problems such as financial planning, and how to enhance the complex decision making capabilities of multi-agent systems.

2.6 Summary

There are many intelligent techniques available for complex decision making, each technique has particular strengths and limitations and cannot be successfully applied to every type of problem. They are complementary and should be used in combinations for complex decision making resulting in systems called hybrid intelligent systems. Multi-agent perspectives are good at modeling hybrid intelligent systems.

However, there are two major technical impediments to the widespread adoption of agent technology [7]:

- the lack of a systematic methodology enabling designers to clearly specify and structure their applications as multi-agent systems; and

- the lack of widely available industrial-strength multi-agent system tool-kits.

In response to the first impediment, we have attempted to tailor a methodology suitable for the analysis and design of agent-based hybrid intelligent systems from the available methodologies, especially the coordination-oriented methodology. This is described in Chapter 3.

In response to the second impediment, we propose a general framework to facilitate the construction of agent-based hybrid intelligent systems. With the support of the framework (intelligent technique agent society), agent-based hybrid intelligent system developers need only to build the domain-specific parts and construct
the ontologies used in the specific application field – rather than re-inventing the wheel as often happens at the moment. This will be discussed in Chapter 4. Some problems related to the framework as well as applying it to financial investment planning are addressed in Chapters 5, 6, 7, and 8.
Chapter 3

An Agent-Oriented Methodology for Hybrid Intelligent Systems

From the discussions in previous chapters, it is clear that hybrid intelligent systems are crucial for complex decision making.

However, the developments of different hybrid intelligent systems indicate that designing and building hybrid intelligent systems is difficult. Hybrid intelligent systems are complex because they have a large number of parts or components that have many interactions (Section 3.1). Currently available methodologies for hybrid intelligent system construction cannot manage these complex interactions efficiently as these interactions may occur at unpredictable times, for unpredictable reasons, between unpredictable components (Section 3.2). For this reason it is vital to seek ways that can facilitate the modeling, designing, and creating hybrid intelligent systems.

In this chapter, it will be argued that agent-oriented approaches are well suited to engineering hybrid intelligent systems (Section 3.3). Existing software development techniques (typically, object-oriented) are inadequate for modeling agent-based hybrid intelligent systems. Extant approaches fail to adequately capture an agent's flexible, autonomous problem-solving behavior, the richness of an agent's interactions, and the complexity of an agent system's organizational structures. For these reasons, an agent-oriented methodology that is suitable for the analysis and design of agent-based hybrid intelligent systems will be extracted (Section 3.4).
3.1 Hybrids are Complex

Based on the discussion in Section 2.1, it is clear that different intelligent techniques have their own advantages and disadvantages. They cannot be applied universally to every problem, whereas many complex problems have many different component problems, each of which require different types of processing. Thus it is necessary to combine different intelligent techniques together so as to overcome the limitations of individual techniques in complex problem solving and decision making. Moreover, hybrid intelligent systems represent not only the combination of different intelligent techniques but also the integration of intelligent techniques with conventional (legacy) computing systems or programs. All these make hybrid intelligent systems complicated. Decisions need to be made about:

- Which techniques are suitable for what kinds of problems;
- The messages exchanged among different processing components (using different intelligent techniques) in the systems as they must communicate with each other, as well as legacy programs, to achieve synergism;
- Allowing the easy exchange or addition of new processing techniques;
- Knowing exactly where components reside on the network in complex problem solving and decision making, and making them work together across a heterogeneous network of computers.

In short, hybrid intelligent systems are complex because they have a large number of parts or components that have many interactions. Fortunately, such complexity exhibits a number of important regularities [88]:

- Complexity frequently takes the form of a hierarchy. That is, the system is composed of inter-related subsystems, each of which is in turn hierarchical in structure. The precise nature of these organizational relationships varies between subsystems, however some generic forms (such as client-server, peer, team, etc.) can be identified. These relationships are not static; they often vary over time.
CHAPTER 3. AN AGENT-ORIENTED METHODOLOGY FOR HYBRIDS

- The choice of which components in the system are primitive is relatively arbitrary and is defined by the observer’s aims and objectives.

- Hierarchical systems evolve more quickly than non-hierarchical ones of comparable size. In other words, complex systems will evolve from simple systems more rapidly if there are stable intermediate forms, than if there are not.

- It is possible to distinguish between the interactions among subsystems and the interactions within subsystems. The latter are both more frequent (typically at least an order of magnitude more) and more predictable than the former. This gives rise to the view that complex systems are nearly decomposable: subsystems can be treated almost as if they are independent of one another, but not quite since there are some interactions between them. Moreover, although many of these interactions can be predicted at design time, some cannot.

Drawing these insights together, it is possible to define a canonical view of a complex system (recall Figure 1.1).

Given these observations, software engineers have devised a number of powerful tools in order to tackle this complexity. The principal mechanisms include [89]:

- **Decomposition**: The most basic technique for tackling any large, complex problem is to divide it into smaller, more manageable chunks, each of which can then be dealt with in relative isolation. Decomposition helps tackle complexity because it limits the designer’s scope.

- **Abstraction**: Abstraction is the process of defining a simplified model of the system that emphasizes some of the details or properties, while suppressing others. Again, this works because it limits the designer’s scope of interest at a given time.

- **Organization**: Organization is the process of identifying and managing interrelationships between various problem solving components. The ability to specify and enact organizational relationships helps designers tackle complexity in two ways: (1) by enabling a number of basic components to be grouped
together and treated as a higher-level unit of analysis; and (2) by providing a means of describing the high-level relationships between various units.

Any approach to building hybrid intelligent systems should support these three mechanisms – decomposition, abstraction, and organization.

3.2 Current Practice in Typical Hybrid Intelligent System Development

In current practice, most hybrid intelligent systems can be classified into three classes: function-replacing, inter-communicating, and polymorphic [68].

Function-replacing hybrids address the functional composition of a single intelligent technique. In this hybrid class, a principal function of the given technique is replaced by another intelligent processing technique. The motivation for these hybrid systems is the technique enhancement factor discussed above.

Inter-communicating hybrids are independent, self-contained, intelligent processing modules that exchange information and perform separate functions to generate solutions. If a problem can be subdivided into distinct processing tasks, then different independent intelligent modules can be used to solve the parts of the problem at which they are best. These independent modules which collectively solve the given task are co-ordinated by a control mechanism.

Polymorphic hybrids are systems that use a single processing architecture to achieve the functionality of different intelligent processing techniques. The broad motivation for these hybrid systems is realizing multi-functionality within particular computational architectures. These systems can functionally mimic or emulate different processing techniques.

Many hybrid intelligent systems used in different application fields appeared in the past ten years [37, 68, 69, 70, 71, 72]. All these systems fall into the three classes. A typical development cycle in the implementation of these hybrid intelligent systems is shown in Figure 3.1, which is based on object-oriented techniques. There are six stages in the construction of hybrid intelligent systems: problem analysis, property matching, hybrid category selection, implementation, validation, and
maintenance [68]. Most current hybrid intelligent systems are built either from scratch or following this development process.

![Diagram of Hybrid Intelligent System Development Cycle]

Figure 3.1: Hybrid Intelligent System Development Cycle

There are some shortcomings of the hybrid intelligent systems by following this development process. The most obvious one is that the organization of such a hybrid system is not adaptive. Once the techniques are selected in the property matching stage, it is difficult to change or replace it even though one may find a better one later on.

Another difficulty lies in the hybrid category selection phase. At hybrid category selection phase, the developers must choose the type of hybrid system required (function-replacing, inter-communicating, or polymorphic) for solving the specific problem. This is not an easy job to do. The hybrid intelligent systems' inherent complexity means it is impossible to know a priori about all potential links or relationships among components consisting of a system; interactions will occur at unpredictable times, for unpredictable reasons, between unpredictable components. For this reason, it is futile to try and predict or analyze all the possibilities at design-
time.

Is there any better way to construct hybrid intelligent systems? The answer is "yes". Agent-oriented approaches are very promising for building hybrid intelligent systems.

3.3 Agent-Oriented Approaches Suitable for Hybrids

In Section 1.3, it was briefly described that the multi-agent perspective is suitable for hybrid intelligent systems. In this section, it will be shown that: (i) agent-oriented decompositions are an effective way of partitioning the problem space of a hybrid intelligent system; (ii) the key abstractions of the agent-oriented mind-set are a natural means of modeling hybrid intelligent systems; and (iii) the agent-oriented philosophy for dealing with organizational relationships is appropriate for hybrid intelligent systems. Other advantages adopting agent-oriented approaches to constructing hybrid intelligent systems will also be discussed. The comparison of object-oriented and agent-oriented approaches to hybrid intelligent systems are discussed.

3.3.1 Agent-Oriented Decomposition, Abstraction, and Organization

From the canonical view of a multi-agent system (recall Figure 1.2), it can be seen that adopting an agent-oriented approach to software engineering means decomposing the problem into multiple, interacting, autonomous components (agents) that have particular objectives to achieve. The key abstraction models that define the "agent-oriented mind-set" are agents, interactions and organizations. Finally, explicit structures and mechanisms are often available for describing and managing the complex and changing web of organizational relationships that exist between the agents.

Some researchers in this field gave a qualitative analysis to provide the intellectual justification of precisely why agent-based systems are well suited to engineering
complex software systems [2, 73, 74, 76, 77]. They also provided a detailed analysis of the merits of agent-oriented decomposition, the appropriateness of agent-oriented abstractions, and the need for flexible management of changing organizational structures in the process of building complex software systems. From the description in Section 3.1, it is evident that hybrid intelligent systems are complex software systems and bear all the natures of other industrial-strength software systems. Thus, agent-oriented approaches can significantly enhance our ability to model, design and build hybrid intelligent systems for the following reasons:

- **The merits of agent-oriented decomposition.** Hybrid intelligent systems consist of a number of related subsystems organized in a hierarchical fashion. At any given level, subsystems work together to achieve the functionality of their parent system. Moreover, within a subsystem, the constituent components work together to deliver the overall functionality. Thus, the same basic model of interacting components, working together to achieve particular objectives occurs throughout the system. The agent-oriented approach advocates decomposing problems in terms of autonomous agents that can engage in flexible, high-level interactions. The fact that agents are active means they know for themselves when they should be acting and when they should update their state (cf. passive objects that need to be invoked by some external entity to do either). Such self-awareness reduces control complexity since the system's control know-how is taken from a centralized repository and localized inside each individual problem solving component. The fact that agents make decisions about the nature and scope of interactions at run-time makes the engineering of hybrid intelligent systems easier for two main reasons. Firstly, the system's inherent complexity means it is impossible to know a priori about all potential links: interactions will occur at unpredictable times, for unpredictable reasons, between unpredictable components. For this reason, it is futile to try and predict or analyze all the possibilities at design-time. Rather, it is more realistic to endow the components with the ability to make decisions about the nature and scope of their interactions at run-time. Thus agents are specifically designed to deal with unanticipated
requests and they can spontaneously generate requests for assistance whenever appropriate. Secondly, the problem of managing control relationships between the software components is significantly reduced.

- *The suitability of the agent-oriented abstractions.* A significant part of the design process is finding the right models for viewing the problem. In general, there will be multiple candidates and the difficult task is to pick out the most appropriate one. When designing software, the most powerful abstractions are those that minimize the semantic gap between the units of analysis that are intuitively used to conceptualize the problem and the constructs present in the solution paradigm. In the case of complex hybrid intelligent systems, the problem to be characterized consists of subsystems, subsystem components, interactions and organizational relationships. Taking each in turn: Subsystems naturally correspond to agent organizations; The appropriateness of viewing subsystem components as agents has been made above; The interplay between the subsystems and between their constituent components is most naturally viewed in terms of high-level social interactions; Complex hybrid intelligent systems involve changing webs of relationships between their various components. They also require collections of components to be treated as a single conceptual unit when viewed from a different level of abstraction. Here again the agent-oriented mind-set provides suitable abstractions.

- *The need for flexible management of changing organizational structures.* Organizational constructs are first-class entities in agent systems. Thus explicit representations are made of organizational relationships and structures. Moreover, agent-based systems have the concomitant computational mechanisms for flexibly forming, maintaining and disbanding organizations. This representational power enables agent-oriented systems to exploit two facets of the nature of complex hybrid intelligent systems. Firstly, the notion of a primitive component can be varied according to the needs of the observer. Thus at one level, entire subsystems can be viewed as a singleton, a collection of agents can be viewed as primitive components, and so on until the system
eventually bottoms out. Secondly, such structures provide a variety of stable intermediate forms. These forms are essential for rapid development of complex hybrid intelligent systems. Their availability means that individual agents or organizational groupings can be developed in relative isolation and then added into the system in an incremental manner. This, in turn, ensures there is a smooth growth in functionality.

3.3.2 The Advantages of Using Middle Agents

When modeling hybrid intelligent systems from an agent-oriented perspective, a special sort of agents, called middle agents, are usually introduced into the framework. Middle agents receive the advertisements of agent capabilities and store these advertisements in an internal database. When an agent (a requester agent) would like to find a service provider agent that possesses certain desired capabilities, it sends a request to a middle agent. The middle agent matches the request to its database of received advertisements to determine whether an agent whose capabilities match the request is known. There are different types of middle agents such as matchmakers, brokers, and facilitators [39] etc.

The presence of the middle agents in an agent-based hybrid intelligent system framework allows adaptive organization. The high level goals and tasks imparted by the user form the context within which agents can adaptively (with the help of middle agents) form teams or coalitions so that this collaboration will fulfill the goals and tasks. Overall system robustness is also facilitated through the use of middle agents. If a particular service provider disappears, a requester agent can find another one with the same or similar capabilities by interrogating appropriate middle agents (refer to Chapter 4).

In a word, an agent-oriented perspective is well-suited to model complex hybrid intelligent systems. Adopting agent-oriented approaches to constructing hybrid intelligent systems also leads to more adaptive system organizations, and more enhanced communication ability among system components as well as conventional computing systems.
3.3.3 Object-Oriented Vs Agent-Oriented

An object is a logical combination of data structures and their corresponding methods (functions). Objects are successfully being used as abstractions for passive entities (e.g., a house) in the real-world, and agents are regarded as a possible successor of objects since they can improve the abstractions of active entities. From the object-oriented point of view in constructing hybrid intelligent systems, different intelligent techniques are represented as objects and integrated with other intelligent technique components via a well-defined message protocol. From the agent-oriented point of view, different intelligent techniques are represented as autonomous agents and integrated with other intelligent technique agents through common or standard agent communication languages.

There are a number of similarities between the object- and agent-oriented views of system development. For example, both emphasize the importance of interactions between entities. However, there are also a number of important differences [7, 8, 73].

The first difference is in the degree to which agents and objects are autonomous. Objects can have control over their own internal state, but fail to exhibit control over their behaviors. That is, if a method \( m \) is made available for other objects to invoke, then they can do so whenever they wish; the object has no control over whether or not that method is executed. The distinction between objects and agents can be summarized in the following slogan: “Objects do it for free; agents do it for money.”

Note that there is nothing to stop one implementing agents using object-oriented techniques. For example, one can build some kind of decision making about whether to execute a method into the method itself, and in this way achieve a stronger kind of autonomy for the objects. However, the point is that this kind of autonomy is not a component of the basic object-oriented model.

The second important distinction between object and agent systems is with respect to the notion of flexible (reactive, pro-active, social) autonomous behaviors. The standard object model has nothing whatsoever to say about how to build systems that integrate these types of behavior. Again, one could argue that we can build object-oriented programs that do integrate these types of behavior. But this
argument misses the point, which is that the standard object-oriented programming model has nothing to do with these types of behavior.

The third important distinction between the standard object model and our view of agent systems is that agents are each considered to have their own thread of control—in the standard object model, there is a single thread of control in the system.

In addition to the above mentioned distinctions, there are two points that qualitatively differentiate agent interactions from those that occur in other software engineering paradigms such as object-oriented paradigm. Firstly, agent-oriented interactions generally occur through a high-level agent communication language. Consequently, interactions are usually conducted at the knowledge level; in terms of which goals should be followed, at what time, and by whom. Secondly, as agents are flexible problem solvers, operating in an environment over which they have only partial control and observability, interactions need to be handled in a similarly flexible manner. Thus, agents need the computational apparatus to make context-dependent decisions about the nature and scope of their interactions and to initiate (and respond to) interactions that were not necessarily foreseen at design time.

To summarize, the traditional view of an object and our view of an agent have at least three distinctions:

- Agents embody a stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agents;

- Agents are capable of flexible (reactive, pro-active, social) behavior, and the standard object model has nothing to say about such types of behavior; and

- A multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of control.
3.4 Methodology for Analysis and Design Agent-Based Hybrids

From the discussions so far, it is clear that hybrid intelligent systems are crucial for complex problem solving and decision making. It is also apparent that agent abstraction can be used by software developers to more naturally understand, model, and develop hybrid intelligent systems. On the other hand, agent-based hybrid intelligent systems have the following main characteristics:

- Agents are heterogeneous, in that different agents may be implemented using different programming languages, architectures, and techniques;
- The organizational structure of the system is dynamic, in which agents can dynamically leave and enter the system;
- Agents exhibit social behavior, in that they interact with one another to cooperate to achieve a common objective;
- There are no self-interested agents in the systems; and
- Integration and interaction of different techniques is crucial.

Existing software development techniques (for example, object-oriented analysis and design) are inadequate for this task. There is a fundamental mismatch between the concepts used by object-oriented developers (and indeed, by other mainstream software engineering paradigms) and the agent-oriented perspective (see Section 2.2 and Section 3.3.3). In particular, extant approaches fail to adequately capture an agent’s flexible, autonomous problem-solving behavior, the richness of an agent’s interactions, and the complexity of an agent system’s organizational structure. For these reasons, this section outlines a methodology that has been specifically tailored to the analysis and design of agent-based hybrid intelligent systems.

3.4.1 Outline of the Methodology

In our problem domain, the dynamic arrival of unknown agents needs to be taken into account, but with no self-interested behavior in the course of the interactions.
The Gaia methodology is ill suited to handling the dynamic arrival of new agents into the system, whereas coordination-oriented methodology is focused on the processing of self-interested agents. Thus both methodologies cannot be applied to our problem directly.

In Gaia methodology, analysis and design are well-separated phases. The analysis phase aims to identify what the actual organization of the multiple agents should look like. It does this by decomposing the system into abstract "loci of control", i.e., the roles to be played in the organization, and the way in which they interact accordingly to specific protocols (this, respectively, defines the role model and interaction model). The design phase starts from the models defined during the analysis phase and aims to define the actual agent system in such a way that it can easily be implemented. To this end, the design phase has to decide which classes of agents and how many have to play the roles identified during the analysis phase (agent model); which services agents must provide to fulfill their role (service model); and what is the actual topology of the interactions that flows from interaction and the agent models (acquaintance model).

The concept of coordination models can be exploited in the context of designing multi-agent systems for use on the Internet, where openness and self-interest are key factors. A coordination model can be thought of as consisting of three elements [78]:

- the *coordinables*: the entities whose mutual interaction is ruled by the model, e.g., the agents in a multi-agent system.

- the *coordination media*: the abstractions enabling agent interactions, as well as the core around which the components of a coordinated system are organized. Examples are semaphores, monitors, channels, or more complex media like tuple spaces, blackboards, etc.

- the *coordination laws*: define the behavior of the coordination media in response to interaction events. The laws can be defined in terms of a communication language (a syntax used to express and exchange data structures) and a coordination language (a set of interaction primitives and their semantics).
The architecture of a multi-agent system can naturally be viewed as a computational organization. For the complete specification of computational organizations, three additional organizational concepts – organizational rules, organizational structures, and organizational patterns are introduced [91]. Zambonelli, Jennings, and Wooldridge [91] argued that these concepts are of fundamental importance in multi-agent systems, and they should play a central role in any methodology.

Furthermore, in complex problem solving and decision making, the rapport of information, knowledge, and skills to use the information and knowledge is of paramount importance. When working on a methodology for such agent-based systems, one should pay attention to the knowledge and skills of agents.

Taking all these factors into account, a methodology for the analysis and design of agent-based hybrid intelligent systems based on the Gaia methodology, coordination model, and organization abstraction is created.

We propose that such a methodology should consist of the construction of six main models: agent models, role models, skill models, knowledge models, an organizational model, and an interaction model. Each of the steps in the methodology below results in the construction of one (or more) of the corresponding models. The methodology for their elaboration and refinement can be expressed in five steps:

- The first step consists of the identification of the roles in the application domain in terms of the corresponding goals or tasks. Consequently, a role is identified by the main tasks or responsibilities required by the system and forms the basis for a specification of agent types;

- Secondly, the main skills associated with agent roles should be identified. Skills consist of the basic services required to be able to perform a role and the ways to manipulate knowledge (reasoning techniques). The issue of agent-wrapping should be addressed in this step, which involves the integration of non-agent software components, and the design of appropriate interface;

- The third step consists of the modeling of the knowledge about the application domain associated with identified roles or skills and should result in the design
of an adequate ontology. Techniques from knowledge engineering can be used here;

- Fourthly, an organizational structure of the multi-agent system should be designed. The coordination among agents to perform a task and the required communication schemes need to be analyzed in this step; and

- Finally, the dynamics of the multi-agent system should be analyzed in terms of the flow of information, resulting in, for example, synchronization requirements derived from the roles associated with individual agents.

With this methodology, analysis and design can be thought of as a process of developing increasingly detailed models of system to be constructed. The main models used in this methodology are summarized in Figure 3.2. More details are given in the following subsections. The descriptions of role models, interaction models, and agent models are based on [84]. The description of organizational model (organizational rules, organizational structure, and organizational patterns) is based on [91].

### 3.4.2 The Role Model

The role model identifies the key roles in the system. Here a role can be viewed as an abstract description of an entity’s expected function. A role is defined by four attributes: responsibilities, permissions, activities, and protocols.
Figure 3.3: Template for Role Schemata.

Responsibilities determine functionality and, as such, are perhaps the key attribute associated with a role. Responsibilities are divided into two types: liveness properties and safety properties. Liveness properties intuitively state that "something good happens". They describe those states of affairs that an agent must bring about, given certain environmental conditions. In contrast, safety properties are invariants. Intuitively, a safety property states that "nothing bad happens" (i.e., that an acceptable state of affairs is maintained across all states of execution).

In order to realize responsibilities, a role has a set of permissions. Permissions are the "rights" associated with a role. The permissions of a role thus identify the resources that are available to that role in order to realize its responsibilities. Permissions tend to be information resources. For example, a role might have associated with it the ability to read a particular item of information, or to modify another piece of information. A role can also have the ability to generate information. The activities of a role are computations associated with the role that may be carried out by the agent without interacting with other agents.

Finally, a role is also identified with a number of protocols, which define the way that it can interact with other roles. A role model is comprised of a set of role schemata, one for each role in the system. A role schema draws together the various attributes discussed above into a single place (see Figure 3.3).

The formal notation for expressing protocols, activities, permissions, and responsibilities adopted by Gaia will be used. To introduce these concepts, the example of a PriceWatcher role will be used. The purpose of this role is to
monitor whether the trading price of a specific security is exceeded the expected value of the shareholder. The protocols and activities involved in the PRICEWATCHER role include: InformShareholder, GetInitialInformation, GetPrice, and Compare. The activity names (like Compare) are underlined to distinguish them from protocols.

The following is an illustration of the permissions associated with the role PRICEWATCHER:

reads supplied SecurityCode // Security code used in Share Exchanger

  supplied ExpectedValue // The value the shareholder expected

  supplied TradingPrice // The current trading price of the security

This specification defines three permissions for PRICEWATCHER: it says that the agent carrying out the role has permissions to access the value of SecurityCode, ExpectedValue, and TradingPrice. The supplied keyword here is used to indicate that some roles are parameterized by certain values. Another two types of permissions are changes (read and modify) and generates (produce a resource). Note that these permissions relate to the knowledge that the agent has.

The liveness responsibilities for the PRICEWATCHER role might be:

- whenever the share exchange is not closed, get the trading price of the specific security (indicated by the SecurityCode);

- whenever the trading price is exceeded the expected value, inform the shareholder.

Following the Gaia notation, liveness properties are specified via a liveness expression, which defines the "life-cycle" of the role and is a regular expression. The general form of a liveness expression is:

\[ \text{ROLENAME} = \text{expression} \]

where ROLENAME is the name of the role whose liveness properties are being defined, and expression is the liveness expression defining the liveness properties of ROLENAME. The atomic components of a liveness expression are either activities or protocols. The operators for liveness expressions are shown in Table 3.1.
Table 3.1: Operators for Liveness Expressions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x . y$</td>
<td>$x$ followed by $y$</td>
</tr>
<tr>
<td>$x</td>
<td>y$</td>
</tr>
<tr>
<td>$x^*$</td>
<td>$x$ occurs 0 or more times</td>
</tr>
<tr>
<td>$x+$</td>
<td>$x$ occurs 1 or more times</td>
</tr>
<tr>
<td>$x^\omega$</td>
<td>$x$ occurs infinitely often</td>
</tr>
<tr>
<td>$[x]$</td>
<td>$x$ is optional</td>
</tr>
<tr>
<td>$x</td>
<td>y$</td>
</tr>
</tbody>
</table>

Thus the liveness responsibilities of the \texttt{PriceWatcher} role can be expressed as:

\[
\texttt{PriceWatcher} = (\texttt{GetInitializeInformation})+. (\texttt{GetPrice, Compare})+. (\texttt{InformShareholder})*. 
\]

This expression says that \texttt{PriceWatcher} consists of executing the protocol \texttt{GetInitializeInformation}, followed by the protocol \texttt{GetPrice}, followed by the activity \texttt{Compare} and the protocol \texttt{InformShareholder}.

Safety requirements are specified by means of a list of predicates. These predicates are typically expressed over the variables listed in a role's permission attribute. By convention, safety expressions are listed as a bulleted list, each item in the list expressing an individual safety responsibility.

When all these are put together, the schema for the \texttt{PriceWatcher} role results (Figure 3.4).

### 3.4.3 The Interaction Model

There are inevitably dependencies and relationships between the various roles in a multi-agent organization. Indeed, such interplay is central to the way in which the system functions. Given this fact, interactions obviously need to be captured and represented in the analysis phase. Such links between roles are represented in the interaction model. This model consists of a set of protocol definitions, one for each
CHAPTER 3. AN AGENT-ORIENTED METHODOLOGY FOR HYBRIDS 57

<table>
<thead>
<tr>
<th>Role Schema: PriceWatcher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
</tr>
<tr>
<td>This role involves monitoring whether the trading price of a specific security is exceeded the expected value of the shareholder.</td>
</tr>
<tr>
<td>Protocols and Activities:</td>
</tr>
<tr>
<td>InformShareholder, GetInitializelnformation, GetPrice, Compare</td>
</tr>
<tr>
<td>Permissions:</td>
</tr>
<tr>
<td>reads supplied SecurityCode // Security code used in Share Exchanger</td>
</tr>
<tr>
<td>supplied ExpectedValue // The value the shareholder expected</td>
</tr>
<tr>
<td>supplied TradingPrice // The current trading price of the security</td>
</tr>
<tr>
<td>Responsibilities</td>
</tr>
<tr>
<td>Liveness:</td>
</tr>
<tr>
<td>PriceWatcher =</td>
</tr>
<tr>
<td>(GetInitializelnformation) + (GetPrice, Compare) + (InformShareholder)</td>
</tr>
<tr>
<td>Safety:</td>
</tr>
<tr>
<td>• True</td>
</tr>
</tbody>
</table>

Figure 3.4: Schema for Role PriceWatcher

type of inter-role interaction. Here a protocol can be viewed as an institutionalized pattern of interaction.

A protocol definition consists of the following attributes:

- **purpose**: brief textual description of the nature of the interaction (e.g., “information request”, “schedule activity”, and “assign task”);

- **initiator**: the role(s) responsible for starting the interaction;

- **responder**: the role(s) with which the initiator interacts;

- **inputs**: information used by the role initiator while enacting the protocol;

- **outputs**: information supplied by/to the protocol responder during the course of the interaction;

- **processing**: brief textual description of any processing the protocol initiator performs during the course of the interaction.
Figure 3.5: The GetPrice Protocol Definition

As an illustration, the GetPrice protocol is considered, which forms part of the PriceWatcher role (Figure 3.5). This states that the protocol GetPrice is initiated by the role PriceWatcher and involves the role ShareExchanger. This protocol involves PriceWatcher providing ShareExchanger with the SecurityCode, and results in ShareExchanger returning the value of the TradingPrice for security designated by the SecurityCode.

3.4.4 Organizational Rules

Role models precisely describe all the roles that constitute the computational organization; in terms of their functionalities, activities, and responsibilities, as well as in terms of their interaction protocols and patterns, which establish the position of each role in the organization. However, such role models cannot be considered as the sole organizational abstraction upon which to base the entire development process. Rather, before the design process actually defines the role model and, consequently, the whole organization, the analysis phase should define how the organization is expected to work, i.e., the organizational rules. These describe the constraints that the actual organization, once defined, will have to respect.

The explicit identification of organizational rules is of particular importance in the context of open agent systems. With the arrival of new and previously unknown agents, the overall organization must somehow enforce its internal coherency despite the dynamic and untrustworthy environment. The identification
of global organizational rules allows the hybrid system designer to explicitly define whether and when to allow newly arrived agents to enter the organization, and once accepted, what their position in the organization should be.

In summary, the analysis phase is tasked with collecting all the specifications from which the design of the computational organization can start (refer to Figure 3.2). The output of the analysis phase should be a triple: \((PR, PP, OL)\), where \(PR\) are the preliminary roles of the system, \(PP\) are the preliminary protocols (which have already been discovered to be necessary for the preliminary roles), and \(OL\) are the organizational rules.

3.4.5 The Agent Model

The purpose of the agent model is to document the various agent types that will be used in the system under development, and the agent instances that will realize these agent types at run-time.

An agent type is best thought of as a set of agent roles. There may, in fact, be a one-to-one correspondence between roles (as identified in the role model) and agent types. However, this need not be the case. A designer can choose to package a number of closely related roles in the same agent type for the purpose of convenience. Efficiency will also be a major concern at this stage – a designer will almost certainly want to optimize the design, and one way of doing this is to aggregate a number of agent roles into a single type.

The agent model is defined using a simple agent type tree, in which leaf nodes correspond to roles, and other nodes correspond to agent types. If an agent type \(t_1\) has children \(t_2\) and \(t_3\), then this means that \(t_1\) is composed of the roles that make up \(t_2\) and \(t_3\).

3.4.6 The Skill Model

The aim of the skill model is to identify the main skills with each agent role. Skills mainly consist of the basic services required to be able to perform a role.

A service is defined as a function of the agent. For each service that may be performed by an agent, it is necessary to document its properties. Specifically,
one must identify the inputs, outputs, pre-conditions, and post-conditions of each service. Inputs and outputs to services will be derived in an obvious way from the interaction model. Pre- and post-conditions represent constraints on services. These are derived from the safety properties of a role. Note that by definition, each role will be associated with at least one service.

The services that an agent will perform are derived from the list of protocols, activities, responsibilities and the liveness properties of a role. The inference mechanisms used by roles are also needed to be identified in this model.

3.4.7 The Knowledge Model

The knowledge model is to identify the different knowledge levels needed by each identified agent in agent model. The first level of knowledge is for agent interaction and communication. This involves in domain-specific and domain-independent terminologies and their relationships etc. The identified domain-specific terms and their relationships will result in the construction of a domain-dependent ontology for a specific application. The identified domain-independent terms and their relationships will result in customizing a domain-independent ontology from some available general-purpose ontologies.

The second level of knowledge is some domain knowledge related to specific problem solving techniques. This part of knowledge can be represented by typical if – then rules. These rules are also domain-specific.

The third level of knowledge is meta knowledge that directs the activities of an agent. This part of knowledge can also be represented by if – then rules. These rules are more abstract than those in the second level.

3.4.8 Organizational Structures and Patterns

In the design of a multi-agent system, as well as in the design of any organization, the role model should derive from the organizational structure that is explicitly chosen. Thus organizational structures should be viewed as first-class abstractions in the design of multi-agent systems.

The definition of the system’s overall organizational structure can derive from
the specifications collected during the analysis phase, as well as from other factors, related to efficiency, simplicity of application design, and organizational theory. In any case, a methodology cannot start the analysis phase by attempting to define a complete role model that implicitly sets the organizational structure. Rather, the definition of the organizational structure is a design choice that should not be anticipated during the analysis phase.

The obvious means by which to specify an organization is by the inter-agent relationships that exist within it. There is no universally accepted terminology set of organizational relationships: different types of organizations make use of entirely different organizational concepts.

The aim of organizational patterns is to encourage re-use of pre-defined components and architectures in order to ease and speed-up the work of both designers and developers. With the availability of catalogs of organizational patterns, designers can recognize in their multi-agent systems the presence of known patterns, and re-use definitions from the catalog. In addition, designers can also be guided by the catalog in the choice of the most appropriate organizational patterns for their multi-agent system. Of course, for patterns to be properly exploited, the organizational structure must have been explicitly identified in the design phase.

The design phase builds on the output of the analysis phase and produces a complete specification of the multi-agent system. The design stage can now be summarized as the following:

- Create an agent model: (1) aggregate roles into agent types, and refine to form an agent type hierarchy; (2) document the instances of each agent type using instance annotations.

- Develop a skill model, by examining activities, protocols, and safety and liveness properties of roles.

- Develop a knowledge model from the interaction model and agent model.

- Identify organizational structures and organizational patterns that respect the organizational rules.
3.5 Summary and Discussion

Taking into account the characteristics of hybrid intelligent systems, a new methodology for the analysis and design of agent-based hybrid intelligent systems was proposed and sketched, which is based on the Gaia methodology and organizational abstractions. These guidelines allow one to start the analysis and design of an agent-based hybrid intelligent system for complex decision making. This methodology differs from other agent-oriented methodologies in its skill and knowledge models. However, further work is needed to detail the proposed methodology, by:

- fully formalizing the concepts of organizational rules and organizational structures;
- fully formalizing the concepts of knowledge model;
- providing suitable notations for expressing the expected outputs of the analysis and design phases.
Chapter 4

An Agent-Based Intelligent Technique Society for Decision Making

In the previous chapters, it was argued that:

- Complex decision making needs hybrid intelligent systems employing some combination of different intelligent techniques;

- A collection of agents (multi-agent systems) possess the characteristics for complex problems. They are good at multi-expertise, uncertain, dynamic, distributed, and heterogeneous problems.

Hence in order to make good decisions, agent-based hybrid intelligent systems are required. How to incorporate intelligent techniques into multi-agent systems (decision making agents) efficiently and effectively is of paramount importance. An agent-based intelligent technique society to accomplish the integration task is now proposed.

It is now widely recognized that interaction is probably the most important single characteristic of complex software, whereas the integration and interaction of different techniques is crucial for the construction of hybrid intelligent systems. Thus hybrid intelligent systems are complex and difficult to build.
As existing software development techniques (for example, object-oriented analysis and design) are inadequate for hybrid intelligent system construction from the agent viewpoint, a methodology for the analysis and design of agent-based hybrid intelligent systems was tailored (see Chapter 3). In this chapter, the methodology will be followed to deal with the analysis, design, and implementation of an agent-based hybrid framework for complex decision making – an agent-based intelligent technique society – starting from requirement statements. A case study will be also provided to verify the society.

Compared with those agent-based hybrid systems described in Section 2.3, the framework has some crucial characteristics that differentiate this work from other hybrid intelligent systems:

- The ability to exchange comprehensible communications (interactions at knowledge level);

- Each service requester agent (decision making agent) can easily access all the intelligent techniques provided by service provider agents (e.g., soft computing agents) in the system;

- The presence of the serving agent in the framework allows adaptive system organization. For example, if one is sure that one hybrid soft computing agent can do a better job than a single technology soft computing agent, one can delete the single soft computing technology agent and add the hybrid agent to the society simply by adding or deleting a record in the database of the serving agent;

- Overall system robustness is facilitated through the use of the serving agent. For example, if a particular service provider (e.g., a soft computing agent) disappears, a requester agent (decision making agent) can find another one with the same or similar capabilities by interrogating the serving agent; and

- Agent-based hybrid intelligent systems based on the framework can make decisions about the nature and scope of interactions at run time.

With the support of the agent-based intelligent technique society, the agent-based hybrid intelligent system developers need only to build the domain-specific
parts and construct the ontologies used in the specific application field – rather than re-inventing the wheel as often happens at the moment.

4.1 Requirement Statements

A large amount of our daily work is making decisions. There is an increasing demand for hybrid intelligent systems in complex decision making. Currently hybrid intelligent system construction methods have many limitations (see Sections 2.3 and 3.2). It is vital to seek new frameworks to facilitate the construction of hybrid intelligent systems for complex decision making. Such frameworks must meet the following requirements:

- Any new intelligent technique components or their combinations can be added to the system dynamically, or some out-of-date technique components can be deleted from the system dynamically;

- Any decision-making component in the system can access any of the intelligent technique components available in the system; any intelligent technique component is also allowed to access any other intelligent technique components;

- If one specific intelligent technique component disappears, decision-making components can find other intelligent technique component with the same or similar capabilities;

- Different decisions can be fused or aggregated when necessary.

In order to identify which components should be contained in a typical decision-making system, without loss of generality, consider a financial house providing investment advice for clients. In such a house, there are: a front counter or reception desk clerk, one or more personnel officer(s), and many financial investment experts (decision makers). The advice giving (decision making) process is initiated by a user contacting the front desk clerk with a set of requirements. The clerk asks the personnel officer to provide the experts’ profile, and then delegates the task to one or more experts based on experts’ profiles. The experts then work on the task
and try to give their recommendations with or without external help. After the experts finish preparing a recommendation (if the task was assigned to more than one expert, the recommendations from different experts must be combined to form a final one), they pass it to the front desk clerk. Finally, the clerk sends the advice to the user. Such a typical process can help us analyze and design a multi-agent system for decision making.

4.2 Analysis of Agent-Based Intelligent Technique Society

Based on the requirement statements in the previous section and the methodology proposed in the last chapter, it is comparatively straightforward to identify the roles in the hybrid framework for complex decision making.

The front desk clerk's behavior falls into two distinct roles: one acting as an interface to the user (USERHANDLER, Figure 4.1) and one overseeing the process inside the organization (WORKPLANNER, Figure 4.2). The personnel officer's behavior falls into another two roles: one keeping track of the profiles (CAPABILITYRECORD, Figure 4.3) and one checking the profiles (CAPABILITYMATCHER, Figure 4.4). The experts' behaviors are covered by DECISIONMAKER (Figure 4.5), HELP PROVIDER (Figure 4.6), and DECISIONAGGREGATOR (Figure 4.7) roles. The final role is that of the USER (Figure 4.8) who requires the decision.

With the respective role definitions in place, the next stage is to define the associated interaction models for these roles. Here we focus on the interactions associated with the DECISIONMAKER role.

This role interacts with the WORKPLANNER role to obtain the task this role will accomplish (ReceiveTask protocol, Figure 4.9a). It interacts with the INFOGATHER role (not included in the decision making module, refer to Figure 1.3) to get some relevant information (known facts) for the task (GetInformation protocol, Figure 4.9b). It also interacts with the CAPABILITYMATCHER role to provide some roles for data pre- and/or post-processing etc. when accomplishing the task (AskforHelp protocol, Figure 4.9c). When the DECISIONMAKER role finishes making decision for the task, it informs the DECISIONAGGREGATOR role its alternative decision for
### Role Schema: USERHANDLER

**Description:**
Receives request/inquiry from the user and oversees process to ensure appropriate decision is returned.

**Protocols and Activities:**
- AwaitCall
- InformUser
- ProduceAssignments
- ReceiveDecision

**Permissions:**
- **reads** supplied UserDetails // personal information of the user
- supplied UserRequirements // what user wants
- Decision // final decision or nil

**Responsibilities**

**Liveness:**
USERHANDLER =
- AwaitCall, ProduceAssignments, ReceiveDecision, InformUser

**Safety:**
- True

---

Figure 4.1: Schema for Role USERHANDLER

### Role Schema: WORKPLANNER

**Description:**
This role elaborates a work plan for decision making and is in charge of ensuring that such a work plan is fulfilled.

**Protocols and Activities:**
- GetUserRequirements
- ProducePlan
- GetCapabilities
- DelegateTasks

**Permissions:**
- **reads** supplied UserDetails // personal information of the user
- supplied UserRequirements // detailed service requirements
- Capabilities // capabilities of other roles
- generates plan // work plan

**Responsibilities**

**Liveness:**
WORKPLANNER =
- GetUserRequirements, GetCapabilities, ProducePlan, DelegateTasks

**Safety:**
- UserDetails and UserRequirements are available

---

Figure 4.2: Schema for Role WORKPLANNER
### Role Schema: CapabilityRecorder

**Description:**
Add capabilities advertised by roles to CapabilityDatabase, or delete capabilities unadvertised by roles from the database.

**Protocols and Activities:**
ReceiveAdvertisement, AddCapability, DeleteCapability

**Permissions:**
- **Reads supplied** CapabilityInfo
  // Advertised or unadvertised capability information
- **Changes** CapabilityDatabase // add or delete capability

**Responsibilities**
- **Liveness:**
  - CAPABILITYRECORDER = ReceiveAdvertisement.(AddCapability|DeleteCapability)

**Safety:**
- CapabilityDatabase exists

---

### Role Schema: CapabilityMatcher

**Description:**
Matches capabilities requested by a role with those capabilities in CapabilityDatabase. Informs requester role with the ROLENAME whose capability matched with the requested one.

**Protocols and Activities:**
GetRequestedCapability, AccessCapabilityDatabase, InformRequester, CompareCapability

**Permissions:**
- **Reads supplied** RequestedCapability // capability requested by a role
- **Supplied** CapabilityDatabase

**Responsibilities**
- **Liveness:**
  - CAPABILITYMATCHER = GetRequestedCapability.
  (AccessCapabilityDatabase, CompareCapability)+.InformRequester

**Safety:**
- True

---

Figure 4.3: Schema for Role CapabilityRecorder

Figure 4.4: Schema for Role CapabilityMatcher
### Role Schema: DECISIONMAKER

**Description:**
Reaches a conclusion for the delegated task based on the information gathered and knowledge the role has.

**Protocols and Activities:**
ReceiveTask, GetInformation, AskforHelp, InformDecisionAggregator, Reasoning

**Permissions:**
reads supplied Task // Task delegated by other roles
supplied Knownfacts // Gathered information
generates Decision // one alternative decision

**Responsibilities**
Liveness:
DECISIONMAKER =

Safety:
- True

---

**Figure 4.5: Schema for Role DECISIONMAKER**

### Role Schema: HELP-provider

**Description:**
Provides appropriate help for other roles (e.g., DECISIONMAKER) when asked.

**Protocols and Activities:**
ReceiveHelpRequirements, ReturnProcessedResults, AskforHelp, Processing

**Permissions:**
reads supplied HelpRequirements // one role asks another role for help
generates ProcessedResults // do the requested processing and generate results

**Responsibilities**
Liveness:
HELP-provider =
ReceiveHelpRequirements.(Processing.AskforHelp)*.ReturnProcessedResults

Safety:
- HELP-provider has the requested capability

---

**Figure 4.6: Schema for Role HELP-provider**
<table>
<thead>
<tr>
<th>Role Schema: DECISIONAGGREGATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td>Produces the final decision based on currently received alternative decisions for a task.</td>
</tr>
<tr>
<td><strong>Protocols and Activities:</strong></td>
</tr>
<tr>
<td>ReceiveDecisions, InformUserHandle, ChooseDecision, AggregateDecisions</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
</tr>
<tr>
<td>reads supplied Decision // alternative decision</td>
</tr>
<tr>
<td>generates FinalDecision // The &quot;best&quot; one of all alternative decisions or aggregated decision</td>
</tr>
<tr>
<td><strong>Responsibilities</strong></td>
</tr>
<tr>
<td>Liveness:</td>
</tr>
<tr>
<td>DECISIONAGGREGATOR =</td>
</tr>
<tr>
<td>ReceiveDecisions.(ChooseDecision</td>
</tr>
<tr>
<td><strong>Safety:</strong></td>
</tr>
<tr>
<td>• alternative decisions are available</td>
</tr>
</tbody>
</table>

Figure 4.7: Schema for Role DECISIONAGGREGATOR

<table>
<thead>
<tr>
<th>Role Schema: USER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td>Organization or individual requesting a decision.</td>
</tr>
<tr>
<td><strong>Protocols and Activities:</strong></td>
</tr>
<tr>
<td>MakeCall, GiveRequirements</td>
</tr>
<tr>
<td><strong>Permissions:</strong></td>
</tr>
<tr>
<td>generates UserDetails // Owner of user information</td>
</tr>
<tr>
<td>UserDetails // Owner of user requirements</td>
</tr>
<tr>
<td><strong>Responsibilities</strong></td>
</tr>
<tr>
<td>Liveness:</td>
</tr>
<tr>
<td>USER = (MakeCall.GiveRequirements)+</td>
</tr>
<tr>
<td><strong>Safety:</strong></td>
</tr>
<tr>
<td>• True</td>
</tr>
</tbody>
</table>

Figure 4.8: Schema for Role USER
Figure 4.9: Definition of Protocols Associated with the DECISIONMAKER Role: (a) ReceiveTask, (b) GetInformation, (c) AskforHelp, and (d) InformDecisionAggregator.
the task (InformDecisionAggregator protocol, Figure 4.9d).

In the society, the most important organizational rule in the organizational model is that if a role says it has a capability then it can perform the tasks corresponding to the capability and will do so when asked.

4.3 Design of the Society

Having completed the analysis of the society, the design phase follows. The first model to be generated is the agent model (Figure 4.10). This shows, for most cases, a one-to-one correspondence between roles and agent types. The exception is for the CapabilityRecorder and CapabilityMatcher roles which, because of their high degree of interdependence, are grouped into a single agent type.

![Diagram of the Agent Model of the Society]

*Figure 4.10: The Agent Model of the Society*

The second model is the skill model. Again to avoid redundancies the focus is on the DecisionMaker role and the Decision Making Agent. Based on the DecisionMaker role, six distinct services can be identified (Table 4.1).

From the ReceiveTask protocol, the service “obtain task” is derived. This service returns the TaskRequirements as output. It has a pre-condition that the agent or role has the corresponding capability to perform the task, but has no post-condition.

The service associated with the GetInformation protocol is “get information”. Its inputs, derived from the protocol definition (Figure 4.9b), are the requirements for information gathering and its outputs are known facts (gathered information).
<table>
<thead>
<tr>
<th>Service</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>obtain task</td>
<td></td>
<td>task requirements</td>
<td>agent or role</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>has corresponding capability</td>
<td></td>
</tr>
<tr>
<td>get information</td>
<td>requirements for gathering</td>
<td>gathered information</td>
<td>information sources are available</td>
<td>relevant</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>information</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>returned</td>
</tr>
<tr>
<td>accomplish reasoning</td>
<td>known facts (decisions)</td>
<td>conclusions</td>
<td>known facts are enough</td>
<td>conclusions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reached</td>
</tr>
<tr>
<td>call for help</td>
<td>capability description</td>
<td>agent names or nil</td>
<td>true</td>
<td>agent name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>with matched</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>capability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>returned</td>
</tr>
<tr>
<td>provide initial data</td>
<td>agent name and initial data</td>
<td>processed results</td>
<td>initial data are ready</td>
<td>data processed</td>
</tr>
<tr>
<td>inform aggregator</td>
<td>alternative decision</td>
<td></td>
<td>true</td>
<td>aggregator knew</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>decision</td>
</tr>
</tbody>
</table>

The pre-condition for this service is that some information sources are available, and the post-condition is that relevant information is returned.

The third service, "accomplish reasoning", is related to the **Reasoning** activity. The inputs for this service are known facts, and outputs are some conclusions. This service has a pre-condition that the known facts are enough to reach conclusions (post-condition).

The next two services, "call for help" and "provide initial data", are derived from the AskforHelp protocol. The "call for help" service takes a specific capability description as input, and returns the agent name with matched capability or nil (if no matched) as outputs. The "provide initial data" service takes the agent name and initial data as inputs and returns the processed data as outputs.

The final service involves informing the decision aggregator of the alternative decisions. Then the inference mechanisms in the skill model are checked. To make decisions, the agents must accomplish some reasoning based on their knowledge and
other available information. In the society, three kinds of inference mechanisms are needed:

- **Traditional inference mechanisms.** Traditional inference mechanisms such as forward chaining and backward chaining etc. are good at processing exact, certain, and text-based information. They demonstrate very high performance for such problems. Reasoning with rules is needed in the society.

- **Approximate reasoning.** In complex decision making, there is much uncertain, fuzzy, inexact information available. To cope with such information, some approximate reasoning mechanisms are needed.

- **Reasoning with multimedia information.** In many cases, the gathered information is not only in text form, but also in image, graphics, audio, and video forms. To process information in multimedia forms, some inference mechanisms that can reason with multimedia information are required.

The reasoning problem will be discussed in detail in Chapter 7.

The final model is the knowledge model, which indicates different levels of knowledge that agents should have. In the society, the agents should have the three levels of knowledge (see Section 3.4.7). The first level of knowledge for agent interaction and communication (ontologies) will be discussed in Chapter 5. The other two levels of knowledge will be discussed in the case study (see Section 4.6).

### 4.4 Architecture of the Society

From the above analysis and design phases, it is clear that there are seven types of agents in the intelligent technique society – user agent, interface agent, planning agent, middle agent, service provider agent, decision making agent, and decision aggregation agent. Now it is time to find an appropriate multi-agent architecture to put all these agents together so as to form the agent-based hybrid intelligent systems for decision making.

When incorporating intelligent techniques into multi-agent systems to form agent-based hybrid intelligent systems, one criterion that must be met, is that
any agent in the system can access any of the intelligent techniques available in the system when needed. To achieve this goal, the following issues must be considered when choosing the architecture:

- The architecture must facilitate the communication among agents. This implies that all the intelligent technique components as well as other agents in the system must speak some type of common or standard agent communication language. When the agents use a far more expressive language than usually seen in existing systems of this type, it provides for the communication of constraints, disjunctions, rules, quantified expressions and so forth. This rich expressive capability can be especially useful when incorporating intelligent techniques such as expert systems, FL, NN, and GA. Furthermore, one of the criteria Genesereth believes must be met for an agent to be called an agent, is to be able to communicate in an agent communication language such as ACL [40]. This not only means that the agent must read and write ACL messages, but it must also conform to behaviors explicated in those messages, together with other behavioristic principles imposed via the architecture or domain.

- Some kinds of middle agents should play a key role in the architecture so as to be used in open environments such as the Internet. There are different types of middle agents such as matchmakers, brokers, facilitators [39, 41] etc. Middle agents receive the advertisements of agent capabilities and store these advertisements in an internal database. When an agent (a requester agent) would like to find a service provider agent that possesses certain desired capabilities, it sends a request to a middle agent. The middle agent matches the request to its database of received advertisements to determine whether an agent whose capabilities match the request is known.

- As there are many legacy intelligent technique software packages used in different applications, one must make sure that there are some easy ways to convert these software packages into agents when doing the integration.

'With all these observations in mind, an architecture similar to the federation
architecture [40] is adopted. The architecture of the agent-based intelligent technique society is shown in Figure 4.11. The behaviors of each kind of agent in the society (except the user agent) are briefly described below:

**Interface Agent** This agent interacts with the user (or user agent). It asks the user to provide his personal information and requirements, and provides the user with a final decision or advice that best meets the user's requirements.

**Planning Agent** The planning agent is in charge of the activation and synchronization of different agents. It elaborates a work plan and is in charge of ensuring that such a work plan is fulfilled. It receives the assignments from the interface agent.

**Decision Making Agent** It is application-specific, i.e., it has its own knowledge base; it must have some meta-knowledge about when it needs the help of soft computing agents (e.g., pre or post processing some data); it can ask intelligent technique agents to accomplish some sub-tasks.

**Serving Agent** The serving agent is a matchmaker – one kind of middle agent. It keeps track of the names, ontologies, and capabilities of all registered soft computing agents in the system; it can reply to the query of a decision making agent with appropriate soft computing agent's name and ontology.

Figure 4.11: Architecture of Agent-Based Intelligent Technique Society
• **Service Provider Agent** Most of the service provider agents in the society are intelligent technique agents. Each intelligent technique agent can provide services for decision making agents with one or some kind of combined intelligent techniques; it can send back the processed results to decision making agents; it must advertise its capabilities to the serving agent.

• **Decision Aggregation Agent** When decision making agents finish the assigned tasks they return the results to the decision aggregation agent. The aggregation agent chooses one of the alternative decisions, or performs an aggregation of the different results into a final one.

The ontology is the foundation for agent communication. All agents in the society interpret the content of received messages based on the ontology.

The next section discusses the implementation details of the society.

### 4.5 Implementation of the Society

The framework proposed here is general. In order to apply such a general-purpose framework in different applications, the implementation should be platform independent. The technology platform supporting the implementation should facilitate the communication among agents in the society. Keeping these criteria in mind, how to choose an appropriate technology platform that meets the requirement is discussed. The internal structures of the agents in the society are also considered.

#### 4.5.1 Technology Platform

The three issues outlined in the previous section are most pertinent when choosing the architecture. The technology platform should first support and facilitate the communication among agents. Currently there are two main agent communication languages available – FIPA’s (Foundation for Intelligent Physical Agents) ACL (Agent Communication Language) [42, 43] and DARPA KSE’s (Knowledge Sharing Effort) [65] KQML (Knowledge Query and Manipulation Language) [42, 44]. As the latter has more support currently, the KQML is used in the society. The platform should also support easy implementation of middle agents, and provide
some templates to convert legacy software packages into agents. Considering the platform independent requirement, all system components should be developed using Java because it is an architecture-neutral language that is independent of the underlying hardware. Is there any technology platform that can provide all these supports?

Up to now, there are some frameworks or tools – either commercially available products or academic and research projects – for building agent systems. Among them, the Open Agent Architecture (OAA) [17, 18], Interactive Maryland Platform for Agents Collaborating Together (IMPACT, http://www.cs.umd.edu/projects/impact) [19, 55], JATLite (http://java.stanford.edu/), AgentBuilder (http://www.agentbuilder.com/), and JACK (http://www.agent-software.com.au/) etc. have attracted more attention than others. They will be examined now.

- **AgentBuilder.** AgentBuilder is an integrated tool suite for constructing intelligent software agents. AgentBuilder consists of two major components—the Toolkit and the Run-Time System. Agents constructed using AgentBuilder communicate using KQML. All components of both the AgentBuilder Toolkit and the Run-Time System are implemented in Java. There is no directly support for middle agents. AgentBuilder is a commercial product. The newest version is 1.2.

- **IMPACT.** IMPACT is a software platform for the creation and deployment of agents, and agent based systems. The intent in IMPACT is to provide a rich formal theory of agent construction and agent interaction that is practically implementable and realizable. IMPACT servers provide a range of infrastructural services used by agents. It is not publicly available.

- **JACK.** JACK Intelligent Agents is an environment for building, running and integrating commercial-grade, multi-agent systems using a component-based approach. It is entirely written in Java, and provides the core architecture and capability for developing and running software agents in distributed applications. The newest version is 3.1, but it was not popular in early 1999 when this project was started. JACK is also a commercial product.
- **JATLite.**  JATLite (Java Agent Template, Lite) is a package of programs written in Java that allow users to quickly create new software agents that communicate robustly over the Internet. JATLite provides a set of Java templates and a ubiquitous Java agent infrastructure that makes it easy to build systems in a common way. JATLite facilitates especially construction of agents that send and receive messages using the emerging standard communication language KQML. JATLite does not impose any particular theory of agents. There is no commitment to any standard of agent internal architecture. The source codes of JATLite are available. All these give agent developers extreme flexibility.

- **OAA.**  The Open Agent Architecture (OAA) is a framework for integrating a community of heterogeneous software agents in a distributed environment. In this framework, agents speak the Interagent Communication Language (ICL). In the OAA ICL there are only two performatives: "solve" that is used to query other agents, and "solved" that is used to answer the query. Communication and cooperation between agents is brokered by one or more facilitators, which are responsible for matching requests, from users and agents, with descriptions of the capabilities of other agents. OAA is structured so as to minimize the effort involved in creating new agents and "wrapping" legacy applications, written in various languages and operating on various platforms; to encourage the reuse of existing agents; and to allow for dynamism and flexibility in the makeup of agent communities. The first release of OAA (version 1.0) is November 1998. The supporting document for the Java library was released in February 1999. The newest version is 2.0.

The comparison of these agent development platforms is summarized in Table 4.2. (For a more comprehensive list of agent construction tools, refer to [http://www.agentbuilder.com/AgentTools/index.htm](http://www.agentbuilder.com/AgentTools/index.htm)). The comparison is based on the support for KQML, implementation of middle agents, integration with legacy software, whether it is written in Java, and whether it was available in early 1999.

Through this comparison, it is easy to discover that JATLite and OAA are two most appropriate tools for our purpose. Considering JATLite has source codes available and better supporting documents when this project was started in early
1999, finally JATLite was chosen as development platform of the agent-based intelligent technique society.

### 4.5.2 Internal Structures of Agents

Under the framework, all decision making agents or intelligent technique agents must register and connect to the serving agent.

Each decision making agent has its own domain-specific knowledge base as well as meta-knowledge about when to use intelligent technique agents. The serving agent records the capabilities, ontologies, and names etc. of all the intelligent technique agents in a multi-agent system. The scenario goes as follows.

At certain stage of the decision making process, the decision making agent sends a KQML message using `recommend-one` performative to the serving agent according to its meta-knowledge. The serving agent then retrieves its service provider agent database and replies with an appropriate service provider agent’s name and ontology which has the capability asked for using `reply` performative. After that, the decision making agent communicates with the service provider agent directly for a specific problem. In most cases, service provider agents in the society are intelligent technique agents. The decision making agent provides the intelligent technique agent with some parameters according to the ontology, and the intelligent technique agent sends the results to the decision making agent.

Based on the above description, the internal structures of the agents in the society can be identified. Figure 4.12 to Figure 4.17 show the internal structures.
of these agents.

As one can see from Figure 4.12 to Figure 4.17, all the agents have a common part – KQML Message Interpreter (KMI). That is because KQML is used for inter-agent communication. The KMI represents the interface between KQML router and agents. Once an incoming KQML message is detected, it will be passed to the KMI. The KMI transfers incoming KQML messages into a form that agents can understand. The implementation of KMI is based on JATLite KQMLLayer Templates.

All the agents except the serving agent have an ontology interpreter. They need
Figure 4.15: Aggregation Agent Structure

Figure 4.16: Planning Agent Structure

Figure 4.17: Interface Agent Structure
to decrypt and process the :content part of the KQML message when they solve a problem. There is no ontology interpreter in the serving agent because it does not care about the :content.

The domain knowledge in decision making agents is not sufficient to make a decision. Relevant information and skills to use the knowledge and information are also needed. They usually have limited skills rather than all the skills needed for decision making. Thus they need the help of intelligent technique agents and other service provider agents for data pre- and/or post-processing. The meta-knowledge in decision making agents tell them when to ask for helps of intelligent technique agents.

The intelligent technique agent maintenance module in the serving agent has three functions: To add an entry that contains the intelligent technique agent’s name, capability, and ontology to the database; delete an entry from the database; and retrieve the database to find out intelligent technique agents with a specific capability. The last function is usually called matchmaking.

For the algorithms based on intelligent techniques in intelligent technique agents, if the agent is under control, it will be built using KQML as a communication language. If not, the Java Native Interface [52] is used to connect the legacy system to the agent society.

4.6 Case Study – Determining a Client’s Investment Policy

This section focuses on how to use the proposed agent-based intelligent technique society to solve practical problems that need to integrate different intelligent technique into agents. This is done through an example of financial investment application – determining a client’s investment policy. It will also demonstrate how agents in the society exchange KQML messages, and how to convert legacy programs written in C/C++ into agents. Some materials about this case study can also be found in [26, 45]. Firstly, the problem of determining a client’s investment policy is described. More about KQML is mentioned as it is used. In determining a client’s investment policy, there is a need to do some approximate reasoning
based on fuzzy logic. This task is very time-consuming. Thus one way to speed up appropriate reasoning – parallel implementation – is provided (refer to Chapter 7). How to convert this parallel approximate reasoning program (coded in C language) into an agent is discussed. Finally, some experimental results are given.

4.6.1 Description of the Problem

When giving investment advice to the client, the first thing a financial investment planning system needs to do is to determine the client’s investment policy. Based on the client’s investment policy (aggressive or conservative etc.), the system can then decide in which categories (stock market, real estate, etc.) the client should invest.

To make a decision about a client’s investment policy (IP), decision making agents need the information about the client’s financial risk tolerance (RT) ability, the falling or rising of interest rates (P1), the state of the stock market (P2), and the unemployment rate (P3) etc. Decision making agents use rules in their domain knowledge bases such as

\[
\text{If RT is } H \text{ and } P_1 \text{ is } B_1 \text{ and } \ldots \text{ then IP is } C
\]

to make the decision (where C is a fuzzy subset indicating the aggression or conservation of the investment policy. H and B1 are also fuzzy subsets).

The agents use the client’s annual income and total net-worth to evaluate the client’s financial risk tolerance ability. In this example, assume the agents agree to describe the input variables annual income and total net-worth and the output variable risk tolerance by the sets:

\[
\text{annual income} = \{L, M, H\}, \text{total networth} = \{L, M, H\}, \text{risk tolerance} = \{L, MO, H\}.
\]

The terms have the following meaning: L = low, M = medium, H = high, and MO = moderate. They are fuzzy numbers whose supporting intervals belong to the universal sets \(U_1 = \{x \times 10^3 | 0 \leq x \leq 100\}\), \(U_2 = \{y \times 10^4 | 0 \leq y \leq 100\}\), \(U_3 = \{z | 0 \leq z \leq 100\}\). The real numbers \(x\) and \(y\) represent dollars in thousands and hundred of thousands, correspondingly, while \(z\) takes values on a psychometric
scale from 0 to 100 measuring risk tolerance. The numbers on that scale have
specified meaning for the financial experts (agents).

For a simplified client financial risk tolerance model, the following nine approx-
imate reasoning rules are used to determine client's risk tolerance [46]:

Rule 1: If the client's annual income ($AI$) is low ($L$) and the client's total
networth ($TN$) is low ($L$), then the client's risk tolerance ($RT$) is low ($L$);
Rule 2: If $AI$ is $L$ and $TN$ is $M$, then $RT$ is $L$;
Rule 3: If $AI$ is $L$ and $TN$ is $H$, then $RT$ is $MO$;
Rule 4: If $AI$ is $M$ and $TN$ is $L$, then $RT$ is $L$;
Rule 5: If $AI$ is $M$ and $TN$ is $M$, then $RT$ is $MO$;
Rule 6: If $AI$ is $M$ and $TN$ is $H$, then $RT$ is $H$;
Rule 7: If $AI$ is $H$ and $TN$ is $L$, then $RT$ is $MO$;
Rule 8: If $AI$ is $H$ and $TN$ is $M$, then $RT$ is $H$;
Rule 9: If $AI$ is $H$ and $TN$ is $H$, then $RT$ is $H$.

In this application, approximate reasoning (part of the skill model) is used for
determining the client’s investment policy and risk tolerance ability. The parallel
implementation of approximate reasoning is discussed in Chapter 7. Here the
parallel approximate reasoning program (written in C) is used directly.

4.6.2 More about KQML

KQML is a high level, message oriented communication language and protocol for
information exchange. It is independent of content syntax, delivery technique and
ontology. This means that KQML is independent of the content language which is
used to deliver content within the message. It is also independent of the delivery
mechanism, i.e., TCP/IP or others.

KQML has a three layer structure – content, communication and message. The
content of a message is essentially ignored by every KQML implementation. The
content can be represented in the agents’ own content representation language and
KQML does not take any note of the section of a message. The communication
section contains the lower level details of communication. This includes information
such as the message sender and receiver, and the unique identifier associated with
the communication. The final layer within KQML is the communication layer and is
the heart of the KQML language. It contains the actual performative. Performative is the name given in KQML to describe any primitive KQML message type. The performative describes if the message is a query, a command, a request, or other primitive message types.

Other features that the communication layer implements include allowing specification of content language, ontology used and a description of the content. These features make it possible for KQML implementations to analyze, route and properly deliver messages whose content is inaccessible. The syntax of KQML messages is relatively simple. The first element is the performative, followed by the performative arguments. Below is an example of KQML message.

(ask-one
 :sender joe
 :content (PRICE IBM ?price)
 :receiver stock-server
 :reply-with ibm-stock
 :language LPROLOG
 :ontology NYSE-TICKS)

In the case of this message, the performative is “ask-one”, meaning the message is a directed message to a certain individual. The sender and receiver can be deducted quickly from their labels, sender and receiver. The label language defines that the content is in “LPROLOG”, and the actual message is paired with the content label. In this case the message is simply querying the price of IBM stock. The following breaks down the message into its three layers:

- Content layer: content;
- Communication layer: sender, receiver and reply-with;
- Message layer: performative, language and ontology.

4.6.3 Converting Legacy Programs into Agents

When integrating the parallel approximate reasoning program into multi-agent systems, we meet the problem of how to convert programs written in programming
languages other than Java into agents. The parallel approximate reasoning program is written in parallel C used in PVMs. This program must be equipped with KQML communication ability, and make it work harmoniously with other agents coded in Java.

Generally, there are three main approaches to be taken: implementing a transducer, implementing a wrapper, and rewriting the original program (see Section 2.4).

Here the second approach – implementing a wrapper – was adopted to wrap the parallel approximate reasoning program by using Java Native Interface [52] and JATLite KQML layer templates.

In order to access a native method (typically written in C/C++) from a Java program, a class was created for the native method and the native method invoked using normal Java method invocation syntax. Native methods are created using the following steps:

- Create a Java class for the native method and include code to load the native method's shared library (under Unix) or dynamically linked library (under Microsoft Windows);

- Use javah to create C language header files for the native method;

- Implement the native method as a C function (like our parallel approximate reasoning program);

- Compile and link the C code to create the shared library or dynamically linked library.

### 4.6.4 Experiments and Results

Based on the general intelligent technique society framework (Figure 4.11), the practical architecture adopted for determining the client's investment policy is shown in Figure 4.18 (under the support of JATLite).
Exchanging KQML Messages

In this example, the service requester agent (decision making agent) has metaknowledge such as using soft computing (SC) agents to evaluate user's RT and using SC agents to predict interest rate $P_1$ etc. Thus, the decision making agent sends KQML messages using `recommend-one` performative to the serving agent:

```
(recommend-one
  :sender investment_policy_agent
  :receiver serving_agent
  :language KQML
  :content (ask
    :ability risk_tolerance_evaluation
    :name ?
    :ontology ?
  )
)
```

```
(recommend-one
  :sender investment_policy_agent
  :receiver serving_agent
)
```

\footnote{SC agents are a subset of intelligent technique agents.}
The serving agent then retrieves its database and replies with name and ontology (see Chapter 5) of an appropriate service provider agent (here is an SC agent) which has the capability asked for using *reply* performative. This matchmaking process is accomplished by using *Nearest Neighbor* matchmaking algorithm (see Chapter 6). In this case study, there are a risk tolerance ability evaluation agent based on fuzzy logic (SC-Agent_FL) and an interest rate prediction agent based on feed-forward neural network (SC-Agent_NN).

{(reply
  :sender serving_agent
  :receiver investment_policy_agent
  :content (:name SC_agent_FL
            :ontology Financial_investment
    )
)

{(reply
  :sender serving_agent
  :receiver investment_policy_agent
  :content (:name SC_agent>NN
            :ontology Financial_investment
    )
)

The decision making agent then communicates with SC-Agent_FL (for risk tolerance evaluation) and SC-Agent_NN (for interest rate prediction) directly.
Running the System

The agents in this example are listed in Table 4.3:

<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Agent Type</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serving Agent</td>
<td>Middle Agent</td>
<td>Match capabilities of SC agents with that requested by Decision Making agents</td>
</tr>
<tr>
<td>SC.Agent_NN</td>
<td>SC Agent (Service Provider)</td>
<td>Predict interest rate using NN model</td>
</tr>
<tr>
<td>SC.Agent_FL</td>
<td>SC Agent</td>
<td>Evaluate client’s risk tolerance ability using approximate reasoning based on FL</td>
</tr>
<tr>
<td>SC.Agent_AR</td>
<td>SC Agent</td>
<td>Parallel approximate reasoning</td>
</tr>
<tr>
<td>DM.Agent</td>
<td>Decision Making Agent (Service Requester)</td>
<td>Making decisions with the help of SC agents</td>
</tr>
</tbody>
</table>

The actual running scenario goes as follows:

With the JATLite Agent Message Router started up and running, all agents should register and connect themselves to the Router. Once the serving agent is connected, intelligent technique agents (In this case study, they are SC agents including SC.Agent_NN, SC.Agent_FL, and SC.Agent_AR) can advertise their capabilities and ontology used to the Serving Agent. The serving agent receives these advertisements and places the data in a local database. The process is depicted by Figure 4.19.

![Figure 4.19: Intelligent Technique Agents Advertise Capabilities to Serving Agent](image)

The decision making agent sends requests (consisting of keywords of task to
be completed) to the serving agent, inquiring about service provider agents (SC agents) that can accomplish the specific task. The serving agent matches the query with its database and replies to the decision making agent that made the request.

Performance of the System

First a search performance for the serving agent was tested. The serving agent in this framework is actually a matchmaker. The matchmaking algorithm discussed in [19, 55] was adopted. We tested the time spent by the serving agent to search its capability database. We started with the database having only one entry. A match from request to response was about 3 seconds. The number of entries in the database is increased gradually by sets of five till it reached 1000. The time spent by the serving agent to retrieve and match a suitable intelligent technique agent was still under 4 seconds.

The whole system was then put together for the test. With all the agents up and running, we demonstrated a fully working basic framework with functional serving agent and basic intelligent technique agents. This test also ensured that all agents exhibited their correct behaviors as specified.

The results of these tests show that the performance of the system depends especially on the number of messages sent by various agents. The access time to the database is negligible compared to the inter-agent communication time.

4.7 Summary

To make good decisions, agent-based hybrid intelligent systems are needed for complex decision making. This is because:

- multi-agent approaches are good at complex problems;

- employing a combination of different intelligent technique can increase agents' "intelligence"; and

- hybrid intelligent systems combining expert systems, FL, NN, and GA etc. allow the representation and manipulation of different types and forms of
data and knowledge which may come from various sources. Refined system knowledge is used during reasoning and decision making processes producing more effective results.

A flexible framework – the agent-based intelligent technique society is presented – to construct multi-agent application systems that need to incorporate different kinds of intelligent techniques into them. Based on the methodology proposed in Chapter 3, the analysis, design, and implementation of the agent-based hybrid framework for complex decision making was systematically discussed. The framework has the following crucial characteristics that differentiate this work from others:

- The ability to exchange comprehensible communications (interactions at knowledge level [54]);

- Each service requester agent (decision making agent) can easily access all the techniques provided by service provider agents (e.g., SC agents) in the system;

- The presence of the serving agent in this framework allows adaptive system organization. For example, if one is sure that one hybrid SC agent can do a better job than a single technology SC agent, one can delete the single SC technology agent and add the hybrid agent to the society simply by adding or deleting a record in the database of the serving agent;

- Overall system robustness is facilitated through the use of the serving agent. For example, if a particular service provider (e.g., an SC agent) disappears, a requester agent (decision making agent) can find another one with the same or similar capabilities by interrogating the serving agent; and

- Agent-based hybrid intelligent systems based on the framework can make decisions about the nature and scope of interactions at run time.

A case study was also presented for determining a client’s investment policy using the proposed intelligent technique society, and some experiments were conducted to test the society under the support of JATLite. The tests indicated that
the framework is fully functional. In this case study, the intelligent techniques integrated are mainly soft computing techniques.

When the agents communicate with each other in the society and the serving agent performs matchmaking tasks, they all need to access the corresponding domain specific ontologies (part of the knowledge model). As an example, constructing an ontology for financial investment will be discussed in Chapter 5.

A question frequently asked in multi-agent systems is the efficient search for suitable agents to solve a specific problem. To answer this question, different types of middle-agents that can assist in locating and connecting the ultimate service provider with the ultimate service requester are usually employed. For this purpose, we employed a middle agent (serving agent) in this agent-based intelligent technique society. To our knowledge, almost all currently used matchmaking algorithms (including the one used in the intelligent technique society) are only based on the advertised capabilities of providers. They did not consider the providers' practical outcomes in accomplishing delegated tasks at all. In fact, the practical performance of service provider agents has a significant impact on the matchmaking outcomes of middle agents. To this end, in Chapter 6 some algorithms are developed to consider the practical performance of service provider agents.
Chapter 5

Building an Ontology for Financial Investment

The agents in the agent-based intelligent technique society should have three levels of knowledge (the knowledge model was discussed in Chapters 3 and 4). The first level of knowledge is for agent interaction and communication.

Recalling the case study in Chapter 4, the same capability may have different descriptions in intelligent technique agents and decision making agents. For example, one soft computing agent advertises its capability to the serving agent as "pattern watcher in the stock market", whereas one decision making agent requests a soft computing agent that is a "pattern watcher in the share market". In such a situation, problems arise when the serving agent tries to match them. How could the serving agent know the "stock market" and the "share market" are the same thing? Moreover, in complex problem solving and decision making, the agents in a multi-agent system need to coordinate, cooperate or communicate with each other. Communications among agents should be based on some common knowledge, as with human beings. When one talks about something, the other person must have the same background. Otherwise, they cannot understand each other.

To answer these questions, ontologies are employed. Ontologies are a key component in how different agents in a multi-agent system can communicate effectively, and how the knowledge of agents can develop [56]. Most methods to resolve semantic heterogeneities rely on using partial or global ontological knowledge which
may be shared among agents [144].

The American Heritage Dictionary defines “ontology” as “the branch of metaphysics that deals with the nature of being.” The term has recently been adopted by the artificial intelligence community to refer to a set of concepts or terms that can be used to describe some area of knowledge or build a representation of it [57]. An ontology can be very high-level, consisting of concepts that organize the upper parts of a knowledge base such as the WordNet [58, 61], or it can be domain-specific, such as an ontology for finance.

When applying multi-agent systems to different application domains, different domain-specific ontologies are needed to support agent communications. This is one part of the knowledge model. This is also why a domain-specific ontology was included in the proposed agent-based hybrid framework for complex decision making (see Figure 4.11). As financial investment planning problems are used as examples in this thesis, it is necessary to build an ontology for financial investment used in the prototype.

Before discussing the construction of the financial ontology, the distinction between an ontology and a knowledge base should be clarified. An ontology provides the basic structure or armature around which a knowledge base can be built. An ontology provides a set of concepts and terms for describing some domain, while a knowledge base uses those terms to represent what is true about some real or hypothetical world. Thus, a financial ontology might contain working definitions of concepts like money, banks, and stocks, but it would not contain assertions that a particular investor bought some securities, although a knowledge base might.

The goals for developing this ontology include:

- facilitate inter-operation and communication between systems with common terminology;

- promote knowledge sharing between systems – in particular, integrate the knowledge acquisition and modeling efforts; and

- create a repository for general knowledge about financial investment to use in other applications, including traditional (not agent-based) ones.
In this chapter, the development of the financial ontology as well as lessons learned from the developing process is described. Based on these discussions, a framework for the next generation of ontology construction tools is proposed. The work in this chapter is also reported in [28].

5.1 Ontologies in Finance

Interest in ontologies has grown as researchers and system developers have become more interested in reusing or sharing knowledge across systems. There are some general-purpose upper ontologies such as CYC [60] and Wordnet [61] and some domain-specific ontologies that focus on the domains of discourse such as chemicals ontology and air campaign planning ontology. (Refer to [56] for an overview of the recent development of the field of ontologies in artificial intelligence). Until now, very few financial ontologies have been reported. In the Larlast project, a financial domain ontology is under construction and will be used for learning finance terminology (http://www.linglink.lu/htl/projects/larlast-inco/ar-99/ar99.html and [59]). In I3 (Intelligent Integration of Information, http://dc.isx.com/I3) project, there is a financial ontology and databases group. They are creating ontologies of financial knowledge in Loom (a kind of knowledge-representation language, http://www.isi.edu/isd/LOOM/documentation/LOOM-DOCS.html) that describe the contents of existing financial databases. We failed to find an existing financial ontology that can be (re)used in multi-agent environments. The lack of financial ontology that can be (re)used directly motivated us to build such an ontology.

5.2 Construction of Financial Ontology

To facilitate the construction of multi-agent application systems in finance, to support the communication among agents in the prototype multi-agent system, and to ease the implementation of Nearest Neighbor matchmaking algorithm in the serving agent, an ontology used in finance was constructed. An ontology for finance would provide working definitions of concepts like money, banks, and stocks. This knowledge is expressed in computer-usable formalisms.
Ontolingua was used to construct the financial ontology. Ontolingua is an ontology development environment that provides a suite of ontology authoring and translation tools and a library of modular reusable ontologies. For details on Ontolingua, visit http://ontolingua.stanford.edu. Ontolingua is based on a top ontology that defines terms such as frames, slots, slot values, and facets. When building the ontology using Ontolingua, the terms such as portfolio, security, share, stock, bond etc. must be defined by determining the slots and giving the slot values. Before this can be done, a hard knowledge-acquisition problem is faced. Like knowledge-based-system development, ontology development faces a knowledge-acquisition bottleneck.

In addition to the knowledge acquisition problem, another two problems involved in the construction of financial ontology are coding and accessing the ontology.

5.2.1 Knowledge Acquisition

Because we are not experts in financial domain, we first read the financial literature to acquire some basic concepts in financial investment. Then preliminary meetings were held with financial experts to look for general, not detailed, knowledge. After this, we studied the documentation very carefully and tried to learn as much as possible about the financial domain. Having obtained some basic and general knowledge, we gradually moved into the particular details for configuring the full ontology. Sets of terms and their relationships were extracted, and then attributes and their values defined. At the later stage of knowledge-acquisition, these were submitted to financial experts for inspection. During this knowledge acquisition, the following set of knowledge-acquisition techniques were used in an integrated manner [9]:

- Non-structured interviews with experts to build a preliminary draft of the terms, definitions, a concept classification, and so on;
- Informal text analysis to study the main concepts in books and handbooks;
- Formal text analysis. This was performed manually without using specialized environments for this purpose. We analyzed the text to extract attributes,
natural-language definitions, assignation of values to attributes, and so on;

- Structured interviews with experts to get specific and detailed knowledge about concepts, their properties, and their relationships with other concepts;

- Detailed reviews by experts. At this later stage of knowledge-acquisition, we submitted the knowledge acquired to experts for detailed inspection. In this way, we could get some suggestions and corrections from financial experts before coding the knowledge.

After acquiring the knowledge, we then constructed the financial ontology conceptual structures based on the acquired knowledge. The conceptual structure of portfolio is shown in Figure 5.1. Unfortunately, this is a time consuming and error containing process. Obviously, more efficient construction tools are needed.

### 5.2.2 Coding the Ontology

As mentioned previously, Ontolingua was used to construct the financial ontology. We manually coded the knowledge with Ontolingua based on the conceptual structures. Currently, one can log on to Ontolingua and check the financial investment ontology in the unloaded category. Some terms of the financial ontology written in Ontolingua are as follows:

```plaintext
::: Securities
```
(Define-Class Securities (?X) "A term that covers the paper certificates that are evidence of ownership of bonds, debentures, notes and shares."
:Def (And (Relation ?X)))

;;; Share
(Define-Class Share (?X) "A unit of equity capital in a company."
:Def (And (Securities ?X)))

5.2.3 Accessing the Ontology

To use this ontology, we adopted the Open Knowledge Base Connectivity (OKBC) protocol [62] as a bridge between the agents in the multi-agent system for financial investment planning and the financial ontology. The ontology constructed using Ontolingua is in Lisp format. Before one can access the ontology through OKBC, one must translate the ontology into OKBC format. This can be accomplished automatically by using the ontology server. One piece of the ontology in OKBC format is shown below:

(define-okbc-frame Securities
   :frame-type
   :class
   :direct-superclasses
   (Relation)
   :direct-types
   (Class Primitive)
   :own-slots
   ((Arity 1)))
   :template-slots
   (((Belongs_To)(Have-Some)(Has-One))
   :template-facets
5.3 Lessons Learned

By analyzing the ontology construction process described in Section 5.2 – it is a typical procedure followed by most researchers in this area – the following two points can be extracted:

- Switching directly from knowledge acquisition to implementation;
- Manually coding the required knowledge for the domain of interest.

It is these two points that cause the following problems or disadvantages.

Firstly, the primary current disadvantage in building ontologies is the danger of developing ad-hoc solutions. Usually, the conceptual models describing ontologies are implicit in the implementation codes. Making the conceptual models explicit usually requires re-engineering. Ontological commitments and design criteria are implicit but must be explicit in the ontology code. All these imply that the built ontologies may contain errors, inconsistencies etc. Secondly, domain experts and human end users have no understanding of formal ontologies codified in ontology languages. Thirdly, as with traditional knowledge bases, direct coding of the knowledge-acquisition result is too abrupt a step, especially for complex ontology. Finally, ontology developers might have difficulty understanding implemented ontologies or even building new ontologies. This is because traditional ontology tools focus too much on implementation issues rather than on design problems.

The source of these problems is the absence of an explicit and fully documented conceptual model upon which to formalize the ontology. To this end, some researchers have proposed ontological engineering [63]. Central to ontological engineering is the definition and standardization of a life cycle ranging from requirement
CHAPTER 5. BUILDING AN ONTOLOGY FOR FINANCE

specification to maintenance, as well as methodologies and techniques that drive ontology development. They have developed a framework called Methontology for specifying ontologies at the knowledge level, and an Ontology Design Environment (ODE). The knowledge acquisition result is not coded by target language, but represented by an intermediate representation. The knowledge in intermediate representation can be automatically converted to Ontolingua codes by using ODE.

Using Methontology and ODE can alleviate some of the problems mentioned above. For example, at the later stage of the financial ontology development, only following the idea of ontological engineering (not accessing the ODE) speeds up the ontology construction. To overcome all the difficulties mentioned, more powerful tools or frameworks are still essential.

5.4 The Framework of the Next Generation Ontology Construction Tools

Ontology construction is difficult and time consuming and is a major barrier to the building of large-scale intelligent systems and software agents. Generally, building ontologies is difficult for three reasons [58]. Firstly, articulating knowledge in sufficient detail so that it can be expressed in computationally effective formalisms is difficult. Secondly, the scope of shared background knowledge underlying interactions of two agents can be enormous. Thirdly, there are unsolved problems in using large bodies of knowledge effectively, including selecting relevant subsets of knowledge, handling incomplete information, and resolving inconsistencies.

It is clear that the creation of easy-to-use tools for creating, evaluating, accessing, using, and maintaining reusable ontologies by both individuals and groups is essential. Based on this ontology building experience and a relatively profound analysis of the current development of ontologies, we propose that the next generation ontology construction tools should include the following capabilities:

- Assemble and extend modules from ontology repositories. A primary means of overcoming the high cost of developing ontologies is to store reusable ontology modules in online repositories and provide tools that enable the ontologies for
a given application to be constructed by assembling and extending modules from the repositories;

- *Adapt and reconcile ontologies.* When ontologies are being used by agents, tools are needed to adapt and reconcile differences that emerge from agent communication. Such tools will be used as mediators that can understand other agents’ abstractions and translate between them. By doing so, agents can agree on a common communication substrate;

- *Extract and taxonomize terms from other sources.* The vocabulary in an ontology is often gathered from documents, database schema, or object schema. Tools must identify the technical terms in such sources and enable the ontology developer to quickly organize those terms into class-subclass taxonomies, relations, and functions;

- *Semi-autonomously synthesize ontologies based on the use of terms in natural language documents.* As mentioned in Section 5.3, knowledge acquisition is a bottleneck in ontology construction. Tools that can semi-autonomously synthesize ontologies based on the use of terms in natural language documents will dramatically reduce the workload in informal and formal text analysis;

- *Merge overlapping ontologies.* Ontology building often involves merging existing ontologies that describe the same domain using differing vocabularies. Integrating the vocabularies of two ontologies involves determining and specifying relationships between the intended meanings of the non-logical symbols (symbols that name the classes, relations, and functions) of the first ontology and the non-logical symbols of the second ontology. Reasoning tools are needed to assist with that task by deriving and asserting equivalence, subsumption, and disjointness relationships among the non-logical symbols of the ontologies. Name comparison tools must identify candidate equivalent and subsuming symbols from the ontologies being merged. Facilities then must enable the ontology builder to add equivalence, subsumption, and disjointness relationships that are intended but not entailed by the source ontologies;

- *Visualize ontologies.* Knowledge acquired from domain experts is needed to
CHAPTER 5. BUILDING AN ONTOLOGY FOR FINANCE

submit to experts for detailed inspection. Visualization tools are very helpful for domain experts as well as ontology developers to validate the knowledge. For example, it is much easier to find the errors or inconsistencies in ontologies from visualized relationships of concepts than from reading the corresponding text. One picture is stronger than one-thousand words.

- Detect inconsistencies. Ontologies require tools that check for logical inconsistencies and let developers use the ontology to describe familiar situations;

- Browse and retrieve ontologies. As ontology repositories become larger and ontology content become more sophisticated, it will become an increasingly difficult task to identify ontologies in an ontology repository that satisfy application-specific requirements. Browsing and querying tools are needed to support retrieval, and effective techniques for indexing ontologies in repositories by topics or by classes in a standard top-level ontology must be developed;

- Translate and reformulate. To incorporate ontologies from an ontology repository into an application system, the knowledge must be translated into that system's representation language and might need to be reformulated to satisfy application-specific requirements regarding abstraction level, perspective, underlying assumptions, and usability by problem solving methods.

In [64], Lopez et al. present a vision of next generation tools and services that will enable the widespread development and use of computer-interpretable ontologies. Central to their vision is the notion of distributed ontology repositories that use a network application programming interface for ontology servers. Compared to our proposed framework, there are at least two points missed – adaptation and reconciliation of ontologies and visualization of ontologies. Lopez et al. are building ontology development and use technology that addresses seven (out of nine) of these needs. The technology builds on the results of the DARPA Knowledge Sharing Effort [65], specifically by using Knowledge Interchange Format (KIF) as a core representation language and the Ontolingua system as a core ontology-development environment.
We can use the current visualization technologies [66] to visualize the spatial relationships, temporal relationships, concept/document associations, and complex, aggregate patterns etc. in ontologies.

The theory of logic programming is a possible solution for adaptation and reconciliation of ontologies. Logic programming provides a good mix between formalism and experimentation. It is a high level language amenable to formal reasoning. A logic program viewed as an executable specification encourages rapid prototyping. Logic programming can profitably address semantic issues in the integration of a heterogeneous system by specifying methods to resolve conflicts, pool information together and define new operations based on operations belonging to different domains [67]. Hence, our framework of next generation ontology construction tools is reasonable and actual.

Ontology construction is difficult and time consuming and is a major barrier to the building of large-scale intelligent systems and software agents. Tools with the nine capabilities will contribute greatly to remove the barrier.

5.5 Summary

Ontologies play a key role in how different agents in a multi-agent system can communicate effectively and how the knowledge of agents can develop. This chapter presented a case study in building an ontology in financial investment. The financial ontology constructed facilitates the communications of agents in the intelligent technique agent society as well as the matchmaking in the serving agent of the society.
Chapter 6

Matchmaking in Middle Agents

It is timely to recall the CapabilityMatcher role outlined in Chapter 4. This role involves matching capabilities requested by a role with those capabilities in a CapabilityDatabase and informing the requester role with the RoleName whose capability matched with the requested one. The CapabilityRecorder role and CapabilityMatcher role are grouped into a single agent type (Middle Agent) because of their high degree of interdependence.

Generally, as far as the matchmaker or broker is concerned, there are two relatively independent sub-problems to capability matchmaking:

- **Capability Storage** (corresponding to the CapabilityRecorder role):
  The matchmaker has to store the capability descriptions received. The most important question here is how capabilities can be described or represented in a way that is useful to the matchmaker.

- **Capability Retrieval** (corresponding to the CapabilityMatcher role):
  The matchmaker has to find service provider agents that have the capabilities required to solve the given problem. The most important question here is how capability descriptions can be reasoned about, that is, capability matchmaking.

There is some work related with capability description as well as capability retrieval (matchmaking). The performance of middle agents relies heavily on the matchmaking algorithms used. To our knowledge, almost all the currently used
algorithms missed one point when doing matchmaking — the matchmaking is only based on the advertised capabilities of provider agents. The actual performance of provider agents in accomplishing delegated tasks is not considered at all. This results in the inaccuracy of the matchmaking outcomes as well as the random selection of provider agents with the same advertised capabilities.

To this end, it is argued that the practical performance of service provider agents has a significant impact on the matchmaking outcomes of middle agents. Our idea is to consider the track records of agents in accomplishing delegated tasks. This includes the acquisition of track records and the use of track records. As the track records of agents are accumulated gradually with the running of the agent system, there is no track record available when the system is just launched. For this reason, algorithms to provide "initial values" for the track records is proposed. The new algorithms will be tested under the agent-based intelligent technique society (see Chapter 4), but there is no limitation on applying the improved algorithms in other applications.

This chapter starts from describing the matchmaking problem. Then some related work of matchmaking in middle agents is recounted, followed by the capability retrieval (matchmaking) algorithm currently used in our prototype. Based on these descriptions, the algorithms for the acquisition of track records and the use of track records are discussed.

6.1 Description of the Problem

One of the basic problems facing designers of MASs used in open environments, such as the Internet, is the connection problem — finding suitable agents (service providers) who might have the information or other capabilities requested by service requesters [39]. Because an agent may not have the necessary knowledge or the necessary resources, the situation can arise for both the problem division and the solution of sub-problems that an agent would like to delegate a task to another agent. Generally, there are three possible ways to do this:

- The meta-knowledge based approach. Provided the requester knows a suitable partner, it does not require any external assistance. It can contact
the appropriate provider and then negotiate the transfer of the task with it. However, it is difficult or even impossible for requesters to have comprehensive knowledge of providers in many application domains, especially in open, dynamic environments.

- The contract net based approach [135] (pp. 96-100) and [136]. Within the contract net framework, the requester (manager) can make a public offer (broadcast) for bids as a so-called contract for every pending sub-problem that is to be solved. The offer for bids is open for all agents (providers). Any agent in a contract net system can be a service requester as well as a service provider.

- The middle agent based approach [39]. Like middle-men in the physical world, middle agents can be employed to assist in locating and connecting the ultimate service provider with the ultimate service requester. This approach is very flexible. Agents (providers and/or requesters) can dynamically enter and exit open MASs.

In open MASs, the middle agent based approach is the most commonly used one to efficiently search for suitable agents to solve a specific problem [39, 137]. Thus the discussion here concentrates on middle agents.

Different systems define their middle agents differently. For example, facilitators in Genesereth’s federated systems [40] and SRI’s Open Agent Architecture [17, 18], and matchmakers and brokers in Retsina [12] all differ in their interactions with providers and requesters. In [39], the authors identified different types of middle agents on the Internet, such as matchmakers (yellow page services), brokers, blackboard etc., and experimentally evaluated different protocols for inter-operation between providers, requesters, and various types of middle agents. The results show that different types of middle agents exhibit different characteristics in terms of privacy, robustness, adaptiveness, etc. Regardless of which kinds of middle agents are used, their performance relies heavily on the matchmaking algorithms adopted. Matchmaking is the process of finding an appropriate provider for a requester through a middle agent, and has the following general form [138]:

- Provider agents advertise their capabilities to middle agents;
• Middle agents store these advertisements;
• A requester asks some middle agent whether it knows of providers with desired capabilities; and
• The middle agent matches the request against the stored advertisements and returns the result, a subset of the stored advertisements.

Matchmaking in middle agents is a crucial issue in multi-agent systems, especially those used in open environments such as the Internet. To improve the matchmaking performance of middle agents so that they can pick up the “right” service provider agents is of paramount importance.

6.2 Related Work of Matchmaking in Middle Agents

Agent matchmaking has been actively studied since the inception of software agent research. The earliest matchmaker we are aware of is the ABSI (Agent-Based Software Interoperability) facilitator [140], which is based on the KQML specification and uses the KIF as the content language. The KIF expression is basically treated like Horn clauses. The matching between the advertisement and request expressed in KIF is the simple unification with the equality predicate.

Kuokka and Harada presented the SHADE and COINS systems for matchmaking [139]. The content language of COINS allows for free text and its matching algorithm utilizes the TF-IDF (Term Frequency-Inverse Document Frequency). The context language of the SHADE matchmaker consists of two parts: one is a subset of KIF, the other is a structured logic representation called MAX. MAX uses logic frames to declaratively store knowledge. The matchmaking algorithm used in SHADE is a Prolog-like unification process.

A more recent service broker-based information system is InfoSleuth [20]. The content language supported by InfoSleuth is KIF. The constraints for both the user request and the resource data are specified in terms of some given central ontology. It is the use of this common vocabulary that enables the dynamic matching of requests to the available resources. The advertisements specify agents’ capabilities in
terms of one or more ontologies. The constraint matching is an intersection function between the user query and the data resource constraints. If the conjunction of all the user constraints with all the resource constraints is satisfiable, then the resource contains data that are relevant to the user request.

In [39] Decker, Sycara, and Williamson presented matchmakers that store capability advertisements of different agents. They concentrated their efforts on architectures that support load balancing and protection of privacy of different agents. In [141], a matchmaking system called A-Match was described. A-Match is a Web based interface to the Matchmaker that allows human users to find agents that can provide needed services. Agents in the Matchmaker are represented on the bases of the inputs that they take and the outputs that they return. The Matchmaker matches the requirements of the user against the advertisements stored and it reports the list of agents whose advertisement matches the request of the user. The match performed by the Matchmaker is based on a taxonomy of terms. A-Match can store different types of advertisements coming from different applications. In [138, 142], Sycara et al. proposed an agent capability description language, called LARKS (Language for Advertisement and Request for Knowledge Sharing), that enables for advertising, requesting and matching agent capabilities. There are three types of matching in LARKS: exact match (the most accurate type of match), plug-in match (a less accurate but most useful type of match), and relaxed match (the least accurate type of match). The matching engine of the matchmaker agent in LARKS contains five different filters: context matching, profile comparison, similarity matching, signature matching, and constraint matching. The computational costs of these filters are in increasing order. Users may select any combination of these filters on demand. The matchmaking process using LARKS has a good trade-off between performance and quality of matching.

V. Subrahmanian, P. Bonatti, J. Dix, et al. introduced a HTML-like Service Description Language (SDL), which is used by the agents to describe their services [19]. By restricting their language, they are able to very clearly articulate what they mean by similar matches in terms of nearest neighbor and range queries, as well as provide very efficient algorithms to implement these operations. However, they do not address issues such as load balancing which is addressed by Decker,
Sycara, and Williamson.

A capability description language called CDL has been developed at the University of Edinburgh [143]. The syntax of CDL is KQML-like. Capabilities of and requests for services are described in CDL either in terms of achievable objectives or as performable actions. Logic-based reasoning over descriptions in CDL is based on the notion of capability subsumption or through instantiation. Capability subsumption in CDL refers to the question of whether a capability description can be used to solve a problem described by a given task description in CDL. Both the CDL and the proposed matching of CDL descriptions have been implemented in Java under the support of JAT (Java Agent Template, the old version of JATLite) (http://www.atai.ed.ac.uk/~oplan/cdl).

For a more comprehensive survey of matchmaking and brokering, see [144].

The above work dealt with many important issues in agent capability description and capability matchmaking. Almost all the work missed one point – the matchmaking is only based on the advertised capabilities of provider agents. The actual performance of provider agents in accomplishing delegated tasks is not considered at all. This problem also exists in current contract net systems.

Usually, there are more than one service provider agents claim that they have the same or very similar capabilities to accomplish a task in an application. For example, in financial investment applications one usually needs to predict the interest rate. There are different techniques for interest rate prediction such as neural network (NN) technique and fuzzy logic with genetic algorithm (FLGA) [51], but their prediction performances are different. If there are two interest rate prediction agents, one based on NN, the other based on FLGA, which one should be chosen to predict the interest rate? In such cases, current matchmaking algorithms can only choose one provider agent randomly. As the quality of service of different service provider agents varies from one agent to another, even though they claimed they have the same capabilities, it is obvious that it cannot meet the requirements of requester agents by randomly choosing one.

We propose algorithms that can pick up the provider agents based on the history information in accomplishing similar tasks in the past rather than choosing
randomly. The focus of this discussion is on how to consider agents’ actual performance based on the available capability description languages and capability matchmaking algorithms. The matchmaking algorithms used in the serving agent of the agent-based intelligent technique society are based on the `find.nn` and `range` algorithms in IMPACT [19, 55]. The thesaurus used in IMPACT is replaced by the financial ontology (see Chapter 5). For the convenience of further discussion, a brief introduction to these two algorithms is given below.

When an agent wants to find another agent providing a service, the serving agent must match the requested service with other service descriptions stored in its database, in order to find appropriate services. A service specification in IMPACT consists of (1) a service name in terms of a verb-noun(noun) expression such as calculate:rate(interest), (2) typed input and output variables, and (3) attributes of services (e.g., the cost for using the service). Agents may request services in one of the following forms:

- **k-Nearest Neighbor Request:** Find the k-nearest service names (pairs of verb and noun term) such that there exists an agent who provides that service and identify this agent;

- **d-Range Search:** Find all service names within a specified distance d.

Searching of appropriate services essentially relies on the exploitation of given weighted verb hierarchy and noun hierarchy, which are special cases of the general concept of a term hierarchy. Similarity between verbs and nouns in the verb and noun hierarchy, respectively, is computed via a given distance function on paths of weighted edges in the hierarchies. A composite distance function then combines both distance functions to calculate the combined similarity value for two word pairs (verb, noun) of service names. If a word cannot be found in the respective hierarchy a synonym will be searched in the ontology instead.

Suppose $\Sigma_v$ is a set of verbs, and $\Sigma_{nt}$ is a set of noun terms. A *noun term* is either a noun or an expression of the form $n_1(n_2)$, where $n_1$ and $n_2$ are both nouns. If $v \in \Sigma_v$ and $nt \in \Sigma_{nt}$, then $v : nt$ is called a *service name*.

Given a pair $(v, nt)$ specifying a desired service, the `find.nn` algorithm will
return a set of \( k \) agents that provide the most closely matching services. Closeness between \( \langle v, nt \rangle \) and another pair \( \langle v', nt' \rangle \) is determined by using the distance functions associated with the verb and noun-term hierarchies, together with a composite distance function \( cd \) specified by the agent invoking the \texttt{find.nn} algorithm. The algorithm uses the following internal data structures and/or subroutines:

- **Todo**: This is a list of verb/noun-term pairs, which are extended by their distances from the verb/noun-term pair that is requested in the initial or recursive call to the \texttt{find.nn} function. The list is maintained in increasing order of distance, and is not necessarily complete.

- **ANSTABLE**: This is a table consisting of at most \( k \) entries (\( k \) being the number of agents requested). At any given point in time during execution of the \texttt{find.nn} algorithm, \texttt{ANSTABLE} will contain the best answers found thus far, together with their distances from the requested service \( \langle v, nt \rangle \). \texttt{ANSTABLE} will be maintained in increasing order with respect to this distance.

- **search.service.table**: This function, given a verb/noun-term pair \( \langle V, NT \rangle \) and an integer \( k \), returns the set of all agents which provide the service \( \langle V : NT \rangle \); if their number exceeds \( k \), it returns \( k \) of them, which are deliberately chosen.

- **num.ans**: This function merely keeps track of the number of answers in \texttt{ANSTABLE}.

- **next_nbr**: This function takes as input the list \texttt{Todo} mentioned above and a pair \( \langle V, NT \rangle \). It returns as output the first member of the \texttt{Todo} list. If the \texttt{Todo} list is empty, it returns a special pair.

- **relax.ontology**: This function is called when either \( V \) or \( NT \) of the specified service name do not appear in the corresponding hierarchy. It returns a pair that is "similar" to \( \langle V, NT \rangle \) whose components do appear in the hierarchies. This function accesses the financial ontology (see Chapter 5) and the verb/noun-term hierarchies.

The algorithm is shown below:
CHAPTER 6. MATCHMAKING IN MIDDLE AGENTS

Algorithm find_nn(V:verb;NT:noun-term;k:integer)

/* Find the k agents offering the services closest to (V,NT), and output them with their */
/* distances; relax (V,NT) first, if it is not in the hierarchy. Output is either */
/* ANSTABLE (which contains a set of tuples of the form */
/* (agent name, service name, composite distance from (V,NT)) or ERROR */
1. create(Todo,V,NT);
2. ClosedList := NIL;
3. ANSTABLE := ∅;
4. if (V,NT) ∈ Σv × Σnt then
   5. { done := false;
   6. Sol := search_service_table(V,NT,k);
   7. while ¬done do
   8. { insert((V,NT),ClosedList);
   9. insert(Sol,ANSTABLE);
10. n := num_ans(ANSTABLE);
11. if n ≥ k then done := true
12. else
13. { (V',NT') := next_nbr(Todo);
14. if error(V',NT') = true then done := true
15. else
16. { (V,NT) := (V',NT');
17. Sol := search_service_table(V,NT,k − n); }
18. }
19. }
20. }
21. else /* search ontology */
22. { (V',NT') := relax_ontology(V,NT);
23. if error(V',NT') = true then return ERROR
24. else return find_nn(V',NT',k); }
25. return ANSTABLE;
end.

Suppose V ∈ Σv and NT ∈ Σnt. Also, let ΣvNT ⊆ Σv (ΣNT ⊆ Σnt) denote the set
of all verbs (noun-terms) in our hierarchies whose distance from V (NT) is finite.
Then in the worst case, \texttt{find.nn}(V, NT, k) will need $O(|\Sigma^V| \cdot |\Sigma^NT| + k)$ time. Note, however, that if there are $k$ services whose composite distance from $(V, NT)$ is finite, then one can obtain a tighter bound. Specifically, let $d_V$ ($d_{NT}$) be the maximum distance from a verb (noun-term) of one of these $k$ services to $V$ ($NT$). Furthermore, let $\Sigma^d_V \subseteq \Sigma^V$ ($\Sigma^{d_{NT}}_n \subseteq \Sigma^{NT}_n$) denote the set of all verbs (noun-terms) in the hierarchies whose distance from $V$ ($NT$) is less than or equal to $d_V$ ($d_{NT}$). Then in this case, \texttt{find.nn}(V, NT, k) will only need $O(|\Sigma^d_V| \cdot |\Sigma^{d_{NT}}_n| + k)$ time.

The range search algorithm below allows the serving agent to answer queries of the form “Find all agents that provide a service $vnt = (V', NT')$ which is within a distance $D$ of a requested service $vnt = (V, NT)$.”

In the range algorithm below, \textit{Todo} is a list of nodes to be processed, each of which is a service $vnt'$ extended by its distance from the service $vnt$. The algorithm has two steps:

- **The first step** is the while loop. It finds all pairs $vnt^* = (v^*, nt^*)$ that are within the specified distance $d$ from $vnt$. This step uses a procedure expand that behaves as follows: expand($vnt, vnt', d$) first computes the set

$$\{vnt^# | d' = cd(vnt^#, vnt) \leq d, vnt^# \in cr(vnt'), \langle vnt^#, d' \rangle \notin RelaxList\}.$$  

Here, RelaxList contains the services which have already been considered. Then, expand inserts the elements of this set into Todo. cr($vn, t$) is the candidate-relaxation of ($vn, t$) (refer to [5] (p. 61) for the definition of cr).

- **The second step** executes a “select” operation on the Service Table, finding all agents that offer any of the service names identified in the first step. As in the \texttt{find.nn} algorithm, if $V$ or $NT$ are not in the relevant verb or noun-term hierarchies, algorithm range calls the relax.ontology procedure specified in the \texttt{find.nn} algorithm to find a similar pair which belongs to them.

**Algorithm range(V:verb; NT:noun-term; D:real)**

/* Find all agents offering a service within distance $D$ to $(V, NT)$. */

/* Output is either an ANSTABLE or an ERROR */

1. if $D < 0$ then return ERROR;
2. if $(V, NT) \in \Sigma_v \times \Sigma_{nt}$ then
3. \{ $RelaxList := NIL$;
4. $Todo := \langle \langle V, NT \rangle, 0 \rangle$;
5. while $Todo \neq NIL$ do
6. \{ $(\langle V', NT' \rangle, D') :=$ first element of $Todo$;
7. insert $\langle \langle V', NT' \rangle, D' \rangle$ into $RelaxList$;
8. remove $\langle \langle V', NT' \rangle, D' \rangle$ from $Todo$;
9. $expand((V, NT), (V', NT'), D)$;
10. \}
11. return $\pi_{Agents,Dist}(RelaxList|Verb = V', NounTerm = NT'|ServiceTable)$
12. \}
13. else
14. \{ /* search ontology */
15. $\langle V', NT' \rangle := relax\_ontology(V, NT)$;
16. if $error(V', NT')$ then return $ERROR$
17. else return $range(V', NT', D - cd((V, NT), (V', NT')))$;
18. \}
end.

Suppose $V \in \Sigma_v$ and $NT \in \Sigma_{nt}$. Also, let $vnt = \{(v, nt)|v \in \Sigma_v \land nt \in \Sigma_{nt} \land cd((v, nt), (V, NT)) \leq D\}$ and let $S$ be the set of all (agent name, $(v, nt)$) pairs where $(v, nt) \in vnt$. Then $range(V, NT, D)$ will need $O(|vnt| + |S|)$ time.

6.3 Considering Agents’ Track Records in Matchmaking

The algorithms discussed above can only find the agents that they claimed to offer the services closest to the services requested. The “best” agent in $find\_nn$ or $range$ algorithm only means that the service name advertised by this agent is closest to the service name requested, but has nothing to do with the actual performance of the agent in accomplishing a task. Here, we believe that the service providers make the binding commitment to perform the corresponding tasks (provide the corresponding services) when they report the availabilities of their services to the
matchmaker. That is, if an agent says it has a capability then it can perform the
tasks corresponding to the capability and will do so when asked. If one really
delegates the same task to different agents with same or similar capabilities, the
quality of service \((Q \circ S)\) may vary from agent to agent. Some agents provide very
good service, demonstrating an expert standard; some only show a novice level.
How can one choose the agents (service providers) that not only claimed to offer
the services but also did well in practice? We propose one solution to this problem
— taking into account agents’ track records in matchmaking. As the agents’ “credit
histories” are considered during the selection process, the selected agent is more
appropriate for the task than the agent chosen by directly using find.nn or range
algorithms. Before presenting the matchmaking algorithm that considers agents’
track records, the representation of track records is discussed.

6.3.1 The Representation of Track Records

In the previous section, it was mentioned that the specification of a single service
consists of four components: service name, inputs, outputs, and attributes. Here
one more component (track records) was added to the specification.

What should be put in the track records? The requester’s evaluation for the
service provided by the selected agent will be put in the track record field of the
agent’s service specification. The requester’s evaluation is actually the satisfactory
degree for the service received, just as when a person goes to a restaurant, and tips
the waiter based on the service received. Here we assume that the requester agent
can give an overall evaluation for the service it received based on a set of criteria.
The following linguistic values are allowed for the requester to describe its overall
evaluation (satisfactory degree): strong satisfaction, satisfaction, weak satisfaction,
neutral, weak unsatisfaction, unsatisfaction, and strong unsatisfaction. The track
records consist of 2 – tuples with a form as \([n^{th} \text{ time service, evaluation}]\). The
first parameter in the 2 – tuple is the ordinal of the service provided, the second
is the satisfactory degree returned by the agent received the service. For example,
if an agent is delegated a task, and this is the third time, the agent delegated the
task deems the service is excellent, then a 2 – tuple, \([3, \text{strong satisfaction}]\), will
be added to the track records of the agent providing the service, which is in the
database of the serving agent.

Such a representation has an extra advantage – it can keep track of an agent’s aging. For example, if an agent’s track record indicates that this agent received very good evaluations from the requesters with a small service ordinal number, and received bad evaluations with larger service ordinal numbers, this means the agent is aged. Its knowledge is out of date. When trying to choose an agent to accomplish a task, the matchmaker should be very cautious with aged agents.

6.3.2 The Accumulation of Track Records

To accumulate track records, we provide a special module for requester agents to let them give evaluation results for the services they received in the prototype. This module can also assemble the evaluation results in the KQML messages, which will be sent to middle agents by requester agents. When middle agents receive the messages, they interpret them and extract the evaluation result from the KQML message. The value (evaluation result) is assigned to the track record field of the corresponding agent, and then stored into the database of the middle agents.

6.3.3 The Generation of Initial Values

The track records or “credit histories” of agents are accumulated gradually during the executing process of a multi-agent system. Thus there are no track records available when the system is first launched. As discussed in the previous section, in such cases one can only randomly choose one agent with the requested capability from the returned results of find.nn algorithm.

If one can find some way to provide reasonable initial values for the track records, one can choose the agents based on the initial values at the very beginning. With the track records accumulated, the initial values and the track records are combined to choose the agents. In this way, the shortcoming of the algorithm considering track records – no track records being available at the very beginning – is overcome.

This section discusses one way to provide initial values for the track records. It is to give a set of problems where the answers are already known (called benchmark
problems), and ask all the agents who claimed that they have the capabilities to solve these problems. By calculating the "distances" between the solutions to the problems the agents gave and the benchmark results, one can determine the initial values for the track records of these agents.

The basic idea of this approach is as follows. Before putting the system into practical operation, the system is "trained" with a set of benchmark problems. That is, the middle agent is run with the find.nn or range algorithm first. The middle agent then asks the agents with the same or similar capabilities (based on the results in ANSTABLE) to solve the benchmark problems. By testing the results of these agents against the benchmarks, one obtains an evaluation of these agents for their performance on solving the benchmark problems. This evaluation is then used as the initial value of these agents' track records.

Formally, let $A = \{A_1, A_2, \ldots, A_k\}$ be the agent set with the same or similar capabilities. We use $S = \{S_1, S_2, \ldots, S_m\}$ to denote the problem set. Each problem $S_i \in S$ ($i = 1, 2, \ldots, m$) has a related attribute set $a = \{a_1, a_2, \ldots, a_n\}$. We say agent $A_i$ solved problem $S_j$ if it returns the values of the $n$ attributes related to the problem. The values can be numeric or non-numeric (linguistic values). Suppose the benchmark values of these attributes denoted by $B_i = \{b_{i1}, b_{i2}, \ldots, b_{im}\}$ ($i = 1, 2, \ldots, m$) and, the values returned by agent $A_j$ denoted by $B_{ij} = \{b_{i1}^{A_j}, b_{i2}^{A_j}, \ldots, b_{im}^{A_j}\}$ ($i = 1, 2, \ldots, m, j = 1, 2, \ldots, k$). The description of the benchmark problems is then summarized in Table 6.1.

The next step in the initial value generation process is to calculate the "distances" between the returned values by agent $A_j$ and the benchmark values.
Table 6.2: Mapping Results between Distance and Satisfactory Degree

<table>
<thead>
<tr>
<th>Distance Range</th>
<th>Satisfactory Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 0.143</td>
<td>strong satisfaction</td>
</tr>
<tr>
<td>0.143 to 0.286</td>
<td>satisfaction</td>
</tr>
<tr>
<td>0.286 to 0.429</td>
<td>weak satisfaction</td>
</tr>
<tr>
<td>0.429 to 0.572</td>
<td>neutral</td>
</tr>
<tr>
<td>0.572 to 0.715</td>
<td>weak unsatisfaction</td>
</tr>
<tr>
<td>0.715 to 0.858</td>
<td>unsatisfaction</td>
</tr>
<tr>
<td>0.858 to 1.0</td>
<td>strong unsatisfaction</td>
</tr>
</tbody>
</table>

are many definitions of "distance". Here the distance is defined in terms of standard Euclidean distance. The distance between $B_i$ and $D_i^{A_j}$ is defined to be $d_j$, where

$$d_j = \sqrt{\sum_{r=1}^{n} (b_{ir} - b_{ir}^{A_j})^2}.$$

Then these distances are added to the database of the middle agent as the initial values of the track records.

Considering that the initial values and the track records need to be combined when accumulated, the distances were mapped to the satisfactory degrees defined in Section 6.3.1. As there are 7 levels of satisfactory degrees, each level accounts for 1/7 of the distance range. Therefore, if the distance is between 0 and 0.143, [0, strong satisfaction] will be added to the agent's track record; If the distance is between 0.143 and 0.286, [0, satisfaction] will be added etc. The mapping results are shown in Table 6.2.

6.3.4 The Use of Track Records

Based on the representation of track records and the accumulated track records (including the initial values of track records), the matchmaking algorithm that can consider agents' track records are now ready to be presented.

This algorithm is based on the returned results, ANSTABLE, of find_nn or range algorithm, and do the following processing:

- For each agent $A_j \in ANSTABLE$, let $A_j$ solve the benchmark problems in
• Calculate the distances between the returned values $B_i^{A_j}$ and the benchmark values $B_i$, and add the corresponding 2-tuple $[0,\text{ satisfactory degree}]$ to $A_j$'s track records.

• Sum up the numbers of different satisfactory degree ($\text{strong satisfaction, satisfaction, etc.}$) for each agent in $\text{ANSTABLE}$. The result for each agent is a vector, $EV_j (j = 1, \ldots, k = |\text{ANSTABLE}|)$, $EV_j = ([\text{strong satisfaction}, n_{1j}], [\text{satisfaction}, n_{2j}], [\text{weak satisfaction}, n_{3j}], [\text{neutral}, n_{4j}], [\text{weak unsatisfaction}, n_{5j}], [\text{unsatisfaction}, n_{6j}], [\text{strong unsatisfaction}, n_{7j}])^t$, $n_{ij} \geq 0 (i = 1, \ldots, 7; j = 1, \ldots, k)$.

• Construct agents' evaluation matrix $M_{7 \times k}$. The matrix looks like as follows:

$$
M = \begin{bmatrix}
\text{strong satisfaction} \\
\text{satisfaction} \\
\vdots \\
\text{strong unsatisfaction}
\end{bmatrix}
\begin{bmatrix}
A_1 & A_2 & \ldots & A_k \\
\begin{bmatrix} n_{11} & n_{12} & \ldots & n_{1k} \\
n_{21} & n_{22} & \ldots & n_{2k} \\
\vdots & \vdots & \vdots & \vdots \\
n_{71} & n_{72} & \ldots & n_{7k}
\end{bmatrix}
\end{bmatrix}
$$

where $A_i (i = 1, \ldots, k)$ are agent names in the returned results of $\text{find.nn}$ or $\text{range}$ algorithm, i.e., they have the same or closely similar capabilities.

• Select the most promising agent to provide the requested service based on the evaluation matrix $M$.

In the last item above, different criteria may be used to describe most promising. Two reasonable approaches we tested are given below.

The first one is based on a collection of heuristic rules. These rules are derived from typical international conference paper choosing procedures in practice. Some example rules are listed below:

• **Rule 1:** If $n_{1i}$ (the number of strong satisfaction for agent $A_i$) is the largest among $n_{1j} (j = 1, \ldots, k)$ and $n_{5j} = n_{6j} = n_{7j} = 0$ then choose $A_i$;
• **Rule 2:** If all \( n_{ij} \) \((j = 1, \ldots, k)\) are equal and \( n_{2i} \) (the number of *satisfaction* for agent \( A_i \)) is the largest among \( n_{2j} \) \((j = 1, \ldots, k)\) and \( n_{6j} = n_{6j} = n_{7j} = 0 \)

then choose \( A_i \);

• **Rule 3:** If all \( n_{ij} = 0 \) except \( n_{4i} \), then randomly choose one agent, \( A_i \);

• ...

The alternative is to map each satisfactory degree (strong satisfaction, satisfaction, etc.) to a weighting value. Then the total score of each agent is calculated and the agent with the highest score is chosen. If there is more than one agent with the same highest score, then one is randomly chosen to accomplish the delegated task. For example, one can map an element in the evaluation vector to a value in \([-1, 1]\).

The weight for strong satisfaction is 1, for satisfaction is 2/3, for weak satisfaction is 1/3, for neutral is 0, for weak unsatisfaction is -1/3, for unsatisfaction is -2/3, and for strong unsatisfaction is -1. The second approach is easier to implement than the first one.

The algorithm uses the following variables and subroutines:

• **initial_value_generation:** This subroutine takes ANSTABLE and benchmark problem set \( S \) as inputs, and generates the initial values of track records for all agents in ANSTABLE as outputs. The initial values are stored in the database of middle agents;

• **\( TR_j \):** The track record of agent \( A_j \in ANSTABLE \). It consists of \( 2 - tuples \) like \([5, satisfaction]\);

• **aging_check:** This subroutine checks the aging status of agent \( A_j \) based on its track record, \( TR_j \). It will mark the agent "aged" if this agent satisfies the pre-set aged conditions;

• **take_one_tuple:** This function takes one tuple from \( TR_j \), and assigns it to variable \( tr \);

• **evaluation_part:** This function returns the satisfactory degree part of \( 2 - tuple, tr \), and assigns the corresponding value (strong satisfaction, satisfaction, etc.) to variable \( sd \);
• **sum_up**: This function counts the numbers of different satisfactory degrees, \( n_{ij} \) \( (i = 1, \ldots, 7; j = 1, \ldots, k) \), and adds \([\text{strong satisfaction}, n_{1j}]\), \([\text{satisfaction}, n_{2j}]\) etc. to \( EV_j \) \( (j = 1, \ldots, k) \). It is actually a case statement;

• **construct_evaluation_matrix**: This function constructs the evaluation matrix based on the evaluation vectors, \( EV_j \) \( (j = 1, \ldots, k) \);

• **most.promise**: This function chooses the best agent, \( FINAL.ANS \), using one of the approaches mentioned above. Here, \( FINAL.ANS \) is the best agent based its advertised capabilities and its actual performance to accomplish delegated tasks.

As a summary, the algorithm that considers agents’ track records is given as follows:

```plaintext
Algorithm find_most.promise(V, NT, k, D)

/* Find the most promising agent based on the agents’ track records and */
/* the returned results of find.nn or range algorithm. */
1. ANSTABLE := find.nn(V, NT, k);
2. if ANSTABLE = \emptyset then
3. ANSTABLE := range(V, NT, D);
4. if ANSTABLE = \emptyset then
5. return NO.ANSWER
6. else
7. For all agents in ANSTABLE, call initial_value_generation
8. { for \( j := 1 \) to \( k = |ANSTABLE| \) do
9. { \( TR_j := A_j \)'s track records;
10. aging_check(\( TR_j \));
11. \textbf{while} \( TR_j \) \( \neq \) NIL \textbf{do}
12. { \( tr := \text{take.one.tuple}(TR_j); \)
13. \( sd := \text{evaluation.part}(tr); \)
14. \( EV_j = \text{sum.up}(sd); \)
15. remove \( tr \) from \( TR_j; \) } }
16. \( M = \text{construct.evaluation.matrix}(EV_j); \)
```
17. \texttt{FINAL.ANS := most.promise}(M);

18. \textbf{return} \texttt{FINAL.ANS}; }

\textbf{end.}

Suppose the maximum number of elements (2 - tuples) in \( TR_j \) is \( m \), then the time complexity from line 7 to line 15 is \( O(mk) \). Both lines 16 and 17 will need \( O(7k) \) time. In the worst case, \texttt{find.nn}(V, NT, k) will need \( O(|\Sigma_v^V| \cdot |\Sigma_{nt}^{NT}| + k) \) time [19]. Thus in the worst case, \texttt{find.most.promise} will need \( O(|\Sigma_v^V| \cdot |\Sigma_{nt}^{NT}| + (15 + m)k) \) time (when calling \texttt{find.nn}). As \( 15 + m \) is a small constant, the time complexity of \texttt{find.most.promise} only increases a little compared with that of \texttt{find.nn}.

With \texttt{find.most.promise} algorithm, if one service provider agent keeps performing well, the algorithm always returns that agent as its matchmaking result. It is natural. Just as in human society, if someone always does his work well, there is no reason to terminate him.

\subsection{Example and Experimental Results}

In [19], Subrahmanian et al. give very detailed performance evaluations of the \texttt{find.nn} and \texttt{range} algorithms by considering the efficiency of finding similar services and the quality of the \textit{matching} services provided as the output. Their experimental results show that these two algorithms are very efficient.

From the time complexity analyses of the algorithms it is apparent that the time complexity of \texttt{find.most.promise} algorithm did not increase much compared with that of \texttt{find.nn}. Thus \texttt{find.most.promise} is also efficient.

\subsubsection{Impact of Track Records on Matchmaking}

To evaluate the impact of different track records on the final matchmaking results, we conducted some simulations. All the data here were generated by a special designed C program calling the \texttt{rand()} function.

Suppose the returned results of \texttt{find.nn} have 10 agents,

\[ \text{ANSTABLE} = [A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}] \]
Table 6.3: Track Records with \( n = 40 \)
(s-s: strong satisfaction, s: satisfaction, w-s: weak satisfaction, n: neutral
w-u: weak unsatisfaction, u: unsatisfaction, s-u: strong unsatisfaction)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Track Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_1 )</td>
<td>([1, s-u], [2, s-u])</td>
</tr>
<tr>
<td>( A_2 )</td>
<td>([1, s-u], [2, s-u], [3, s-u], [4, s-u], [5, w-s], [6, w-s])</td>
</tr>
<tr>
<td>( A_3 )</td>
<td>([1, u], [2, w-u], [3, u], [4, u], [5, w-s])</td>
</tr>
<tr>
<td>( A_4 )</td>
<td>([1, s], [2, n])</td>
</tr>
<tr>
<td>( A_5 )</td>
<td>([1, s-u], [2, s], [3, n], [4, u], [5, w-s], [6, u])</td>
</tr>
<tr>
<td>( A_6 )</td>
<td>([1, s], [2, s], [3, s-u], [4, u])</td>
</tr>
<tr>
<td>( A_7 )</td>
<td>([1, s], [2, w-u], [3, u])</td>
</tr>
<tr>
<td>( A_8 )</td>
<td>([1, w-s], [2, n], [3, w-u], [4, s-u])</td>
</tr>
<tr>
<td>( A_9 )</td>
<td>([1, w-u], [2, u], [3, s], [4, s-s])</td>
</tr>
<tr>
<td>( A_{10} )</td>
<td>([1, s-u], [2, s], [3, w-s], [4, s-u])</td>
</tr>
</tbody>
</table>

That means these 10 agents have the same or closely similar capabilities. Assume there are \( n \) (similar) tasks. For each task, we randomly delegate to an agent, \( A_4 \), from \textit{ANSTABLE}, and randomly attach an evaluation result (satisfactory degree) to \( A_i \). The track records of these 10 agents with \( n = 40 \) are shown in Table 6.3.

The corresponding evaluation matrix is as follows:

\[
M = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 2 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 3 & 0 & 2 & 1 & 1 & 0 & 1 & 0 \\
2 & 4 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 2
\end{pmatrix}
\]

Using the second approach (mapping each satisfactory degree to a weighting value in \([-1, 1]\)), we can calculate the total score of each agent in \textit{ANSTABLE}. For example, agent \( A_9 \) has 4 items in its track records when \( n = 40 \). The evaluation (satisfactory degree) for its first delegated task is \textit{weak unsatisfaction} (the mapped weight is \(-1/3\)), the second delegated task is \textit{unsatisfaction} (corresponding weight is \(-2/3\)), the third is \textit{satisfaction} (corresponding weight is \(2/3\)), and the fourth is
CHAPTER 6. MATCHMAKING IN MIDDLE AGENTS

Table 6.4: Agents’ Scores with Different Track Records

<table>
<thead>
<tr>
<th>n</th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
<th>$A_7$</th>
<th>$A_8$</th>
<th>$A_9$</th>
<th>$A_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>-1.0</td>
<td>-4.0</td>
<td>-1.0</td>
<td>0.7</td>
<td>-1.0</td>
<td>1.7</td>
<td>0.7</td>
<td>0.3</td>
<td>-0.3</td>
<td>-0.3</td>
</tr>
<tr>
<td>40</td>
<td>-2.0</td>
<td>-3.3</td>
<td>-2.0</td>
<td>0.7</td>
<td>-1.3</td>
<td>0.0</td>
<td>-0.3</td>
<td>-1.0</td>
<td>0.7</td>
<td>-1.0</td>
</tr>
<tr>
<td>60</td>
<td>-2.0</td>
<td>-4.3</td>
<td>-1.7</td>
<td>0.7</td>
<td>-2.0</td>
<td>-0.3</td>
<td>-1.3</td>
<td>-1.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>80</td>
<td>-2.3</td>
<td>-4.3</td>
<td>-2.0</td>
<td>0.3</td>
<td>-2.3</td>
<td>0.0</td>
<td>0.7</td>
<td>-1.0</td>
<td>1.0</td>
<td>0.7</td>
</tr>
<tr>
<td>100</td>
<td>-2.3</td>
<td>-5.3</td>
<td>-3.3</td>
<td>2.0</td>
<td>-1.7</td>
<td>-0.3</td>
<td>0.0</td>
<td>-1.0</td>
<td>0.7</td>
<td>-0.3</td>
</tr>
<tr>
<td>200</td>
<td>1.7</td>
<td>-2.3</td>
<td>1.0</td>
<td>0.0</td>
<td>-5.3</td>
<td>-3.0</td>
<td>0.7</td>
<td>-0.3</td>
<td>-0.3</td>
<td>-3.0</td>
</tr>
<tr>
<td>300</td>
<td>0.7</td>
<td>-0.3</td>
<td>-1.7</td>
<td>-3.3</td>
<td>-7.3</td>
<td>-2.0</td>
<td>2.3</td>
<td>-3.3</td>
<td>2.0</td>
<td>-7.7</td>
</tr>
</tbody>
</table>

Strong satisfaction (corresponding weight is 1). Thus the total score for $A_9$ with $n = 40$ is

$$1 \times 1 + 1 \times (2/3) + 1 \times (-1/3) + 1 \times (-2/3) \approx 0.7.$$  

The higher the total score, the better the actual performance. Table 6.4 summarizes the results when task number $n$ is 20, 40, 60, 80, 100, 200, and 300, respectively.

Follow "choosing the agent with highest score" principle, the selected agents are $A_6$ ($n = 20$), $A_4$ ($n = 40$), $A_9$ ($n = 60$), $A_9$ ($n = 80$), $A_4$ ($n = 100$), $A_1$ ($n = 200$), and $A_7$ ($n = 300$). If one did not consider agents' track records, the selected agent is always $A_1$. The simulation results indicate that the selected agent is different based on different track records. This shows that agents’ track records have a strong impact on the outcome of matchmaking. It is best to consider agent's track records whenever possible.

### 6.4.2 Example for Initial Value Generation

In the case study of Chapter 4, a soft computing (SC) agent was used to predict the interest rate, which is based on feed-forward neural network. Actually one can have other SC agents for interest rate prediction based on different models (e.g., fuzzy logic and genetic algorithm model [51]). In this section, two SC agents are taken for interest rate prediction (one is based on neural network, called *SC_agent_NN*, the other based on fuzzy logic and genetic algorithm, called *SC_agent_FLGA*) as an example to show how the *initial_value_generation* algorithm works and test the predictive capabilities of these two SC agents.
Construction of Benchmark Problems

When predicting the interest rate (as represented by 91-day Treasury bill rates), both of the agents take the changes of previous Treasury-bill (T-bill) rates, real gross national product (GNP), consumer price index (CPI), M2 money supply, and personal wealth (W) as inputs. Personal wealth is the accumulation of the difference between personal income and personal consumption. The M2 money supply consists of all cash in circulation and deposits in savings and check accounts, and represents readily available liquid assets. The consumer price index is a measure of the inflation trend. The outputs are the changes of next T-bill rates (predicted interest rates). Quarterly data are used.

We use the history financial data of the five factors provided in Appendix B of [51], which we list here from Table 6.5 to Table 6.9, to construct the required benchmark problems.

Table 6.5: Change in Consumer Price Index

<table>
<thead>
<tr>
<th>Year</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr 4</th>
<th>Year</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
<td>0.30</td>
<td>0.30</td>
<td>0.30</td>
<td>0.30</td>
<td>1977</td>
<td>1.00</td>
<td>1.30</td>
<td>0.50</td>
<td>0.70</td>
</tr>
<tr>
<td>1967</td>
<td>0.00</td>
<td>0.30</td>
<td>0.30</td>
<td>0.30</td>
<td>1978</td>
<td>1.00</td>
<td>1.60</td>
<td>1.60</td>
<td>1.30</td>
</tr>
<tr>
<td>1968</td>
<td>0.40</td>
<td>0.30</td>
<td>0.50</td>
<td>0.40</td>
<td>1979</td>
<td>1.70</td>
<td>2.40</td>
<td>2.30</td>
<td>2.10</td>
</tr>
<tr>
<td>1969</td>
<td>0.40</td>
<td>0.60</td>
<td>0.60</td>
<td>0.50</td>
<td>1980</td>
<td>3.00</td>
<td>2.90</td>
<td>1.50</td>
<td>2.20</td>
</tr>
<tr>
<td>1970</td>
<td>0.50</td>
<td>0.60</td>
<td>0.50</td>
<td>0.50</td>
<td>1981</td>
<td>2.30</td>
<td>2.00</td>
<td>2.60</td>
<td>1.30</td>
</tr>
<tr>
<td>1971</td>
<td>0.30</td>
<td>0.40</td>
<td>0.50</td>
<td>0.20</td>
<td>1982</td>
<td>0.80</td>
<td>1.40</td>
<td>1.80</td>
<td>0.20</td>
</tr>
<tr>
<td>1972</td>
<td>0.30</td>
<td>0.30</td>
<td>0.40</td>
<td>0.40</td>
<td>1983</td>
<td>0.00</td>
<td>1.20</td>
<td>1.20</td>
<td>0.90</td>
</tr>
<tr>
<td>1973</td>
<td>0.50</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1984</td>
<td>1.10</td>
<td>1.10</td>
<td>1.10</td>
<td>0.80</td>
</tr>
<tr>
<td>1974</td>
<td>1.30</td>
<td>1.30</td>
<td>1.50</td>
<td>1.50</td>
<td>1985</td>
<td>0.70</td>
<td>1.30</td>
<td>0.70</td>
<td>1.00</td>
</tr>
<tr>
<td>1975</td>
<td>0.90</td>
<td>0.80</td>
<td>1.20</td>
<td>0.80</td>
<td>1986</td>
<td>0.20</td>
<td>-0.20</td>
<td>0.80</td>
<td>0.60</td>
</tr>
<tr>
<td>1976</td>
<td>0.60</td>
<td>0.70</td>
<td>0.90</td>
<td>0.60</td>
<td>1987</td>
<td>1.20</td>
<td>1.50</td>
<td>1.30</td>
<td>1.00</td>
</tr>
</tbody>
</table>

There is some evidence to suggest that fundamental financial market characteristics change over a period of four to five years [145]. That is, the market "forgets" the influence of data that is more than five years old. For this reason, five-year data windows are used. 15 data windows are examined, each starting in the first quarter of the years 1967 through to 1981, respectively. The ending quarters for each data window will be the fourth quarters of the years 1971 through to 1985.
### Table 6.6: Change in Gross National Product in 1982 US Dollars (Billions)

<table>
<thead>
<tr>
<th>Year</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr 4</th>
<th>Year</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
<td>42.50</td>
<td>5.70</td>
<td>22.50</td>
<td>10.90</td>
<td>1977</td>
<td>39.20</td>
<td>46.70</td>
<td>59.10</td>
<td>-7.70</td>
</tr>
<tr>
<td>1968</td>
<td>26.70</td>
<td>39.60</td>
<td>18.40</td>
<td>-2.30</td>
<td>1979</td>
<td>0.10</td>
<td>-3.00</td>
<td>28.70</td>
<td>-6.10</td>
</tr>
<tr>
<td>1969</td>
<td>33.50</td>
<td>3.30</td>
<td>13.40</td>
<td>-9.70</td>
<td>1980</td>
<td>32.10</td>
<td>-76.40</td>
<td>2.10</td>
<td>40.10</td>
</tr>
<tr>
<td>1971</td>
<td>64.80</td>
<td>-0.20</td>
<td>12.70</td>
<td>-0.10</td>
<td>1982</td>
<td>-48.60</td>
<td>9.50</td>
<td>-25.40</td>
<td>4.80</td>
</tr>
<tr>
<td>1972</td>
<td>54.60</td>
<td>49.50</td>
<td>27.00</td>
<td>49.20</td>
<td>1983</td>
<td>27.30</td>
<td>71.70</td>
<td>48.10</td>
<td>58.70</td>
</tr>
<tr>
<td>1973</td>
<td>62.70</td>
<td>7.00</td>
<td>-2.70</td>
<td>24.50</td>
<td>1984</td>
<td>86.60</td>
<td>46.30</td>
<td>22.60</td>
<td>14.60</td>
</tr>
<tr>
<td>1974</td>
<td>-15.40</td>
<td>7.80</td>
<td>-35.90</td>
<td>-23.90</td>
<td>1985</td>
<td>42.30</td>
<td>21.70</td>
<td>36.60</td>
<td>26.60</td>
</tr>
<tr>
<td>1975</td>
<td>-52.70</td>
<td>26.90</td>
<td>45.30</td>
<td>37.80</td>
<td>1986</td>
<td>58.70</td>
<td>-16.50</td>
<td>7.80</td>
<td>21.20</td>
</tr>
<tr>
<td>1976</td>
<td>51.70</td>
<td>12.50</td>
<td>11.70</td>
<td>28.20</td>
<td>1987</td>
<td>49.40</td>
<td>40.50</td>
<td>49.30</td>
<td>62.80</td>
</tr>
</tbody>
</table>

### Table 6.7: Change in M2 Money Supply in 1982 US Dollars (Billions)

<table>
<thead>
<tr>
<th>Year</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr 4</th>
<th>Year</th>
<th>Qtr1</th>
<th>Qtr2</th>
<th>Qtr3</th>
<th>Qtr 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
<td>13.40</td>
<td>1.00</td>
<td>-2.70</td>
<td>2.10</td>
<td>1977</td>
<td>30.00</td>
<td>16.80</td>
<td>20.00</td>
<td>16.10</td>
</tr>
<tr>
<td>1967</td>
<td>19.40</td>
<td>24.20</td>
<td>28.50</td>
<td>16.20</td>
<td>1978</td>
<td>0.50</td>
<td>-7.50</td>
<td>-6.90</td>
<td>-3.40</td>
</tr>
<tr>
<td>1971</td>
<td>32.50</td>
<td>47.80</td>
<td>28.10</td>
<td>37.00</td>
<td>1982</td>
<td>23.90</td>
<td>4.30</td>
<td>14.20</td>
<td>37.90</td>
</tr>
<tr>
<td>1972</td>
<td>35.40</td>
<td>33.50</td>
<td>42.40</td>
<td>40.40</td>
<td>1983</td>
<td>95.00</td>
<td>30.30</td>
<td>19.00</td>
<td>20.70</td>
</tr>
<tr>
<td>1973</td>
<td>16.90</td>
<td>-7.40</td>
<td>-6.80</td>
<td>-23.00</td>
<td>1984</td>
<td>10.00</td>
<td>17.30</td>
<td>14.80</td>
<td>29.30</td>
</tr>
<tr>
<td>1974</td>
<td>-17.00</td>
<td>-24.10</td>
<td>-30.00</td>
<td>-31.20</td>
<td>1985</td>
<td>45.10</td>
<td>9.90</td>
<td>41.20</td>
<td>14.10</td>
</tr>
<tr>
<td>1975</td>
<td>-1.70</td>
<td>41.60</td>
<td>24.90</td>
<td>13.10</td>
<td>1986</td>
<td>20.30</td>
<td>66.80</td>
<td>50.50</td>
<td>39.80</td>
</tr>
<tr>
<td>1976</td>
<td>35.80</td>
<td>39.20</td>
<td>19.70</td>
<td>42.60</td>
<td>1987</td>
<td>5.50</td>
<td>-15.50</td>
<td>-1.30</td>
<td>7.20</td>
</tr>
</tbody>
</table>
### Table 6.8: Personal Wealth in 1982 US Dollars (Billions)

<table>
<thead>
<tr>
<th>Year</th>
<th>Qtr 1</th>
<th>Qtr 2</th>
<th>Qtr 3</th>
<th>Qtr 4</th>
<th>Year</th>
<th>Qtr 1</th>
<th>Qtr 2</th>
<th>Qtr 3</th>
<th>Qtr 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
<td>318.20</td>
<td>332.60</td>
<td>339.10</td>
<td>356.00</td>
<td>1977</td>
<td>507.00</td>
<td>518.80</td>
<td>537.30</td>
<td>537.10</td>
</tr>
<tr>
<td>1967</td>
<td>369.70</td>
<td>366.40</td>
<td>378.50</td>
<td>383.20</td>
<td>1978</td>
<td>553.80</td>
<td>560.10</td>
<td>578.80</td>
<td>593.10</td>
</tr>
<tr>
<td>1968</td>
<td>381.20</td>
<td>394.20</td>
<td>394.20</td>
<td>407.20</td>
<td>1979</td>
<td>601.60</td>
<td>598.10</td>
<td>597.80</td>
<td>593.60</td>
</tr>
<tr>
<td>1969</td>
<td>406.80</td>
<td>420.40</td>
<td>430.20</td>
<td>442.20</td>
<td>1980</td>
<td>597.60</td>
<td>600.70</td>
<td>603.60</td>
<td>625.90</td>
</tr>
<tr>
<td>1970</td>
<td>434.20</td>
<td>456.40</td>
<td>449.90</td>
<td>448.30</td>
<td>1981</td>
<td>626.30</td>
<td>627.30</td>
<td>655.10</td>
<td>650.20</td>
</tr>
<tr>
<td>1971</td>
<td>441.80</td>
<td>456.70</td>
<td>454.30</td>
<td>451.40</td>
<td>1982</td>
<td>629.20</td>
<td>638.20</td>
<td>614.10</td>
<td>601.40</td>
</tr>
<tr>
<td>1972</td>
<td>469.40</td>
<td>462.10</td>
<td>476.20</td>
<td>511.90</td>
<td>1983</td>
<td>591.80</td>
<td>578.90</td>
<td>558.60</td>
<td>592.20</td>
</tr>
<tr>
<td>1973</td>
<td>506.30</td>
<td>522.30</td>
<td>536.70</td>
<td>569.50</td>
<td>1984</td>
<td>630.60</td>
<td>611.00</td>
<td>632.80</td>
<td>635.90</td>
</tr>
<tr>
<td>1974</td>
<td>548.30</td>
<td>523.90</td>
<td>528.20</td>
<td>543.50</td>
<td>1985</td>
<td>642.80</td>
<td>637.00</td>
<td>597.90</td>
<td>618.50</td>
</tr>
<tr>
<td>1975</td>
<td>507.90</td>
<td>506.30</td>
<td>507.30</td>
<td>514.40</td>
<td>1986</td>
<td>632.50</td>
<td>651.60</td>
<td>608.80</td>
<td>620.70</td>
</tr>
<tr>
<td>1976</td>
<td>509.90</td>
<td>516.40</td>
<td>517.80</td>
<td>512.80</td>
<td>1987</td>
<td>633.60</td>
<td>614.80</td>
<td>604.80</td>
<td>678.30</td>
</tr>
</tbody>
</table>

### Table 6.9: Two-Quarter Moving Average Change in T-Bill Discount Rate

<table>
<thead>
<tr>
<th>Year</th>
<th>Qtr 1</th>
<th>Qtr 2</th>
<th>Qtr 3</th>
<th>Qtr 4</th>
<th>Year</th>
<th>Qtr 1</th>
<th>Qtr 2</th>
<th>Qtr 3</th>
<th>Qtr 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1966</td>
<td>0.39</td>
<td>0.22</td>
<td>0.21</td>
<td>0.33</td>
<td>1977</td>
<td>-0.28</td>
<td>0.07</td>
<td>0.42</td>
<td>0.65</td>
</tr>
<tr>
<td>1967</td>
<td>-0.26</td>
<td>-0.79</td>
<td>-0.09</td>
<td>0.56</td>
<td>1978</td>
<td>0.47</td>
<td>0.17</td>
<td>0.46</td>
<td>1.10</td>
</tr>
<tr>
<td>1968</td>
<td>0.36</td>
<td>0.36</td>
<td>0.09</td>
<td>0.03</td>
<td>1979</td>
<td>1.02</td>
<td>0.34</td>
<td>0.14</td>
<td>1.22</td>
</tr>
<tr>
<td>1969</td>
<td>0.45</td>
<td>0.33</td>
<td>0.46</td>
<td>0.54</td>
<td>1980</td>
<td>1.91</td>
<td>-0.88</td>
<td>-2.11</td>
<td>1.83</td>
</tr>
<tr>
<td>1970</td>
<td>0.11</td>
<td>-0.29</td>
<td>-0.44</td>
<td>-0.69</td>
<td>1981</td>
<td>2.57</td>
<td>0.56</td>
<td>0.36</td>
<td>-1.40</td>
</tr>
<tr>
<td>1971</td>
<td>-1.26</td>
<td>-0.58</td>
<td>0.60</td>
<td>0.01</td>
<td>1982</td>
<td>-1.10</td>
<td>0.17</td>
<td>-1.59</td>
<td>-2.21</td>
</tr>
<tr>
<td>1972</td>
<td>-0.81</td>
<td>-0.24</td>
<td>0.40</td>
<td>0.55</td>
<td>1983</td>
<td>-0.82</td>
<td>0.25</td>
<td>0.55</td>
<td>0.18</td>
</tr>
<tr>
<td>1973</td>
<td>0.70</td>
<td>0.88</td>
<td>1.38</td>
<td>0.42</td>
<td>1984</td>
<td>-0.03</td>
<td>0.53</td>
<td>0.61</td>
<td>-0.43</td>
</tr>
<tr>
<td>1974</td>
<td>-0.40</td>
<td>0.41</td>
<td>0.34</td>
<td>-0.47</td>
<td>1985</td>
<td>-1.08</td>
<td>-0.73</td>
<td>-0.54</td>
<td>-0.18</td>
</tr>
<tr>
<td>1975</td>
<td>-1.20</td>
<td>-0.96</td>
<td>0.23</td>
<td>0.14</td>
<td>1986</td>
<td>-0.11</td>
<td>-0.51</td>
<td>-0.68</td>
<td>-0.39</td>
</tr>
<tr>
<td>1976</td>
<td>-0.69</td>
<td>-0.25</td>
<td>0.11</td>
<td>-0.24</td>
<td>1987</td>
<td>0.00</td>
<td>0.19</td>
<td>0.25</td>
<td>0.13</td>
</tr>
</tbody>
</table>
Table 6.10: Predicting Results on Benchmark Problems

<table>
<thead>
<tr>
<th>Agent</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC-Agent_NN</td>
<td>-0.53</td>
<td>0.45</td>
<td>0.03</td>
<td>-0.93</td>
<td>-0.67</td>
<td>-0.14</td>
<td>0.43</td>
</tr>
<tr>
<td>SC-Agent_FLGA</td>
<td>-0.78</td>
<td>0.73</td>
<td>-0.16</td>
<td>-1.36</td>
<td>-0.60</td>
<td>-0.28</td>
<td>0.53</td>
</tr>
<tr>
<td>Benchmark</td>
<td>-0.81</td>
<td>0.70</td>
<td>-0.40</td>
<td>-1.20</td>
<td>-0.69</td>
<td>-0.28</td>
<td>0.47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_8$</th>
<th>$S_9$</th>
<th>$S_{10}$</th>
<th>$S_{11}$</th>
<th>$S_{12}$</th>
<th>$S_{13}$</th>
<th>$S_{14}$</th>
<th>$S_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.79</td>
<td>1.41</td>
<td>2.15</td>
<td>-0.95</td>
<td>-1.16</td>
<td>-0.33</td>
<td>-0.45</td>
<td>-0.41</td>
</tr>
<tr>
<td>1.03</td>
<td>1.84</td>
<td>2.66</td>
<td>-1.17</td>
<td>-1.00</td>
<td>-0.22</td>
<td>-0.87</td>
<td>-0.53</td>
</tr>
<tr>
<td>1.02</td>
<td>1.91</td>
<td>2.57</td>
<td>-1.10</td>
<td>-0.82</td>
<td>-0.03</td>
<td>-1.08</td>
<td>-0.11</td>
</tr>
</tbody>
</table>

This means there are 15 benchmark problems. The inputs of benchmark problem $S_1$, for example, are the data from 1967 to 1971. The benchmark value for these inputs is the T-bill rate of the first quarter of 1972, $-0.81$.

**Experimental Results**

The 15 data windows are used to train these two agents (neural network and genetic algorithm). We then let the agents predict the interest rate of the first quarter following the training data windows. For example, for training data of 1967-1971, the outputs of the agents are the (predicted) T-bill rate of the first quarter of 1972. The prediction results of the two agents on the 15 benchmark problems are summarized in Table 6.10.

Figure 6.1 shows the predicted values of SC-Agent_NN and SC-Agent_FLGA and the benchmark values from 1972 to 1986. The plot of the distances from the prediction values to the benchmark values for each data windows is depicted in Figure 6.2.

The average distance for the prediction values of SC-Agent_NN is 0.387. The value for SC-Agent_FLGA is 0.123. Based on Figures 6.1 and 6.2 and the average distances, one can see that the prediction performance of SC-Agent_FLGA is better than that of SC-Agent_NN for the benchmark problems. Mapping the distances to satisfactory degrees (refer to Table 6.2), [0, weak satisfaction] is added to the track record of SC-Agent_NN as its initial value, and [0, strong satisfaction] to the track record of SC-Agent_FLGA as its initial value. Hence at this stage, if
Figure 6.1: Curves of Benchmark Values, SC-Agent_NN, and SC-Agent_FLGA

Figure 6.2: Distances with Benchmark Values
the middle agent needs to pick one agent for interest rate prediction, the result is SC.Agent_FLGA. Of course, the situation may change with the accumulation of track records of these agents.

6.4.3 Discussion

There is an underlying assumption for the proposed matchmaking algorithm, i.e., all requesters (agents receiving services) should have the ability to give evaluations and they are willing to do so after they receive the services. This represents a small binding commitment for a requester, but it is not too strict to be practical. For collecting the evaluation data, what a requester agent needs to do is to send a short message with its satisfactory degree to the middle agent. The middle agent then stores the satisfactory degree in its database. No action is needed for provider agents. Thus the cost for collecting the evaluation data is very low and can be ignored.

In this algorithm, both the representation of track records and the satisfactory degree values can be changed to suit the needs of different applications. The criteria for “most promising” can also be defined according to specific applications. What is emphasized here is that agents’ track records in matchmaking should be considered as well as how to provide initial values for track records.

6.5 Summary

Matchmaking in middle agents is a crucial issue in MASs in open environments like the Internet. It has been actively studied since the inception of software agent research. To our knowledge, almost all the work missed one point – the matchmaking is only based on the advertised capabilities of provider agents. The actual performance of provider agents in accomplishing delegated tasks is not considered at all.

To this end, we proposed to consider agents’ track records in matchmaking. The simulation results have shown that agents’ track records have a strong impact on the outcome of matchmaking. The proposed algorithms can pick up the provider agents based on the history in accomplishing similar tasks in the past rather than
choosing randomly. At the launching of an agent system, the proposed algorithms can provide initial values of the track records. With the agents' history and the initial values of the track records, the performance of matchmaking algorithms can be improved significantly, and the returned results are more accurate and reasonable. Although the discussions were concentrated on middle agents, the proposed algorithm is equally applicable to contract net systems.

In matchmaking, the middle agents face the syntactic and semantic heterogeneity of data and knowledge they access and receive from multiple heterogeneous agents, information systems and sources. In the prototype, the semantic heterogeneities were resolved by using a domain-dependent ontology (as discussed in Chapter 5).

In the initial value generation approach, one must know a priori of the attribute values of the benchmark problems, but this is not always the case. In some situations, it is impossible to obtain the attribute values in advance. In such cases, one can ask the agents to solve these problems first. One can then try to cluster the attribute values returned by agents using cluster analysis methods [146]. In this way, "heuristic" attribute values can be obtained. One can then use the "heuristic" attribute values as the benchmark values and back to the proposed initial value generation algorithm. Of course, this issue is subject to further research.
Chapter 7

Reasoning in Decision Making

In the skill model of the agent-oriented methodology for the analysis and design of hybrid intelligent systems (see Chapter 3), the inference mechanisms used by roles or agents need to be identified in addition to the basic services performed by agents.

In the agent-based intelligent technique society for complex decision making, two types of reasoning were identified – reasoning with information only in text form and reasoning with information in multimedia forms other than text form (see Section 4.3).

As if — then rules were adopted to represent the second and third levels of knowledge in the knowledge model, the discussions about the reasoning with information in text form are focused on reasoning with rules. Rule-based reasoning can be exact reasoning that only deals with precise information or inexact reasoning that can deal with fuzzy information.

The inexact reasoning based on fuzzy logic (also called approximate reasoning) is a time-consuming task as it is computationally intensive. We will propose a parallel implementation to speed up it.

Reasoning with multimedia information is a new concept. To help understand this concept, the three levels of multimedia information processing that we identified will be given. Reasoning with multimedia information includes reasoning with image or graphic information, reasoning with audio information, reasoning with video information etc. As they all are very difficult to tackle, this chapter only
discusses reasoning with still image information in detail. The contributions of this chapter include:

- The implementation of one form of rule-based reasoning with precise information – forward chaining in the prototype of agent-based intelligent technique society (Section 7.1);

- Exploring the parallelism in approximate reasoning to speed up this time-consuming, computationally intensive task (Section 7.2);

- Identifying the three levels of multimedia information; clarifying the concepts of reasoning with multimedia information (Section 7.3); and solving reasoning with still image information by using symbolic projection (Section 7.4).

Some material in this chapter is also reported in [45, 95].

7.1 Reasoning with Rules

If-then rules have become the most popular form of declarative knowledge representation used in artificial intelligence applications. There are several reasons for this. Knowledge represented as if–then rules is easily understandable. Most people are comfortable reading rules, in contrast to knowledge represented in predicate logic. Each rule can be viewed as a stand-alone piece of knowledge or unit of information in a knowledge base. New knowledge can be easily added, and existing knowledge can be changed simply by creating or modifying individual rules. To this end, the if–then rules are used to represent the second and third levels of knowledge (domain specific knowledge and meta knowledge) in the knowledge model.

Rules are easily manipulated by reasoning systems. Forward chaining can be used to produce new facts (hence the term “production” rules), and backward chaining can deduce whether statements are true or not. Forward chaining was adopted in the agent-based soft computing society.

In the prototype of the agent-based intelligent technique society, we implemented the forward-chaining based on Bigus’s work [97]. The specificity conflict resolution strategy is adopted. That is, the number of antecedent clauses, as the
primary method for selecting a rule to fire. If two or more rules have the same number of antecedent clauses, we select the first rule we encounter. This is chosen because it is easier to implement.

The Java implementation includes a Rule class, a RuleVariable class, and a RuleBase class, as well as several support classes such as Clause.

Our forward-chaining implementation uses methods in both the RuleBase class and the Rule class. The forwardChain() method in the RuleBase class contains the main control logic for forward chaining. The method first allocates the conflictRuleSet vector. The match() method is called with a Boolean true parameter to force an initial test of all rules in the rule base. This returns with the initial conflictRuleSet, a Vector of the rules which are triggered and could be fired. We then enter a while() loop, which runs until we have an empty conflictRuleSet. Inside the loop, we first call the selectRule() method, passing the conflictRuleSet as a parameter. The selectRule() method performs the conflict resolution strategy (specificity) and returns with a single rule to fire. We call the Rule.fire() method to perform the consequent clause assignment, and then retest all Clauses and Rules which refer to the updated Variable. With the updated variableList, we call match() again, but this time we pass in a Boolean false parameter value. This tells match() to only look at the rule truth values, not to test each rule. The Java codes are shown below (for demonstration purpose):

```java
public void forwardChain() {
    Vector conflictRuleSet = new Vector();
    // first test all rules, based on initial data
    conflictRuleSet = match(true); // see which rules can fire

    while (conflictRuleSet.size() > 0) {
        Rule selected = selectRule(conflictRuleSet);
        // select the 'best' rule
        selected.fire(); // fire the rule
        // do the consequent action/assignment
        // update all clauses and rules
        conflictRuleSet = match(false);}}} // see which rules can fire
```
This implementation works well in the prototype when accomplishing exact reasoning, but is inefficient in doing approximate reasoning. This motivated us to explore the parallel (software) implementation of approximate reasoning that is suitable for agents.

7.2 Parallel Implementation of Approximate Reasoning

In complex decision making, there is much fuzzy and inexact information available. To deal with such information, approximate reasoning based on fuzzy logic is employed in the skill model of the society. For example, when determining a client's investment policy, approximate reasoning is used (see Section 4.6). But approximate reasoning is a time-consuming task as it is computationally intensive.

Over the past decade, in the application areas requiring high speed processing such as real-time control, special purpose microprocessors and fast fuzzy inference systems have been developed. Togai and Watanabe proposed a design for a special purpose VLSI chip for processing fuzzy logical rules [47]. Kim et al. developed a high speed flexible fuzzy hardware called KAFA [48]. There appeared some commercial fuzzy information processing products such as FC110, FP5000. The author also discussed the hardware implementation of approximate reasoning [49]. These pieces of dedicated fuzzy processing hardware work well in some specific application areas. However, their applications are rather limited, especially inappropriate for multi-agent applications. When incorporating into multi-agent systems, one needs some kind of general purpose and fast software implementations of approximate reasoning.

To this end, the parallelism in approximate reasoning is explored and a parallel implementation of it will be proposed. The experiment conducted on a PVM (Parallel Virtual Machine) will be also discussed. A brief introduction to approximate reasoning is given in Section 7.2.1 first, and then the parallel implementation as well as some experimental results is presented in Section 7.2.2.
7.2.1 A Brief Introduction to Approximate Reasoning

The theory of approximate reasoning was first developed by L. A. Zadeh [50]. This theory provides a mechanism for reasoning with information that is imprecise, nonspecific, and fuzzy. The theory translates the linguistic statements into possibility distributions and then manipulates these possibility distributions using the projection principle, extension principle, and generalized modus ponens etc. to obtain results.

In the approximate reasoning model, the knowledge in the knowledge base (KB) is represented by if – then rules in canonical forms. In more concrete terms, any unconditional proposition, \( p_1 \), e.g., annual income is low, may be represented in a canonical form, \( cf(p_1) \),

\[
p_1 \rightarrow cf(p_1) = X \text{ is } A
\]  

(7.1)

Any conditional proposition, \( p_2 \), e.g., if annual income is low then risk tolerance is low, may be expressed as

\[
 cf(p_2) = IF \ X \text{ is } A \ THEN \ Y \text{ is } B
\]  

(7.2)

where \( X \) and \( Y \) are variables taking values in some universe of discourse \( V \) and \( U \), \( A \) and \( B \) are fuzzy subsets. A more generally conditional proposition, \( p_3 \), its canonical form is:

\[
 cf(p_3) = IF \ X_1 \text{ is } A_1 \text{ and } \ldots \text{ and } X_n \text{ is } A_n \ THEN \ Y \text{ is } B \ (n \geq 1)
\]  

(7.3)

(7.1) induced a possibility distribution

\[
 \Pi_x = A(v), v \in V.
\]

(7.3) can be expressed as the proposition

\[
 (X_1, X_2, \ldots, X_n, Y) \text{ is } D,
\]

where \( D \) is a fuzzy subset of \( V_1 \times V_2 \times \ldots \times V_n \times U \). A number of possible forms exist for representation \( D \). Two of the more commonly used are:

\[
 D(x_1, x_2, \ldots, x_n, y) = \max[A'_1(x_1), \ldots, A'_n(x_n), B(y)],
\]  

(7.4)
where $A'_i(x) = 1 - A_i(x)$, and

$$D(x_1, x_2, \ldots, x_n, y) = \min\{1, (1 - \min_j[A_j(x_i)]) + B(y)\}. \quad (7.5)$$

If one has a rule such as (7.3), and has known some pieces of data

$$X_1 \text{ is } C_1$$
$$X_2 \text{ is } C_2$$
$$\ldots \text{ (known facts)}$$
$$X_n \text{ is } C_n$$

According to the compositional rule of inference, one can obtain the inferred result

"$Y \text{ is } E$", where

$$E(y) = \max_x[(1 - A_1(x)) \land C_1(x)] \lor \max_x[(1 - A_2(x)) \land C_2(x)] \lor \ldots \lor \max_x[(1 - A_n(x)) \land C_n(x)] \lor B(y) \quad (7.6)$$

or

$$E(y) = \max\{\min_j[C_j(x_i)] \land (1 - \min_j[A_j(x_i)]) + B(y)\} \quad (7.7)$$

Recall the fuzzy rules for evaluating a client's financial risk tolerance ability in Section 4.6.1. The terms of the linguistic variables (annual income (AI), total net-worth (TN), and risk tolerance (RT)) described by triangular and part of trapezoidal numbers formally have the same membership functions presented analytically below (refer to Figure 7.1):

$$\mu_L(v) = \begin{cases} 1 & \text{for } 0 \leq v \leq 20, \\ \frac{50-v}{30} & \text{for } 20 \leq v \leq 50, \end{cases}$$

$$\mu_M(v) = \begin{cases} \frac{v-20}{30} & \text{for } 20 \leq v \leq 50, \\ \frac{80-v}{30} & \text{for } 50 \leq v \leq 80, \end{cases} \quad (7.8)$$

$$\mu_H(v) = \begin{cases} \frac{v-50}{30} & \text{for } 50 \leq v \leq 80, \\ 1 & \text{for } 80 \leq v \leq 100. \end{cases}$$
Here \( v \) stands for \( x, y, \) and \( z \), meaning \( x \) substituted for \( v \) in (7.8) gives the equations of the terms in Figure 7.1(a), \( y \) substituted for \( v \) produces the equations of terms in Figure 7.1(b), and \( z \) substituted for \( v \) gives the equations of terms in Figure 7.1(c) (the second term \( \mu_M(v) \) should read \( \mu_{MO}(z) \)).

Assume a client provides the following information: \( x = 40 \) in thousands (annual income) and \( y = 25 \) in ten of thousands (total net-worth). They are matched against the appropriate terms in Figure 7.1. The corresponding membership function values calculated from (7.8) are

\[
\mu_L(40) = 1/3, \mu_M(40) = 2/3, \mu_L(25) = 5/6, \mu_M(25) = 1/6.
\]

There are four active rules (among the nine in Section 4.6.1), 1, 2, 4, and 5. When these active rules are fired based on (7.6) and combined together, the conclusion "the client's risk tolerance is B" is obtained, where \( \mu_B(z) \) is defined by the following equation:

\[
\mu_B(z) = \begin{cases} 
\frac{1}{5} & \text{for } 0 \leq z \leq 25.625, \\
\frac{z-20}{30} & \text{for } 25.625 \leq z \leq 40.625, \\
\frac{2}{3} & \text{for } 40.625 \leq z \leq 59.375, \\
\frac{80-z}{50} & \text{for } 59.375 \leq z \leq 80.
\end{cases}
\]


![FAM Matrix Diagram]

**Figure 7.2: An Example of a Two-Dimensional FAM Matrix**

### 7.2.2 Parallel Implementation

We use the fuzzy associative memory (FAM) matrix to store and represent fuzzy rules when implementing approximate reasoning (both sequential and parallel ones). FAM is a very simple and useful way to process fuzzy rules [51]. Figure 7.2 shows an example of a FAM matrix that represents the nine rules described in Section 4.6.1. For example, the shadowed entry in Figure 7.2 represents rule 8 (refer to Section 4.6.1). By representing fuzzy rules in FAM, we get the bonus of being able to handle multiple lower-dimensional FAM matrices simultaneously.

**Exploiting Parallelism**

The membership function of a fuzzy subset is usually continuous. When implemented, it must be digitized. Without loss of generality, we assume that all the membership functions take values at $S$ sampling points (typically, $S$ takes the value of 64 or 128 to guarantee the precision of results.) The antecedents of each rule in KB have at most $n$ terms. And the number of processors in the parallel system is $p$.

Assume that there are $m$ rules ready to fire in the KB after one Recognition-action cycle completed. In a new Recognition-action cycle, there are three types of parallelism available (refer to formula (7.6)):

- Fine granularity.

  The processing within each antecedent term, $\max_x [(1 - A_i(x)) \land C_i(x)]$, is parallel, but the processing of all $n$ antecedent terms and $m$ rules is sequential.
For $i := 1$ to $m$

begin

for $j := 1$ to $n$

Doall for $k := 1$ to $\lfloor S/p \rfloor$

Compute $(1 - A_{ij}(x_k)) \land C_j(x_k)$;

Compute the maximum value of $S$ values and store it in $R[i,j]$; /* on $p$ processors */

Endall

$R_{max} := \max_j R[i,j]$; /* compute the maximum value of $n$ value on $p$ processors */

for $k = 1$ to $\lfloor S/p \rfloor$

$E(y_k) = \max [R_{max}, B_i(y_k)]$

end

• Middle granularity.

We can parallel process the different antecedent terms $\max_x [(1 - A_i(x)) \land C(x)] (i = 1, 2, \ldots, n)$ when executing a rule. But within one antecedent term, the processing is sequential. The processing of $m$ rules is also sequential.

For $i := 1$ to $m$

Begin

Doall for $j := 1$ to $\lfloor n/p \rfloor$

for $k := 1$ to $S$

compute $(1 - A_{ij}(x_k)) \land C_j(x_k)$;

Compute the maximum value of $S$ values and store it in $R[i,j]$; /* on $p$ processors */

$R_{max} := \max_j R[i,j]$;

Endall

for $k = 1$ to $\lfloor S/p \rfloor$
\[ E(y_k) = \max [R_{\text{max}}, B_t(y_k)] \]

End

- Coarse granularity.

Because of the existence of the partial matching problem in the approximate reasoning model, there are more rules ready to fire at the same time compared to the exact reasoning models. So we can exploit the parallelism at rule level, i.e., the processing of \( m \) triggered rules is parallel, but the processing within one rule is sequential.

Do all

For \( i := 1 \) to \([m/p]\)
for \( j := 1 \) to \( n \)
begin
for \( k := 1 \) to \( S \)
compute \((1 - A_{ij}(x_k)) \land G_j(x_k);\)
compute the maximum value of \( S \) values;
end
compute the maximum value of \( n \) values and assign it to \( R_{\text{max}};*/\) on \( p \) processors */
for \( k := 1 \) to \( S \)
\[ E(y_k) = \max [R_{\text{max}}, B_t(y_k)] \]
End all

**Overall Speedup Results**

The platform we used to test the programs is a PVM. The PVM software provides a unified framework within which parallel programs can be developed in an efficient and straightforward manner using existing hardware. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine. PVM transparently handles all message routing, data conversion, and task scheduling across a network of incompatible computer architectures.
In the PVM, one can use the "add host name" command to configure it up to 16 machines. For the sake of comparison, all the host machines we used were personal computers with Intel Pentium II MMX-200Mhz CPUs. In this experiment, the crowd computation (master-slave) model and data decomposition workload allocation strategy were used.

We tested the parallel speedups using the simplified client financial risk tolerance model. For the purpose of this experiment, the rules were extended to have eight antecedent terms and increased the total rule number to 1000. The speedup in fine granularity was not remarkable, but the speedup in middle and coarse granularities was good. Figure 7.3 shows the parallel speedups on the PVM. The result shows that for the approximate reasoning and PVM environment, one should use a relatively coarse grain size in parallel processing.

Because PVM is actually a network, the execution time is slightly different for each run. The results shown in Figure 7.3 are the average values of five runs. The
difference compared to the ideal case is mostly due to the network communication overheads. If it was implemented on a shared-memory multiprocessor computer, the performance should be better.

By exploiting the parallelism of approximate reasoning, good speedup was obtained. The parallel program can be easily converted to agents by a wrapper (see Section 4.6.3).

Now the problem of reasoning with multimedia information is examined. But prior to this, the three levels of multimedia information processing we identified are outlined.

### 7.3 Three Levels of Multimedia Information Processing

In today’s electronic world, there are volumes of digital multimedia data generated every second. The first thing needed is to store these data for later use. With the advent of relatively cheap, large online storage capacities and advances in digital compression, comprehensive sources of text, image, video, and audio etc. multimedia data can be stored and made available for research and applications.

As the amount of multimedia data is increasing dramatically, some effective and efficient retrieval techniques are needed to find useful and relevant multimedia information from these large digital data repositories such as multimedia databases and multimedia-based WWW pages etc. These retrieval techniques should provide not only fewer bits but also the right bits to users.

In some applications, to find out the relevant information is the final goal, but in others this is not so. Just like in traditional expert systems, some “known facts” must be given before the systems can reach some conclusions. In many multimedia application systems, the retrieved multimedia information is used as “known facts”. Based on these “known facts” and domain knowledge, the systems try to obtain further conclusions the users want. Thus, in typical multimedia information processing, there are three levels – storage, retrieval, and post-processing of multimedia information (Figure 7.4).
Figure 7.4: Three Levels of Multimedia Information Processing

Currently, most of research work in multimedia information processing is concentrated on multimedia information storage and retrieval, especially effective indexing of multimedia data and content-based access to multimedia information, whereas the post-processing is left nearly untouched. In the next two subsections, brief summaries of indexing of multimedia information and content-based multimedia retrieval are given. Then the concepts and problems of multimedia information post-processing are identified and, some promising techniques that can be used to solve these problems are briefly presented in Section 7.3.3.

7.3.1 Indexing of Multimedia Data

To store large volumes of multimedia data, in addition to the requirements of very high capacity, fast access times, and high transfer rates, another very important aspect is effective indexing. This is also closely related to multimedia information retrieval.

Indexing is a fundamental operation for large databases. Index structures are closely related to data representation. This implies that one should have different index structures for data in different media forms such as image, video, and audio. If visual elements are characterized as feature vectors, then multidimensional
indexing structures can be employed. Currently available index structures only allow indexing of low-dimensional vectors. However, most index structures available perform filtering instead of ranking, as needed in multimedia information retrieval. Moreover, it is often necessary to change similarity functions at run-time, by attributing different weights to each dimension. All this requires the development of new indexing methods [101] (p. 15). The following are some typical currently available indexing methods:

- Indexing of string attributes. String attributes of image/videos are in the form of keywords, alphanumeric strings or scripts (referring to image concepts or spatial relationships) [102]. Conventional indexing techniques for textual information can be used to index keywords or alphanumeric strings. Hashing tables and signature files are the most common indexing methods employed. In particular, signature files act as a filter that preliminarily discards uninteresting documents. A signature is a code which represents a document and is inserted in a hash table.

- Indexing of visual attributes. If visual properties—color, texture, shape—are modeled as points in a multidimensional metric feature space, index structures, known as point access methods (PAMs), developed for spatial data, can be employed [103]. PAMs employ non-hierarchical as well as hierarchical indexing. In general, hierarchical index structures are more effective. Non-hierarchical index structures are only suited for low-dimensional spaces. In addition to PAMs, other multidimensional point indexing methods include triangle inequality, fixed grids and grid files, K-d trees, R-trees, \( R^+ \) trees, \( R^* \) trees, and \( SS \)-trees.

- Active indexes. Active indexes have been proposed by Chang [98, 104]. They are based on the idea that images in a database can invoke actions by themselves. In particular, images can request image processing operations, display themselves in an appropriate way, or search for other related images. For this purpose, images are associated to a knowledge structure composed of active index cells. Index cells accept input messages, perform some form of processing and output messages to other cells. They output different messages
depending on their current state. User interaction determines the changes in response by these cells. Active indexes are, therefore, built in the form of dynamically changing nets.

7.3.2 Content-Based Multimedia Information Retrieval

Information retrieval has been attracting the attention of many researchers [105, 106, 107]. Now in this field, more emphasis has been placed on content-based retrieval [108, 109, 21]. That is because there is such motivation and need.

The amount, variety, and distribution of online information is rapidly exploding with the advent of the WWW model of correlating digital multimedia information in a global Internet. Such online information sources are becoming increasingly large and multimedia in nature, taking the form of text, image, video, and audio repositories. Unfortunately, this information is highly unstructured and current analysis and search technologies leave users and application developers either (1) yearning for facilities beyond simple keywords, color histograms, and other shallow content cues; (2) yearning for the ability to adjust or extend the few rigid content-oriented concepts a technology can deliver; or (3) yearning for the ability to decompose and manifest domain concepts from content collected from multiple information sources. Simply speaking, end-users want access to these multimedia objects based on situational concepts and semantic patterns in and across the content itself. In other words, there is a growing need to efficiently structure and archive the content in multimedia repositories in a manner amenable to extendible and deep analysis of domain specific patterns.

Content-based multimedia information retrieval contains two parts: content-based retrieval for multimedia databases and content-based access to WWW multimedia information.

Content-Based Retrieval for Multimedia Databases

In content-based retrieval for multimedia databases, there are two principal ways for the representation of queries, namely, “query-by-example” (QBE) and “query-by-subject/object” [108].
QBE allows the user to specify a query condition in an intuitive way, i.e., it is easy to express a query condition in a natural way. In QBE, a query condition for non-textual data is represented, for example, in the form of a rough sketch, a rough painting with colors, or a motion example of trajectory and/or velocity. Such representations express the query condition for non-textual data better than keywords, since it is often difficult to express slight differences of shape, color, or spatio-temporal relation with keywords. QBE is one of the promising schemes for representing query conditions for multimedia database retrieval in a natural and an intuitive way. Since a query condition in QBE is the representation of an example that a user wishes to retrieve, semantics of the data are not analyzed and processed by a database system during the process of query evaluation.

However, there are also cases where a database is queried by specifying semantic contents. This type of query specification is usually called query-by-subject/object. Query-by-subject/object allows the user to specify a subjective description of a query condition. In such cases, knowledge is required to capture the semantic contents of multimedia data as well as to interpret the query.

In short, QBE works well for content-based retrieval in the case where contents are formed in terms of a single data type. However, the QBE approach is not adequate when two or more heterogeneous types of data form the content. Rather "query-by-subject/object" is appreciated for such cases, where a keyword can well represent the semantic content.

Currently, many more studies have been done in relation to content-based retrieval that refers to single non-textual data. However, in our opinion content-based retrieval studies for multimedia databases should pay more attention to the multimedia content that is associated with heterogeneous types of data. Extracting implicit contents from semantically related heterogeneous types of data is advantageous in some aspects. One reason is that clues from two or more pieces of heterogeneous data which are semantically related to each other give more implicit content, which cannot be extracted from only one of them. Another reason is that evaluating contents extracted from two or more pieces of data together may give the results with more certainty, since current image/video processing or audio processing techniques do not always give a result with enough accuracy.
Content-Based Access to Web-Based Multimedia Information

For content-based access to web-based multimedia information, the most interesting work is the Content-Based Access to Multimedia Document effort (CBAM) of Microelectronics and Computer technology Corporation (MCC) [21]. The CBAM effort is an outgrowth of the MCC InfoSleuth project [20], wherein technologies have been developed for using agents to achieve semantic access to networks of heterogeneous and diverse information sources. InfoSleuth is a general agent-based environment for advertising, finding, and integrating information annotated by domain ontologies. The CBAM mission is to enable timely and semantically-driven filtering and gathering of multimedia resources in a dynamic information network. The general InfoSleuth approach – the separation of the data space from the semantic space and the use of networks of agents to dynamically link these spaces – holds in networks of multimedia information sources. The key to accomplishing CBAM applications in this manner is the semantic content extraction algorithms encapsulated in multimedia resource agents and the advancement of ontologies for content correlation to better handle temporal, spatial, and uncertain aspects of multimedia content analysis.

Based on MCC’s study on CBAM, the following two opportunities are evident:

- The ability to detect content-based concepts as patterns of content abstractions across multiple information sources and over temporal histories. In other words, the InfoSleuth approach enables multi-source information collection and multi-source monitoring over temporal sequences. Such multi-source primitives have yet to be applied in a content-based multimedia environment.

- The ability to separate the concept (or semantic) space from the document (or data) space and introduce dynamic association operators. This is a key concept in InfoSleuth that manifests a scalable and dynamic information network. This separation holds in CBAM applications and provides a natural process of wrapping multimedia semantic content extraction algorithms in resource agents that interact with the information network.
7.3.3 Post-Processing of Multimedia Information

It is worth noting that there are many researchers working on multimedia information retrieval (especially content-based retrieval), whereas the processing of retrieved multimedia information is almost ignored. Here, the problems are highlighted so as to trigger further research on this interesting field. In our opinion the most important topics in multimedia information post-processing are fusion of multimedia information and reasoning with multimedia information.

Figure 7.5: A Framework for Multimedia Information Fusion (Based on [21], pp. 107-109)

Fusion of Multimedia Information

Fusion of information is about how to use pieces of information simultaneously provided by several sources in order to come to a conclusion or a decision. It concerns the fusion of evidence provided by several sources regarding the unique object. In this framework, complementary information is used to construct a global description of the object from partial views and to eliminate inconsistencies from the elementary descriptions.
Currently, most approaches in information fusion can only deal with information in alphanumeric form. In many multimedia application systems, one may have information in different media forms such as text, image, audio, and video to describe the unique object. For example, the medical information of a patient can contain text (e.g., patient's history document), CT image, B-ultrasonic wave graphics etc. All these information must be combined for doctors to hold group consultations.

Intuitively, multimedia information fusion should based on some effective transformation among information in different media forms and media abstractions. Some work has been done on transformation [111] and media abstractions [110]. A framework for multimedia information fusion is shown in Figure 7.5.

Figure 7.5 illustrates a three-level model for information: data, abstracted information, and fused knowledge. Information sources such as cameras, sensors or computers usually provide continuous streams of data. Such data need to be abstracted into various forms of abstractions, so that the retrieval, processing, consistency analysis and combination of abstracted information becomes possible. Finally, the abstracted information needs to be integrated and transformed into fused knowledge.

Figure 7.5 also illustrates the relationships among data sources, data, abstracted information and fused knowledge, with emphasis on the diversity of data sources and the multiplicity of abstracted representations. For example, a video camera is a data source that generates video data. Such video data can be transformed into various forms of abstracted representations:

- text (video-to-text abstraction by human or software agents);
- keyword (video-to-text abstraction by human or software agents);
- assertions (logical representation of abstracted facts);
- qualitative spatial description (abstraction such as the symbolic projection [25]); and
- time sequences of frames (abstraction where both spatial and temporal relations are preserved).
In Figure 7.5, the transformations from data to abstracted representations are indicated by circles. For example, the image data can be transformed into keywords, assertions (facts) and qualitative spatial description.

As illustrated in Figure 7.5, multimedia information fusion is feasible when information from different sources can be converted into similar representations, indicated by several circles in the same horizontal row. For example, the system may support the transformation of image, text and web pages into assertions (facts). The fusion then takes place among assertions. Such fusion is termed horizontal fusion because it combines information abstracted from different media encoded in the same uniform representation.

Another type of fusion is applicable to data from similar media with different abstracted representations so that they can be combined and checked for consistency etc. This fusion is called vertical fusion because it combines information having different representations at different levels of abstraction.

Horizontal fusion can be accomplished with the help of an artificial neural network due to its ability to combine information abstracted from different media and adequately encoded in the same uniform representation. The active index [98, 104] can be used in vertical fusion due to its ability to obtain information from different sources and actively connecting them by dynamic linking (using index cells). For example, an image can be linked to a keyword or to an assertion (fact), and then domain-specific algorithms can be applied to check their consistency. Vertical fusion is associative and combines information in different representations. An artificial neural network with fixed connections is not as appropriate as an active index with flexible connections.

Recall the overall agent framework for complex problems (Figure 1.3). The fusion of multimedia information is one of the principal tasks of the information gathering agent subsystem.

Reasoning with Multimedia Information

As previously noted, in many multimedia applications, to retrieve the relevant information is not the final step of these applications. One needs to use the retrieved information (in different media forms) as known facts to do some reasoning based
on domain knowledge. As these known facts are represented in different media forms—not only in alphanumeric form, traditional reasoning algorithms cannot be applied directly. This kind of reasoning is termed reasoning with multimedia information. It is different from visual reasoning and spatial reasoning etc. used in multimedia information retrieval, especially image retrieval.

**Visual reasoning** is the process of reasoning and making inferences, based upon visually presented clues. Visual reasoning may help the user find the most desirable query from examples and clues [53].

Visual reasoning is widely used in human-to-human communication. For example, the teacher draws a diagram on the blackboard. Although the diagram is incomplete and imprecise, the students are able to make inferences to fill in the details, and gain an understanding of the concepts presented. Such diagram understanding relies on visual reasoning so that concepts can be communicated. Humans also use gestures to communicate. Again, gestures are imprecise visual clues for the receiving person to interpret.

In human-to-computer communication, a recent trend is for the human to communicate to the computer using visual expressions. Typically, the human draws a picture, a structured diagram, or a visual example, and the computer interprets the visual expression to understand the user's intention. This has been called visual coaching, programming by example, or programming by rehearsal by various researchers.

Visual reasoning is related to spatial reasoning. **Spatial reasoning** is the process of reasoning and making inferences about problems dealing with objects occupying space. These objects can be either physical objects (e.g., books, chairs, cars, etc.) or abstract objects visualized in space (e.g., database objects). Physical objects are tangible and occupy physical space in some measurable sense. Abstract objects are intangible but nevertheless can be associated with a certain space in some coordinate system. Therefore, visual reasoning can be defined as spatial reasoning on abstract objects visualized in space.

To solve the problems of reasoning with multimedia information requires the
contribution of many research disciplines such as image/video analysis and processing, pattern recognition and computer vision, multimedia data modeling, multidimensional indexing etc. The following technologies promise to partially solve this problem:

- Effective transformations among different media. As discussed in the previous subsection, if information in different media forms can be converted into similar representations, many traditional inference algorithms can be employed to do the reasoning.

- Symbolic projection theory [53]. The theory of symbolic projection is a theory of spatial relations. This theory is the basis of a conceptual framework for image representation, image structuring and spatial reasoning. This theory is also suitable for reasoning with still images (refer to Section 7.4).

- The Tele-Action Object (TAO) techniques [112]. A TAO, \( (G, K) \), is a multimedia object with associated hyper-graph structure \( G \) and knowledge structure \( K \). The hyper-graph \( G \) is used to describe the connections and relations between the sub-TAOs within it. The knowledge \( K \) is used to describe the actions. If the knowledge structure \( K \) can be extended and provided with an inference capability, it is possible to use TAO to conduct reasoning with information in different media forms as the media types in a TAO can be text, graphics, image, moving-graphics, moving-image, audio, video, and so on.

In the next section reasoning with still image information by using symbolic projection will be discussed in detail [86].

### 7.4 Reasoning with Multimedia Information Using Symbolic Projection

As stated previously, post-processing of the retrieved multimedia information – fusing information in different media forms and reasoning with multimedia information – is needed in many multimedia applications. For example, when the agent-based
intelligent technique society is applied to financial investment planning, one sub-
task is to give advice about stock buying or selling. To accomplish this task, some 
reasoning is necessary, which is based on the moving average chart of a specific 
security (Figure 7.6) as well as many other analyses.

![Figure 7.6: Example Moving Average Chart](image)

The essential part of the problem is the matching of two moving average images 
(charts). Traditional approaches to such problems are measured on the basis of 
maximum-likelihood or a minimum distance criterion. The symbolic description of 
visual information, such as shape or spatial relations, is a very difficult task using 
the traditional approaches. Attempts to describe this information textually can 
lead to representations that are either too general or too complex. It is proved that 
the symbolic projection approach is more flexible and efficient [113, 114, 115]. Thus 
we use symbolic projection theory to solve such problems and take the processing 
of moving average charts as an example.

### 7.4.1 Description of the Problem

When giving advice about investment in the stock market in the agent-based finan-
cial investment planning system, the system will use *fundamental analysis, technical*
**CHAPTER 7. REASONING IN DECISION MAKING**

Analysis of securities, and other domain knowledge. Fundamental analysis endeavors to determine the fair value of a share. The emphasis of technical analysis is on information generated by the market itself. Technical analysis is an attempt to forecast future prices by studying past prices. Traditionally, this has been done using various types of charts that provide a visual record of past prices. There are four conventional types of charts—the bar chart, the candlestick chart, the point and figure chart, and moving averages. The emphasis here is on the moving average charts.

A moving average of past prices can be used as an indicator of a price trend. There are two main trading rules for moving averages:

- A buy signal is given when the price moves up and crosses over the moving average from below.

- A sell signal is given when the price moves down and crosses over the moving average from above.

Based on the trading rules, there are three rules that relate to the moving average chart in the knowledge base of decision making agents:

- Rule 1: If the moving average chart of a security is similar to Figure 7.7(a), then buy this security;

- Rule 2: If the moving average chart of a security is similar to Figure 7.7(b), then sell this security;

- Rule 3: If the moving average chart of a security is similar to Figure 7.7(c), then do not buy and sell this security.

Here, it is natural to represent the condition parts of these rules by graphics (chart) directly. Actually, it is very difficult to represent them in text precisely whereas they are easy to operate. If one retrieved some web-sites on the Internet and obtained the moving average chart of a specific security (Figure 7.8), how can one infer the conclusion using these rules? This is one form of reasoning with multimedia information. For this specific problem, it is actually a problem of reasoning with still images.
Figure 7.7: Moving Averages of Security Price

Figure 7.8: Retrieved Moving Average of a Specific Security
We employed *symbolic projection theory* to do such reasoning. We represented the moving average chart as 2D strings in symbolic projection, and then used the 2D string matching algorithm to accomplish the reasoning.

### 7.4.2 A Brief Introduction to Symbolic Projection

The theory of symbolic projection was first developed by Chang and co-workers [113]. It forms the basis of a wide range of image information retrieval algorithms. It also supports pictorial-query-by-picture, so that the user of an image information system can simply draw a picture and use the picture as a query. Many researchers have since extended this original concept, so that there is now a rich body of theory as well as empirical results [114, 116]. The extended theory of symbolic projection can deal not only with point-like objects, but also objects of any shape and size. Moreover, the theory can deal with not only one symbolic picture, but also multiple symbolic pictures, three-dimensional (3D) pictures, a time sequence of pictures, etc.

A symbolic picture is a two dimensional matrix of symbols. Each object of the real image is represented by a symbol located in the centroid of the object. A symbolic picture can have at least two symbolic projections: the $x$-projection and $y$-projection. The $x-$ or $y-$projection of a symbolic picture can be constructed by projecting the names of objects in each column of the symbolic picture onto the $x-$ or $y-$axis. A pair of two symbolic projections is called a 2D string.

Let $\Sigma$ be a set of symbols, or the vocabulary. Each symbol might represent a pictorial object, a pixel, etc.

Let $A$ be the set $\{=,<,\}|$, where "="", "<", and "|" are three special symbols not in $\Sigma$. These symbols will be used to specify spatial relationships between pictorial objects.

A 1D *string* over $\Sigma$ is any string $x_1x_2\ldots x_n, n \geq 0$, where the $x_i$'s are in $\Sigma$.

A 2D *string* over $\Sigma$, written as $(u,v)$, is defined to be

$$(x_1y_1x_2y_2\ldots y_{n-1}x_n, x_p(1)z_1x_p(2)z_2\ldots z_{n-1}x_p(n))$$

where $x_1\ldots x_n$ is a 1D string over $\Sigma$, $p : \{1,\ldots,n\} \rightarrow \{1,\ldots,n\}$ is a permutation over $\{1,\ldots,n\}$, $y_1,\ldots, y_{n-1}$ is a 1D string over $A$ and $z_1,\ldots, z_{n-1}$ is a 1D string
over $A$.

In the above, the symbol ‘$<$’ denotes the left-right spatial relation in string $u$, and the below-above spatial relation in string $v$. The symbol ‘$=$’ denotes the spatial relation ‘at the same spatial location as’. The symbol ‘$|$’ denotes ‘edge-to-edge’ spatial relations (two objects are in direct contact either in the left-right or in the below-above direction). Therefore, the 2D string representation can be seen as the \textit{symbolic projection} of picture $f$ along the $x$- and $y$-axes.

A \textit{symbolic picture} $f$ is a mapping $M \times M \rightarrow W$, where $M = \{1, 2, \ldots, m\}$, and $W$ is the power set of $\Sigma$ (the set of all subsets of $\Sigma$). The empty set $\{\}$ denotes a null object.

Given $f$, one can construct the corresponding 2D string representation $(u, v)$, and vice versa, such that all left-right and below-above spatial relations among the pictorial objects in $\Sigma$ are preserved. In [53] (pp. 32-33), a formal algorithm (called \textbf{2Dstring}) for constructing 2D string $(u, v)$ from $f$ is presented.

2D string representation also provides a simple approach to perform sub-picture matching on 2D strings. An algorithm for 2D string matching (\textbf{2Dmatch}) is given in [53] (pp. 38-40).

With the edge-to-edge operator ‘$|$’, one can further segment an object into its constituent parts. This is accomplished by introducing cutting lines. A systematic way of drawing the cutting lines is as follows. First, the extremal points are found in both the horizontal and vertical directions. Next, vertical and horizontal cutting lines are drawn through these extremal points. This technique gives a natural segmentation of planar objects into the constituent parts.

With the cutting mechanism, one can also formulate a general representation, encompassing the other representations based upon different operator sets. This consideration leads to the formulation of a generalized 2D string system [117].

A \textit{generalized 2D string system} is a five-tuple $(\Sigma, C, A, e, "\langle, \rangle")$, where $\Sigma$ is the vocabulary; $C$ is the \textit{cutting mechanism}, which consists of cutting lines at the extremal points of objects; $A = \{<, =, |\}$ is the set of spatial operators; $e$ is a special symbol which can represent an area of any size and any shape, called the \textit{empty-space object}; and “$\langle, \rangle$” is a pair of operators which is used to describe local structure.
The cutting mechanism defines how the objects in an image are to be segmented, and also makes it possible for the local operator ",, ," to be used as a global operator to be inserted into original 2D strings.

The spatial operator set $A$ can be extended to contain other spatial relation operators used in different applications. For extension examples, refer to [114, 116].

### 7.4.3 2D String Representation of Moving Average Charts

Having these basic concepts about symbolic projections, the 2D string representation of moving average charts will be discussed.

Take Figure 7.7(b) as an example. The following cutting mechanism $C$ is applied: Choose the upper horizontal extremal point, $p$, and draw two vertical cutting lines $x = p + \delta$ and $x = p - \delta$. Others remain the same as described in Section 7.4.2. The segmented moving average chart is shown in Figure 7.9(a).

![Figure 7.9](image)

*Figure 7.9: (a) Moving Average Chart Segmentation Using Cutting Lines; (b) Moving Average with Benchmark Object; (c) Final Moving Average Segmentation with Benchmark Object*

The vocabulary is $\Sigma = \{a, b, c\}$. The 2D string representing the picture in Figure 7.9(a) is as follows:

$u : a | b | c, v : c = a < b.$

Therefore, rule 2 in Section 7.4.1 can be expressed as:

**Rule 2:** If $u : a | b | c$ and $v : c = a < b$ then sell the security.

Similarly, one can obtain the 2D string representations of other moving average charts and rewrite the corresponding rules. Then the 2D string matching algorithm or other string matching algorithms can be employed to do the reasoning.
7.4.4 Special Considerations for the Problem

The above representation does not fully describe the meaning lies in the antecedents of trading rules. For example, in Figure 7.10, \( g_1 \), \( g_2 \), and \( g_3 \) have different slopes, but their 2D representations are the same. This causes one problem. Whether the price trend is indicated by \( g_1 \), \( g_2 \), or \( g_3 \), a "sell" signal will be given. In practice, when the price drops slightly (indicating by \( g_3 \)), no "sell" signal is given.

![Image of Moving Average with Different Slopes](image)

Figure 7.10: Moving Average with Different Slopes

This is a difficult problem and it occurs due to two reasons. One is that the descriptions of trading rules themselves are fuzzy. The other is the insufficient representation of the 2D strings. For such a problem, simply extending the spatial operator set \( A \) cannot help. One may argue that one generalized symbolic projection - slope projection [53] can be used to solve this problem. But if one remembers that no object exists to make comparisons, then it still cannot be used directly. What is needed is a benchmark picture object that indicates whether the "sell" or "buy" signal should be given or not. Our concept is to introduce auxiliary object(s) in symbolic pictures that need to be processed. In the example, such an auxiliary picture object is termed the benchmark picture object. The benchmark picture object can be determined by consulting stock technical analysis experts.

Still take Figure 7.7(b) as an example. After introducing the benchmark object, the resulting moving average chart is shown in Figure 7.9(b) where the same cutting mechanism used in last subsection is applied, but with one more vertical cutting line through the upper extremal point.
In Figure 7.9(b), if the slope of the price trend is greater than that of the benchmark object \( s \), no "sell" signal is given. If less than or equal to that of \( s \), a "sell" signal is generated. The 2D string representation corresponding to Figure 7.9(b) is as follows:

\[ u : a|b_1|b_2 = s_1|c = s_2, \quad v : c = a < s_2 = b_2 = b_1 < s_1. \]

This 2D string representation can be further simplified as \( s_1 \) is useless for the reasoning, only \( s_2 \) can help. In this application, no reconstruction from the 2D strings is needed. Thus \( s_1 \) can be omitted in the 2D representation. We applied exactly the same cutting mechanism as that used in the last subsection (see Figure 7.9(c)), we obtained the final 2D string representation for the problem:

\[ u : a|b|c = s, \quad v : c = a < s = b. \]

For rule 2, if the 2D string of the retrieved price trend is \( u : a|b|c = s, \quad v : c = a < s = b \), then a "sell" signal is given. Otherwise, if the 2D string of the retrieved price trend is \( u : a|b|s = c, \quad v : s = a < c = b \), then no "sell" signal is given.

With the introduction of the benchmark object in the moving average charts, one can exactly represent the meaning in trading rules with 2D strings. One more problem remains to be solved: how can the benchmark object be added to a symbolic picture, and automatically construct the 2D strings from the corresponding picture?

In this application, the benchmark object is a line segment starting from the extremal point, \( p \). Assume the slope of the benchmark object (line segment) determined by stock technical analysis experts is \( k \). To add the benchmark object is simply drawing a line segment starting from \( p \) with slope \( k \). To construct the 2D strings from the symbolic picture with added benchmark object, we can still use the 2DString algorithm. The extended 2DString algorithm that can handle the benchmark object problem, 2DStringE, is outlined below:

**Algorithm 2DStringE**

\[
\text{Algorithm 2DStringE}(\text{picture}, k, u, v) \\
/ * This algorithm takes symbolic picture picture and the slope of benchmark */ \\
/ * object k as inputs, k is determined by stock technical analysis experts. */ \\
/ * Outputs are the 2D string representations \((u, v)\) of picture. */
\]
begin
/* object recognition */
recognize objects in the picture;
find the upper or lower extremal point \( p \) in \textit{picture};
/* add benchmark object */
draw a line segment starting from \( p \) with slope equal to \( k \);
/* segmentation */
applying cutting mechanism \( C \) to \textit{picture};
find centroid of each object;
find 2D string representation using Procedure \texttt{2Dstring}
end

7.4.5 Discussion

The above discussion is focused on the processing of moving average charts using symbolic projection. In the financial investment application and many other applications, another commonly used chart form to represent retrieved information is the pie chart. For example, when the system gives investment advice to the client, the system may analyze the revenues of the relevant companies or the trading volumes of some promising securities. They are usually represented by pie charts (refer to Figures 7.11 and 7.12).

![Figure 7.11: Revenue of a Company](image)

It is more difficult to reason with those kinds of pie charts. If the boundary lines (see Figure 7.13) that separate different component parts in the pie chart
are known, the task is relatively easy. Otherwise, one must first determine the boundary lines based on the gray levels of different component parts in the pie chart. After the boundary lines in the pie chart are obtained, one can convert the real pie chart to a symbolic picture. The symbolic projection theory can then be used to process this problem. When using symbolic projection to process pie charts, polar projection and concentric cutting mechanisms must be used [118].

We successfully solved the problem of reasoning with curve information (still
image) by using symbolic projection. The work presented here indicates that symbolic projection is a very promising methodology to solve the problem of reasoning with still image information. Of course, reasoning with other still image information (other than curve information) is subject to further research.

When solving the "reasoning with moving average charts" problem in the application, we introduced the concept of "benchmark objects". By introducing benchmark objects into the moving average charts, the problem was solved effectively. This approach can be generalized and applied to many other problems. For example, one kind of difficult problem, called dynamic route planning, can be solved by introducing extra "reference objects" in the symbolic pictures when using symbolic projection theory. This problem can be generally described as follows. A moving object (e.g., a robot) needs to determine its moving direction to an unfamiliar place. In the moving object's memory, there are some reference objects. Once it observes enough reference objects in the environment, it can then decide which direction it will move to. For such problems, one can represent the environment as a symbolic picture. When the moving object sees an object that is identical to one of the reference objects in its memory, this object is added to the symbolic picture as an auxiliary object. After introducing enough auxiliary objects, one can describe the relationships among all the objects in the symbolic picture and furthermore, decide the direction in which the moving object should move. This problem is similar to the problem of determination of the views of a moving object addressed by E. Jungert [120], but much more difficult. Once again, to thoroughly solve this problem needs further research.

The theory of symbolic projection was originally developed as a technique for iconic indexing to image databases, and this is still an important application area. Symbolic projection theory also includes other characteristics that made it suitable for various forms of spatial reasoning and in particular for qualitative spatial reasoning. We applied this theory for reasoning with still image information. These two concepts are different but they have some relations. Spatial reasoning is the process of reasoning and making inferences about problems dealing with objects occupying space [119]. The emphasis is on the spatial relationships of objects. The focus of reasoning with multimedia information in general, still image information
specific, is on the meaning or knowledge that lies behind the images.

7.5 Summary

This chapter addressed the reasoning problem in the agent-based intelligent technique society, which is part of the skill model.

Considering the domain knowledge and meta knowledge in the knowledge model are represented by if – then rules, one part of the discussion is on reasoning with rules. This includes exact reasoning with rules and approximate reasoning with fuzzy rules. All information processed is in text form. We implemented forward chaining (for exact reasoning) in Java. As approximate reasoning is very time-consuming, the parallelism in it was explored. The parallel implementation sped up approximate reasoning greatly and is suitable for incorporating into agents.

In many situations information is represented by not only text but also other media forms. Thus for these applications, reasoning with information in multimedia forms is needed. We then introduced the concept of “reasoning with multimedia information” by first identifying the three levels of multimedia information processing.

Reasoning with multimedia information includes reasoning with still image information, reasoning with video information, and reasoning with audio information, etc. To our knowledge, this problem has not previously been addressed.

In this chapter, we proposed using symbolic projection theory for reasoning with still image information. A case study of reasoning with moving average charts in finance was provided. When using symbolic projection to solve this problem, a new concept – introducing a benchmark object – was developed. There is much work still to be done for reasoning with multimedia information.
Chapter 8

Decision Aggregation in Multi-Agent Systems

When conducting the analysis and design of the agent-based intelligent technique society (see Chapter 4), a DECISIONAGGREGATOR role and corresponding Decision Aggregation Agents in the society were identified.

Decision aggregation is about the combination of different decisions from different decision makers to obtain an overall decision. It is an important issue in multi-agent systems (MASs) because in many multi-agent application systems, each agent may only have a limited amount of domain knowledge or information. An agent can make decisions based on its existing knowledge; however, its decisions will have to be analyzed and combined with other agents’ decisions for the decision quality. In financial investment planning applications, for example, some decision making agents have expertise in the stock market, some in real estate, etc. After they make decisions according to their own knowledge, there is a need to find some reasonable way to fuse or aggregate these partial decisions to obtain a final one.

There are two problems involved in decision aggregation. One is how to choose appropriate aggregation algorithms for specific applications. The other is how to implement these algorithms effectively and efficiently in MASs.

As aggregation algorithms are application-dependent, the financial investment planning was selected as an example to demonstrate why we choose ordered weighted averaging (OWA) aggregation algorithm for this application and how to aggregate
using OWA (Section 8.2). Three ways are then proposed – the stationary agent approach, the token passing approach, and the mobile agent approach – to implement aggregation algorithms in MASs (Section 8.3). After comparing the three approaches, we conclude that the first two are suitable for small-scale MASs, and the mobile agent approach is appropriate for large-scale MASs. A case study for financial portfolio selection based on different models and the aggregation of different portfolios is provided. The aggregated results are verified through experiments (Section 8.4). The materials in this chapter are also reported in [30, 31, 32].

8.1 Description of the Aggregation Problem

In multi-agent systems, each of the agents may have its own expertise. When they are asked to make a decision on the same task, the results may be different. In such situations, different decisions need to be aggregated to obtain a final one. To better understand the problem, here is an example in financial investment planning.

Suppose a user (investor) wants to know whether his investment policy (IP) should be aggressive or conservative. First, the user gives his annual income (AI) and total net-worth (TN) to the decision making agents through the interface agent. The decision making agents use their own knowledge (with the help of intelligent technique agents) to evaluate the user’s risk tolerance (RT) ability using rules such as If user’s AI is low (L) and TN is L, then user’s RT is L (see Section 4.6.1). Note that different decision making agents may have different rules similar to this one.

The decision making agents then delegate the information gathering agents to collect data concerning the rising or falling of interest rates, the state of the stock market, the trade balance, the unemployment rate, the level of inventory stock, etc. These data are called parameters and represented as \( P = \{P_1, P_2, \ldots\} \). The parameters collected by different information gathering agents may be different.

Assume there are \( k \) parameters to be collected: \( P = \{P_1, P_2, \ldots, P_k\} \), and \( m \) information gathering agents are asked to collect the \( k \) parameters independently. The gathered results are \( \{P_{i1}, P_{i2}, \ldots, P_{ik}\} \) (\( i = 1, 2, \ldots, m \)). The first aggregation
CHAPTER 8. DECISION AGGREGATION IN MAS

problem is to combine \( \{P_{i1}, P_{i2}, \ldots, P_{ik}\} \) \( (i = 1, 2, \ldots, m) \) together in some reasonable way to obtain \( P = \{P_1, P_2, \ldots, P_k\} \).

Now suppose there are \( n \) decision making agents. Each agent has rules in its knowledge base such as

\[
\text{If } RT \text{ is } H \text{ and } P_1 \text{ is } B_1 \text{ and } \ldots \text{ then } IP \text{ is } C_i
\]  

(8.1)

where \( C_i (i = 1, 2, \ldots, n) \) is a fuzzy subset indicating the aggression or conservation of the investment policy.

Because the knowledge of the decision making agents and their decision attitudes may be different, the answers to the same question may also be different. Usually they are either close or conflicting to various degrees. They have to be combined or reconciled in order to produce one decision.

There are many aggregation algorithms. Different aggregation algorithms are needed in different applications. In this financial application, there exists much fuzzy or uncertain information. So approaches are needed that can deal with fuzzy or uncertain information to do the aggregation. There are three main classes of aggregation operations that can deal with fuzzy or uncertain information – the \( t \)-norm and \( t \)-conorm, generalized means, and ordered weighted averaging (OWA). The three classes of aggregation operations are outlined and compared with each other in the next section.

8.2 Choosing Appropriate Aggregation Operations

Aggregation operations on fuzzy sets are operations by which several fuzzy sets are combined in a desirable way to produce a single fuzzy set. Formally, any aggregation operation on \( n \) fuzzy sets \( (n \geq 2) \) is defined by a function

\[
h : [0, 1]^n \to [0, 1].
\]

When applied to fuzzy sets \( A_1, A_2, \ldots, A_n \), defined on \( X \), function \( h \) produces an aggregated fuzzy set \( A \) by operating on the membership grades of these sets for each \( x \in X \). Thus,

\[
A(x) = h(A_1(x), A_2(x), \ldots, A_n(x))
\]
for each \( x \in X \).

In order to qualify as an intuitively meaningful aggregation function, \( h \) must satisfy at least three axiomatic requirements, which express the essence of the notion of aggregation [121] (pp. 88-94):

- **Axiom 1.** \( h(0, 0, \ldots, 0) = 0 \) and \( h(1, 1, \ldots, 1) = 1 \) (boundary conditions).

- **Axiom 2.** For all pairs \( (a_1, a_2, \ldots, a_n) \) and \( (b_1, b_2, \ldots, b_n) \) of \( n \)-tuples such that \( a_i, b_i \in [0, 1] \) for all \( i = 1, 2, \ldots, n \), is \( a_i \leq b_i \) for all \( i = 1, 2, \ldots, n \), then
  \[
  h(a_1, a_2, \ldots, a_n) \leq h(b_1, b_2, \ldots, b_n);
  \]
  that is, \( h \) is monotonic increasing in all its arguments.

- **Axiom 3.** \( h \) is a continuous function.

Besides these essential and easily understood requirements, aggregating operations on fuzzy sets are usually expected to satisfy the following additional axiomatic requirement:

- **Axiom 4.** \( h \) is an idempotent; that is, \( h(a, a, \ldots, a) = a \) for all \( a \in [0, 1] \).

Axiom 4 expresses our intuition that any aggregation of equal fuzzy sets should result in the same fuzzy set. Observe that Axiom 4 subsumes Axiom 1.

Thus far, researchers have invested significant resources in devising and evaluating a large array of aggregation algorithms on fuzzy sets [122, 124]. Among them, there are three main classes [121, 123].

The first class of aggregation operators are the \( t \)-norm and \( t \)-conorm, which generalize the fuzzy intersections and unions. However, they are not idempotent, with the exception of the standard min and max operations.

Another class of aggregation operations that covers the entire interval between the min and max operations consists of generalized means. These are defined by the formula

\[
h_\alpha(a_1, a_2, \ldots, a_n) = \left( \frac{a_1^\alpha + a_2^\alpha + \cdots + a_n^\alpha}{n} \right)^{1/\alpha},
\]

where \( \alpha \in (-\infty, +\infty)(\alpha \neq 0) \) and \( a_i \neq 0 \) for all \( i \in \{1, 2, \ldots, n\} \) when \( \alpha < 0; \alpha \) is a parameter by which different means are distinguished. For \( \alpha < 0 \) and \( a_i \rightarrow 0 \)
for any $i \in \{1, 2, \ldots, n\}$, it is easy to see that $h_\alpha(a_1, a_2, \ldots, a_n)$ converges to 0. For $\alpha \to 0$, the function $h_\alpha$ converges to the geometric mean, $(a_1 a_2 \ldots a_n)^{1/n}$. One typical type of generalized means is (weighted) fuzzy averaging [46].

The third class of aggregation operations that covers the entire interval between the min and max operations is OWA aggregation operators, which are also a type of mean operators [125].

One typical aggregation operation is chosen from the second class (fuzzy averaging) and one from the third class (OWA) to aggregate different decisions in the financial investment planning prototype. Based on the aggregated results as well as other features of these two types of aggregation operations, we finally chose OWA in the application.

More detailed introductions to these two aggregation operations are given and comparisons between their aggregated results are presented in the next subsections.

### 8.2.1 Fuzzy Averaging

When dealing with fuzzy subsets, it is necessary to determine the membership functions of fuzzy subsets. In many applications, the membership functions can be represented by triangular fuzzy numbers or trapezoidal fuzzy numbers. Without loss of generality, one can assume that the membership functions used in this example are represented by trapezoidal fuzzy numbers. A trapezoidal fuzzy number $A$ or shortly trapezoidal number can be denoted by a 4-tuple, $A = (a_1, b_1, b_2, a_2)$. The trapezoidal average formulas used here are as follows:

- **Trapezoidal average formula**

  If $A_i = (a_{i1}, b_{i1}, b_{i2}, a_{i2}), i = 1, 2, \ldots, n$ are trapezoidal numbers, then

  $$
  A_{\text{ave}} = (m_1, m_{M_1}, m_{M_2}, m_2) = \left(\frac{\sum_{i=1}^{n} a_{i1} + \sum_{i=1}^{n} b_{i1} + \sum_{i=1}^{n} b_{i2} + \sum_{i=1}^{n} a_{i2}}{n}\right) \tag{8.2}
  $$

- **Weighted trapezoidal average formula**

  If the real numbers $w_i$ represent the importance of $A_i = (a_{i1}, b_{i1}, b_{i2}, a_{i2}), i = 1, 2, \ldots, n$, then we obtain the weighted trapezoidal average,

  $$
  A_{\text{ave}}^{w} = (m_1^{w}, m_{M_1}^{w}, m_{M_2}^{w}, m_2^{w}) = \left(\sum_{i=1}^{n} w_i a_{i1}, \sum_{i=1}^{n} w_i b_{i1}, \sum_{i=1}^{n} w_i b_{i2}, \sum_{i=1}^{n} w_i a_{i2}\right) \tag{8.3}
  $$
The trapezoidal average and weighted average formulas (8.2)-(8.3) produce a result which can be interpreted as follows. It is a conclusion or aggregation of all combined meanings expressed by trapezoidal numbers \( A_1, \ldots, A_n \) considered either of equal importance or of different importance expressed by weights \( w_i \).

The aggregation defined by the trapezoidal average number very often has to be expressed by a crisp value which best represents the corresponding average. This operation is called defuzzification. There are many options for defuzzifying \( A_{ave} = (m_1, m_{M1}, m_{M2}, m_2) \) such as center of area method (CAM), mean of maximum method (MMM), and height defuzzification method (HDM) [46] etc. The most frequently used defuzzification method is CAM. However, the required computations are sometimes complex. MMM is the simplest among the three. HDM could be considered as both a simplified version of CAM and a generalized version of MMM. As only the relative values of aggregated results are important, we adopted MMM and the formula is as follows:

\[
x_{max} = \frac{m_{M1} + m_{M2}}{2}.
\]

(8.4)

Recall the decision aggregation problem described in Section 8.1. The words aggressive and conservative in Section 8.1 are linguistic variables, i.e., fuzzy concepts. The decision making agents (financial experts) dealing with the investment model agree to describe aggressive (aggressive investment policy) by a suitable (left) trapezoidal number on a scale from 0 to 100 (universal set—the interval \([0, 100]\)) and conservative by a (right) trapezoidal number on a scale from -100 to 0 (universal set \([-100, 0]\)). The numbers on the joined scale from -100 to 100 have a certain meaning accepted by the agents. For instance 50 and -50 can be interpreted as indicators for moderately aggressive investment and moderately conservative investment, correspondingly; 70 and -70 as aggressive and conservative investment, etc.

Without loss of generality, assume that there are three decision making agents \( DA_i (i = 1, 2, 3) \). They use their rules like (8.1) and obtain their own decisions \( C_i (i = 1, 2, 3) \) independently. These decisions are represented by trapezoidal numbers as follows (refer to Figure 8.1):
CHAPTER 8. DECISION AGGREGATION IN MAS

\[ C_1 = (-100, -100, -50, -30), C_2 = (-10, 10, 10, 30), C_3 = (60, 90, 100, 100); \]

\[ (8.5) \]

\( C_1 \) indicates conservative, \( C_2 \) slightly aggressive, and \( C_3 \) aggressive. If all the agents have equal importance in the decision making process, the aggregation according to formula (8.2) produces

\[ C_{ave} = C_1 + C_2 + C_3 = (-16.67, 0, 20, 33.33). \]

The defuzzification value according to formula (8.4) is \( \frac{9 + 20}{2} = 10 \). It suggests a policy on the aggressive side of the scale but a very cautious one.

Now consider the case when the opinions of the three conflicting agents have different importance on a scale from 0 to 10. The ranking of agents \( DA_1, DA_2, \) and \( DA_3 \) is assumed to be \( r_1 = 4, r_2 = 6, \) and \( r_3 = 10 \), correspondingly. The degrees of importance are mapped into unit interval, and obtain \( U = [u_1, u_2, u_3] = [0.2, 0.3, 0.5] \). That is the weights \( w_i \) for \( DA_i \): \( w_1 = 0.2, w_2 = 0.3, w_3 = 0.5 \). Using formula (8.3) to aggregate the conflicting agents' opinions gives

\[ C_{ave}^w = 0.2C_1 + 0.3C_2 + 0.5C_3 = (7, 28, 43, 53) \]

whose defuzzified value is 35.54. It indicates that the investment policy should be cautiously aggressive.

There is some difference between \( C_{ave} \) and \( C_{ave}^w \), and also between the defuzzified values 10 and 35.54 due to the high ranking of agent \( DA_3 \) who favors an aggressive investment policy.
8.2.2 Ordered Weighted Averaging

Yager introduced the OWA operator to provide a family of aggregators having the properties of mean operators [125, 126, 127].

A mapping $F : \mathbb{R}^n \rightarrow \mathbb{R}$ is called an OWA operator of dimension $n$ if it has an associated weighting vector $W$ of dimension $n$ such that its components satisfy

1. $w_j \in [0, 1]$;
2. $\sum_{j=1}^n w_j = 1$; and
3. $F_w(a_1, a_2, \ldots, a_n) = \sum_{j=1}^n w_j b_j$, where $b_j$ is the $j$th largest of the $a_i$.

A fundamental feature of this operator is the reordering process which associates the arguments with the weights. This aggregation can be expressed in a vector notation as $F_w(a_1, a_2, \ldots, a_n) = W^T B$. In this expression, $W$ is the OWA weighting vector associated with the aggregation, and $B$ is the ordered argument vector; where the $j$th component in $B$, $b_j$ is the $j$th largest of the $a_i$.

Expressing the OWA operator $F_w(a_1, a_2, \ldots, a_n)$ in its vector notation form $W^T B$ makes very clear the distinct components involved in the performance of this operation. First, there is a weighting vector $W$; this is required to have components $w_j$ which lie in the unit interval and sum to one. The second part of the OWA aggregation is the vector $B$, called the ordered argument vector. This vector is composed of the arguments of the aggregation. To solve a specific problem using the OWA operator, one needs to find out the appropriate weighting vector $W$ and the ordered argument vector $B$.

There are two characterizing measures associated with the weighting vector $W$ [125, 126].

The first of these, the $\alpha$ value of an OWA operator, is defined as

$$\alpha(W) = \frac{1}{n-1} \sum_{j=1}^n w_j (n - j).$$ (8.6)

This measure, which takes its values in the unit interval, is determined by the weighting used in the aggregation. The actual semantics associated with $\alpha$ is dependent upon the application in which it is being used. In our case, the $\alpha$ can be the degree that the aggregation prefers decisions with high confidence, or the attitude of the decision making agent.
CHAPTER 8. DECISION AGGREGATION IN MAS

The second measure is called the dispersion (or entropy) of $W$ and is defined as

$$H(W) = -\sum_{j=1}^{n} w_j \ln(w_j).$$

It was shown that this helps measure the degree to which $W$ takes into account all of the information in the aggregation.

One method to determine these weights, $w_1, \ldots, w_n$, requires the solution of the following mathematical programming problem:

Maximize $-\sum_{j=1}^{n} w_j \ln(w_j)$ subject to

1. $\alpha(W) = \frac{1}{n-1} \sum_{j=1}^{n} w_j (n - j)$;
2. $w_j \in [0, 1]$;
3. $\sum_{j=1}^{n} w_j = 1$.

Assume that the agents' decisions are still represented by trapezoidal numbers.

If $C_i = (a_{i1}, b_{i1}, b_{i2}, a_{i2}), i = 1, 2, \ldots, n$, are trapezoidal numbers, then

$$C_{OWA} = (F_w(a_{11}, \ldots, a_{n1}), F_w(b_{11}, \ldots, b_{n1}), F_w(b_{12}, \ldots, b_{n2}), F_w(a_{12}, \ldots, a_{n2}))$$

(8.7)

where $F_w$ is an OWA operator. How to decide the weighting vector $W$ and the ordered argument vector $B$ in different situations when aggregating use formula (8.7) is now discussed.

Suppose that the three agents still present their decisions on the investment policy by the fuzzy numbers in (8.5). Corresponding to the first case in Section 8.2.1, the weighting vector $W = [w_1, w_2, w_3] = [1/3, 1/3, 1/3]$. The arguments are ordered by their values.

Corresponding to the weighted case, if the arguments are ordered using the values of $r_i$, i.e., let $b_j$ be the $a_i$ value which has the $j$th largest of $r_i$, and let $W = [w_1, w_2, w_3] = [u_3, u_2, u_1] = [0.5, 0.3, 0.2]$. Formula (8.7) is then used to aggregate. In both cases, the same results are obtained as those using fuzzy averaging.

The problem here is that the degrees of importance in aggregation were not used directly. Actually in this case, the arguments needed to aggregate are the pairs such as

$$(u_1, a_{11}), (u_2, a_{21}), \ldots, (u_n, a_{n1}).$$

Here, the formula $G(u, a) = \delta a + u a$ is used to transform the tuple into a single
value \([127]\) (pp. 41-49), where \(\alpha\) is defined by (8.6). The following are the steps of the procedure:

1. Calculate the \(\alpha\) value of the OWA operator:
\[
\alpha = \sum_{j=1}^{3} \frac{3-j}{3-1} w_j = w_1 + w_2/2 = 0.5 + 0.3/2 = 0.65.
\]

2. Transform each of the argument tuples using \(G(u_j, a_j) = \bar{\alpha} a_j + u_j a_j\), hence
\[
G(u_1, a_{11}) = -49.72, G(u_2, a_{21}) = -2.755, G(u_3, a_{31}) = 30.175,
\]
\[
G(u_1, b_{11}) = -49.72, G(u_2, b_{21}) = 3.245, G(u_3, b_{31}) = 45.175,
\]
\[
G(u_1, b_{12}) = -9.72, G(u_2, b_{22}) = 3.245, G(u_3, b_{32}) = 50.175,
\]
\[
G(u_1, a_{12}) = -5.72, G(u_2, a_{22}) = 9.245, G(u_3, a_{32}) = 50.175.
\]

3. Calculate \(C_{OWA}\)
\[
C_{OWA} = (F_w(-49.72, -2.755, 30.175), F_w(-49.72, 3.245, 45.175),
\]
\[
F_w(-9.72, 3.245, 50.175), F_w(-5.72, 9.245, 50.175))
\]
\[
\]

The defuzzification value is 18.87. This still indicates a very cautious investment policy – much more cautious than that not using the degrees of importance.

The concept of agents’ decision making attitudes is also important. Because the agents usually have different knowledge, this results in different attitudes when making decisions. Some are aggressive, some conservative. \(\alpha_i (\alpha_i \in [0, 1])\) is used to indicate the agents’ attitudes. The bigger the value of \(\alpha_i\), the more aggressive of the attitude of the decision making agent \(DA_i\).

Suppose there are still three agents, and their attitudes are \(\alpha_1 = 0.3, \alpha_2 = 0.5,\) and \(\alpha_3 = 0.8\). The decisions they made and their degrees of importance remain unchanged as described above.

To aggregate, the first step is to decide the attitude \(\alpha\) of all the agents (here three). The OWA operator is still used. The degrees of importance are mapped to unit interval as the weighting vector for combining \(\alpha_i\), called \(W(\alpha)\).
\[
W(\alpha) = [w(\alpha)_1, w(\alpha)_2, w(\alpha)_3] = [0.5, 0.3, 0.2].
\]
CHAPTER 8. DECISION AGGREGATION IN MAS

Then
\[ \alpha = F_w(\alpha_1, \alpha_2, \alpha_3) = \alpha_3 \times w(\alpha)_1 + \alpha_2 \times w(\alpha)_2 + \alpha_1 \times w(\alpha)_3 = 0.61. \]

By solving the mathematical programming problem with \( \alpha = 0.61 \), the weighting vector \( W \) is obtained for the final aggregation as follows:
\[ W = [w_1, w_2, w_3] = [0.45, 0.32, 0.23]. \]

The arguments are ordered according to the values of \( r_i \). The final aggregation using (8.7) gives \( C_{OWA} = (0.8, 20.7, 36.7, 47.3) \). The defuzzification value according to (8.4) is 28.7. It suggests a policy on the aggressive side of the scale but a cautious one – more cautious than that using fuzzy averaging. This is because the decision attitude of \( DA_1 \) is slightly conservative, but its decision is very conservative. Taking all the information into account, the investment policy should be cautiously aggressive.

If the degrees of importance are used directly in the aggregation in this case, \( C_{OWA} = (1.26, 9.93, 21.38, 24.22) \) is obtained. The defuzzification value is 15.66.

8.2.3 Discussion

From the above description, it is easy to see that the fuzzy averaging is a special case of OWA operator. Based on the definition of OWA, one knows that using OWA to aggregate can take much more information into account than using other methods.

When using OWA operators to evaluate the worth of each of the alternatives, it is easy to model the decision makers' (agents') attitude that affects the decision. Central to OWA is the use of a weighted average operation. The OWA can also be extended to deal with the situation in which the payoffs are non-numeric values – which is often the case in our application. There are usually two scales for representing non-numeric information. One scale assumes only the existence of a linear ordering on the allowable values. A second scale, called a uniform scale, assumes in addition to a linear ordering, the scale values are uniformly spaced. In the case of only a linear ordering on the payoffs one can use a weighted median operation to replace the weighted average. In the case in which the payoffs come
from a uniform scale, an operation called the weighted average on a uniform scale is used to replace the weighted average [128]. That is why OWA was chosen in our application.

8.3 Approaches to Implementing Decision Aggregation in Multi-Agent Systems

After choosing the appropriate aggregation operation for a specific application, the next step is to implement the operation in MASs. Although much work has been done on the aggregation algorithms, little is involved in the implementations of these aggregation algorithms in MASs. From the implementation point of view, it is hard to find information about which kinds of approaches can be used in which problem scales. It is the lack of efficient and effective practical implementations that motivated us to explore some flexible ways to implement aggregation algorithms in MASs.

To implement decision aggregation in MASs, two steps are involved. One is to collect decisions from different decision making agents. The other is the aggregation itself: to reconcile these decisions to produce one final decision. Three ways are proposed – the stationary agent approach, the token passing approach, and the mobile agent approach – to implement aggregation algorithms in MASs. We will also evaluate the three approaches by their scalability and flexibility, response time, and the degree of difficulty with implementation.

8.3.1 The Stationary Agent Approach

The idea of this approach is to exploit a special stationary agent, called aggregation agent, to do the job. Refer to Figure 4.15 for the internal structure of the aggregation agent.

The decision gathering process is as follows. All the decision making agents inform the aggregation agent by sending a KQML message with their decisions, degrees of importance, and decision attitudes etc. The experimental system is under the support of JATLite. Figure 8.2 shows the architecture.
The scenario goes as follows. The aggregation agent presents a time limitation for all registered and connected decision making agents and informs them that they must send back their decisions to the aggregation agent within the limitation. After the preset time interval elapses, the aggregation agent sends a time-out message to those decision making agents who fail to send back the results within the limitation. The aggregation agent waits for the response of some special cases (e.g., resent results by some decision making agents). The aggregation then begins with the received results.

The implementation is very general. In this experimental system, the OWA aggregation algorithm is used. Actually, the OWA can be substituted with other aggregation algorithms in the aggregation component of the aggregation agent. Thus, one can easily implement other aggregation algorithms.

When the number of agents in a multi-agent system is large (≥ 40 in JATLite environment\(^1\)), this solution produces high network traffic, makes the aggregation agent a bottleneck during decision collecting and requires stable network connections. This prevents one from applying this method to large-scale multi-agent systems. To overcome this drawback, we developed a second approach – the token passing mechanism.

\(^1\)Based on our test, if there are more than 40 agents registered and connected at the same time, the system becomes so slow that it is intolerable.
8.3.2 The Token Passing Approach

For the second approach, we borrowed the “token passing” idea from Token Ring Network [129]. The token passing idea itself is not new, but we first introduced it into the implementation of multi-agent decision aggregation. We constructed a virtual decision making agent ring (Figure 8.3) which is under the support of the JATLite Message Router. The “token” passed here is a KQML message. Each decision making agent is equipped with an aggregation module (in our experimental system, an OWA aggregation module).

![Diagram](Image)

Figure 8.3: Aggregation by Token Passing

Assume there are \( n \) decision making agents. Each agent registers and connects to the Router with a unique number as its name. The format of the token (KQML message) used is as follows:

```prolog
(tell
   :sender
   :receiver
   :counter
   :status
   :content
)
```
CHAPTER 8. DECISION AGGREGATION IN MAS

If sender is agent $i$, then receiver is $(i + 1) \mod n$. The status consists of a binary string with length $n$ and its initial value is $0 \ldots 0$. 0 indicates that the corresponding agent has not added its decision to content. The content is an array consisting of $n$ 3-tuples such as [decision, importance degree, decision attitude]. At the beginning, all the 3-tuples are empty.

At first, the Router initializes the token (KQML message) and sends it to the next agent. After that, when an agent receives the token and finishes its decision making, it adds its own information to the token (the content part of the KQML message) and passes it to the next agent. The process continues until the last agent receives the token. After the last agent finishes its decision making, it adds its information to the token and activates the aggregation module. The algorithm is as follows:

**Algorithm** Decision Aggregation:

1. The Router sends the initial KQML message with status $= 0 \ldots 0$, content $= null$.

2. When an agent receives the token, it does the following:

   KQML message kqml = mail.getKQMLmessage(); // receive the KQML message
   int sender = kqml.getValue("receiver");
   String status = kqml.getValue("status");
   if (sender is ready) and (the corresponding bit in status is '0')
   // agent ready means it has completed its decision making
   { receiver = (sender + 1) mod n;
   Change the corresponding bit in status to "1";
   Add the 3-tuple to content;
   }
   if (all the bits in status are '1')
   CALL OWA_aggregation_processing;
To implement other aggregation algorithms,

one only needs to change the above line.

else

send out the new KQML message;

When implementing this algorithm, a virtual agent ring maintenance component was added to the Router. In this way, an agent can be added to or deleted from the virtual ring dynamically.

In this solution, there is no bottleneck for decision gathering as in the first solution. One can still implement other aggregation algorithms, but at this time, one must add the aggregation module to all decision making agents. Compared with the first approach, there is more work to do. On the other hand, this solution demonstrates some kind of inefficiency if many decision making agents finish their decision making at the same time as an agent (after it finishes its decision making) must wait for the token to come around.

Can an approach be found that has all the advantages of the two approaches discussed, and at the same time, overcomes the shortcomings of the two methods? The answer is positive. This leads to the third approach – aggregation by employing mobile agents.

8.3.3 The Mobile Agent Approach

A mobile agent is a self-contained software element responsible for executing a program process, which is capable of autonomously migrating through a network between logical "places" or "contexts". Mobile agents have the unique ability to transport themselves from one system in a network to another.

We developed the mobile aggregation agent using JATLite and Aglets SDK (http://www.trl.ibm.co.jp/aglets/ and [130]). The kernel part of the mobile aggregation agent is the same as that of the stationary aggregation agent, but "state" and "identifier" are added for the purpose of mobility (Figure 8.4). "State" is needed for the agent to resume computation after traveling. "Identifier" is needed to recognize and locate the traveling aggregation agents. The "place" in Figure
8.4 is a context in which an agent can execute. It is a stationary object that provides a means for maintaining and managing running aglets in a uniform execution environment.

In this method, mobile agents are used to collect different decisions. The aggregation itself stays unchanged.

![Diagram of Mobile Aggregation Agent Model](image)

Figure 8.4: Mobile Aggregation Agent Model

The mobile aggregation approach preserves all the advantages of the first two methods. Because its kernel part is the same as that of the first implementation, it is very flexible to implement other aggregation algorithms.

Before aggregating, the mobile aggregation agent moves to the hosts in which the decision making agents reside instead of all decision making agents sending results to the aggregation agent. Thus, it does not require continuous network connections, and the decision gathering bottleneck no longer exists in this situation.

Because mobile agents can create a cascade of clones in the network, if there are many decision making agents finishing their decision making at the same time, the cloned aggregation agents can be dispatched to the network parallel collecting the decisions of different decision making agents. The latency in the second case is eliminated.
Security is a big issue in any practical application of mobile agents [130, 131]. There is more work to do in the implementation compared with that of the stationary agent situation.

8.3.4 Comparison of the Three Approaches

We tested the three approaches. The experiments were under the support of JATLite and Aglets. The following is the evaluation of these approaches based on the criteria mentioned at the beginning of this section.

- **Scalability and flexibility** All three approaches proposed in this chapter have scalability and are flexible to implement other aggregation algorithms. In fact, they are "aggregation algorithm independent". When we conducted the experiments, we only implemented the OWA aggregation, but other aggregation algorithms can be easily added to the stationary and mobile aggregation agents. In the token passing method, if one wants to use other aggregation algorithms, one needs to add the aggregation module to all the decision making agents. Thus, the second approach is not so scalable and flexible as the first and the third ones. It is only practical in small-scale systems.

- **Response time** When the number of decision making agents is small, there is no obvious performance difference among the three approaches. There are significant differences in performance when the number is large.

By cloning the decision making agents, we tested the performance of the first approach. With the number of decision making agents increasing, the performance drops. When the number of decision making agents is increased to 40, the response of the aggregation agent is so slow that it becomes intolerable in the experimental environment (10M/100M Ethernet LAN). This implies that the stationary aggregation agents become a bottleneck of the system when the number of decision making agents is large and they send KQML messages to the aggregation agent simultaneously (the number of agents that make the response time intolerable may vary in different environments).

The second method does not produce high network traffic, but if many decision making agents finish their decision making at the same time, they must
wait for the token to come around. This results in some kind of inefficiency of the system performance.

For the third alternative, the mobile aggregation agent moves to the hosts where the decision making agents reside instead of all decision making agents sending results to the aggregation agent. Thus, the decision gathering bottleneck no longer exists. If there are many decision making agents who finish their decision making at the same time, the mobile aggregation agent can clone as many as needed and dispatch them to the corresponding decision making agents. They then collect the decisions in parallel. Hence, the mobile agent approach is the most efficient one in large-scale systems.

- **The degree of difficulty with implementations** The stationary agent approach is the easiest one to implement. Compared with the first one, the second approach has more work to do when implementing because of the virtual agent maintenance. The mobile agent approach is the most difficult one to implement because one must pay more attention on the security problems.

The evaluation results of these approaches are summarized in Table 8.1 (1: stationary agent approach, 2: token-passing approach, 3: mobile agent approach).

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Order (Good→bad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability and Flexibility</td>
<td>3 1 2</td>
</tr>
<tr>
<td>Response time</td>
<td>3 2 1</td>
</tr>
<tr>
<td>The degree of difficulty with implementations</td>
<td>1 2 3</td>
</tr>
</tbody>
</table>

8.4 Case Study: Financial Portfolio Selection

Portfolio selection is about how to determine a most suitable portfolio for the large private or institutional investor. It is a key step in financial investment. There are many portfolio selection models available. It is too difficult to choose the “best”
model to use in real-world financial investment applications. Each model has its own strengths and weaknesses. One model may put more emphasis on some factors or attributes in the portfolio selection process but ignores others. No single model can take all the factors into consideration.

In this case study, three typical models are employed – Markowitz’s model [132], the fuzzy probability model, and the possibility portfolio selection model [133, 134] to select portfolios independently based on the data. The OWA operator is then used to aggregate these three different portfolios to a final one.

In this section, there is first an introduction to the three portfolio selection models. Then the agent architecture for portfolio selection is discussed. Finally, some experimental results are presented.

8.4.1 Portfolio Selection Models

After H. Markowitz proposed the first portfolio selection model in the 1950s [132], many portfolio selection models appeared based on different techniques. Here a brief introduction of three typical models – Markowitz’s model, the fuzzy probability model, and the possibility portfolio selection model – is given. Markowitz’s model is based on a probability distribution. The fuzzy probability model can be regarded as a natural extension of Markowitz’s model because of extending probability into fuzzy probability. The possibility portfolio selection model is based on a possibility distribution that is used to characterize experts’ knowledge. A possibility distribution is identified using the returns of securities associated with possibility grades offered by portfolio experts.

Markowitz’s Portfolio Selection Model

Assume that there are \( n \) securities denoted by \( S_j (j = 1, \ldots, n) \). The return of the security \( S_j \) is denoted as \( r_j \) and the proportion of total investment funds devoted to this security is denoted as \( x_j \). Thus, the equation \( \sum_{j=1}^{n} x_j = 1 \) holds.

Since the returns of the securities \( r_j (j = 1, \ldots, n) \) vary from time to time, those are assumed to be random variables which can be represented by the pair of the average vector and the covariance matrix. For instance, it is assumed that the
observation data on returns is denoted as \( r_i = [r_{i1}, ..., r_{in}]^t \). Thus, the total data over \( m \) periods are denoted as the following matrix:

\[
\begin{pmatrix}
  r_{11} & r_{12} & \cdots & r_{1n} \\
  r_{21} & r_{22} & \cdots & r_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  r_{m1} & r_{m2} & \cdots & r_{mn}
\end{pmatrix},
\]

where \( r_{ij} \) denoting the return of the \( j \)th security at the time \( i \) is defined as (Closing price of the \( j \)th security at the time \( i \)) minus (its closing price at time \( i - 1 \)) plus (Its dividends at the time \( i \)) all divided by (Its closing price at time \( i - 1 \)). The average vector of returns over \( m \) periods denoted as \( r^0 = [r^0_1, ..., r^0_n]^t \) is defined as

\[
r^0 = \left[ \frac{\sum_{i=1}^{m} r_{i1}}{m} \right. \left. \cdots \right. \left. \frac{\sum_{i=1}^{m} r_{in}}{m} \right].
\]

Also the corresponding covariance matrix \( Q = [q^2_{ij}] \) is defined as \( q^2_{ij} = \sum_{k=1}^{m} (r_{ki} - r^0_i)(r_{kj} - r^0_j)/m (i = 1, \ldots, n, j = 1, \ldots, n) \).

Therefore, random variables can be represented by the average vector \( r^0 \) and the covariance matrix \( Q \), denoted as \( (r^0, Q) \). Now, the return associated with a portfolio \( x = [x_1, ..., x_n]^t \) is given by \( z = x^t r \). The average and variance of \( z \) are given as:

\[
E(z) = E(x^t r) = x^t E r = x^t r^0,
\]

\[
V(z) = V(x^t r) = x^t Q x.
\]

Since the variance of a portfolio return is regarded as the risk of investment, the best investment is one with the minimum variance subject to a given average return \( r_s \). This is the famous Markowitz’s model [1]. It can be formalized as the following quadratic programming (QP) problem:

\[
\begin{align*}
\text{Minimize} & \quad x^t Q x \\
\text{subject to} & \quad x^t r^0 = r_s, \sum_{i=1}^{n} x_i = 1, x_i \geq 0.
\end{align*}
\]

The Fuzzy Probability Portfolio Selection Model

In this model, the data are given as \((r_i, h_i)(i = 1, \ldots, m)\) where \( h_i \) is a possibility grade to reflect a similarity degree between the future state of stock markets and
the state of the $ith$ sample offered by experts. These grades $h_i(i = 1, \ldots, m)$ are
graded as weights to determine the fuzzy average vector and covariance matrix in
fuzzy probabilities.

Given the data $(r_i, h_i)(i = 1, \ldots, m)$, the fuzzy weighted average vector $\alpha = [
\alpha_1, \ldots, \alpha_n]^t$ can be defined as follows:

$$\alpha = \frac{\sum_{i=1}^{m} (h_i r_i)}{\sum_{i=1}^{m} h_i}.$$ 

Similarly, the fuzzy weight covariance matrix $\Sigma = [\sigma_{ij}]$ can be defined by

$$\sigma_{ij} = \frac{\sum_{k=1}^{n} (r_{ki} - \alpha_i)(r_{kj} - \alpha_j)h_k}{\sum_{k=1}^{n} h_k (i = 1, \ldots, n, j = 1, \ldots, n)}.$$ 

Thus, the given data $(r_i, h_i)(i = 1, \ldots, m)$ can be summarized as the parametric
representation $(\alpha, \Sigma)$, which is used to construct the fuzzy portfolio selection model.

Given the weight average vector and covariance matrix, $(\alpha, \Sigma)$, the average and
covariance of the return $z$ are given as follows:

$$E(z) = x^t \alpha,$$

$$V(z) = x^t \Sigma x.$$ 

Thus, the fuzzy probability portfolio selection problem can be obtained as:

$$\text{Min}_x \ x^t \Sigma x$$

s.t. $x^t \alpha = r_s$, $\sum_{i=1}^{n} x_i = 1$, $x_i \geq 0$.

It should be noted that the average vector and covariance matrix in Markowitz's
model are replaced by the weight-average vector and covariance, respectively, in
which the expert judgment $h_i$ is contained. It is still a QP problem.

The Possibility Portfolio Selection Model

Assume that the returns $r_i(i = 1, \ldots, m)$ are governed by a possibility distribution.
The possibility distribution denoted as $\Pi_A(r) = (\alpha, D_A)$, where $\alpha$ is a center
vector and $D_A$ is a symmetric positive-definite matrix. The possibility return of a
portfolio $x = [x_1, \ldots, x_n]^t$ can be written as $z = x^t r$.

The possibility distribution of $Z$, denoted as $\Pi_Z(z)$, can be defined by the
extension principle as follows:

$$\Pi_Z(z) = \text{Max}_{(r'|x = x^t r') \Pi_A(r')}.$$
Solving this simple optimization problem produces the following:

$$\Pi_Z(z) = \exp\{-(z - x^i\alpha)^2 \cdot (x^iD_Ax)^{-1}\},$$

where $x^i\alpha$ is the center value and $x^iD_Ax$ is the spread of the possibility return $Z$. Following Markowitz's model, the following possibility portfolio selection model is given:

$$\min_x x^iD_Ax$$

s.t. $x^i\alpha = r_Z, \sum_{i=1}^{n} x_i = 1, x_i \geq 0$, which is also a QP problem minimizing the spread of possibility return subject to a given center return $r_Z$.

### 8.4.2 Agent Architecture for Portfolio Selection

Follow the methodology in Chapter 3 and based on the architecture of the agent-based intelligent technique society (Figure 4.11), it is easy to determine the architecture for portfolio selection. Corresponding to the **HELP PROVIDER** role, there are three agents: the **Markowitz Model Agent**, the **Fuzzy Model Agent**, and the **Possibility Model Agent**. In the three portfolio selection models, the final step is the same: solving a QP problem, but with a different average return vector and covariance matrix. With these observations in mind, one QP problem solving agent can provide help for the three portfolio model agents. The three portfolio selection agents are responsible for preparing the average return vector and covariance matrix based on the information of the past performance of individual securities and the information of the beliefs of one or more security analysts concerning future performances. Of course, the interface agent, the planning agent, the aggregation agent, and the serving agent are still needed. When all these are put together, they result in the architecture for portfolio selection (Figure 8.5). Such an architecture is very adaptive and robust: (1) One can dynamically add new models to or delete unwanted models from the system; (2) If one model fails, a solution can still be found by using other models.

The practical architecture under the support of JATLite is shown in Figure 8.6. We downloaded the *Lingo* software package (evaluation version, [http://www.lindo.com](http://www.lindo.com)) as the QP problem solver. This time we used a KQML-speaking agent transducer as the bridge between *Lingo* program and the agent system.
8.4.3 Experimental Results

We conducted experiments with nine securities (listed in New York Stock Exchange) numbered from $S_1$ to $S_9$. The returns of the nine securities in the past 18 years are listed in Table 8.2. These data are based on the past performance of individual securities retrieved by information gathering agents.

Assume the investor provides the interface agent with the following personal information:

- Amount of money to invest: $100,000
- Annual income: $65,000
- Total net-worth: $380,000
- Age: 30
### Table 8.2: Returns on Nine Securities

<table>
<thead>
<tr>
<th>Year</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$S_8$</th>
<th>$S_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1982</td>
<td>-.305</td>
<td>-.173</td>
<td>-.318</td>
<td>-.477</td>
<td>-.457</td>
<td>-.065</td>
<td>-.319</td>
<td>-.400</td>
<td>-.435</td>
</tr>
<tr>
<td>1984</td>
<td>.055</td>
<td>.200</td>
<td>.047</td>
<td>.165</td>
<td>.424</td>
<td>-.078</td>
<td>.381</td>
<td>-.093</td>
<td>-.295</td>
</tr>
<tr>
<td>1985</td>
<td>-.126</td>
<td>.030</td>
<td>.104</td>
<td>-.043</td>
<td>-.189</td>
<td>-.077</td>
<td>-.051</td>
<td>-.090</td>
<td>-.036</td>
</tr>
<tr>
<td>1986</td>
<td>-.280</td>
<td>-.183</td>
<td>-.171</td>
<td>-.277</td>
<td>.637</td>
<td>-.187</td>
<td>.087</td>
<td>-.194</td>
<td>-.240</td>
</tr>
<tr>
<td>1987</td>
<td>-.003</td>
<td>.067</td>
<td>-.039</td>
<td>.476</td>
<td>.865</td>
<td>.156</td>
<td>.262</td>
<td>1.113</td>
<td>.126</td>
</tr>
<tr>
<td>1988</td>
<td>.428</td>
<td>.300</td>
<td>.149</td>
<td>.225</td>
<td>.313</td>
<td>.351</td>
<td>.341</td>
<td>.580</td>
<td>.639</td>
</tr>
<tr>
<td>1990</td>
<td>.446</td>
<td>.216</td>
<td>.419</td>
<td>.216</td>
<td>.373</td>
<td>.349</td>
<td>.352</td>
<td>.229</td>
<td>.578</td>
</tr>
<tr>
<td>1991</td>
<td>-.088</td>
<td>-.046</td>
<td>-.078</td>
<td>-.272</td>
<td>-.037</td>
<td>-.209</td>
<td>.153</td>
<td>-.126</td>
<td>-.289</td>
</tr>
<tr>
<td>1992</td>
<td>-.127</td>
<td>-.071</td>
<td>.169</td>
<td>.144</td>
<td>.026</td>
<td>.355</td>
<td>-.099</td>
<td>.009</td>
<td>.184</td>
</tr>
<tr>
<td>1993</td>
<td>-.015</td>
<td>.056</td>
<td>-.035</td>
<td>.107</td>
<td>.153</td>
<td>-.231</td>
<td>.038</td>
<td>.000</td>
<td>.114</td>
</tr>
<tr>
<td>1994</td>
<td>.305</td>
<td>.038</td>
<td>.133</td>
<td>.321</td>
<td>.067</td>
<td>.246</td>
<td>.273</td>
<td>.223</td>
<td>-.222</td>
</tr>
<tr>
<td>1995</td>
<td>-.096</td>
<td>.089</td>
<td>.732</td>
<td>.305</td>
<td>.579</td>
<td>-.248</td>
<td>.091</td>
<td>.650</td>
<td>.327</td>
</tr>
<tr>
<td>1996</td>
<td>.016</td>
<td>.090</td>
<td>.021</td>
<td>.195</td>
<td>.040</td>
<td>-.064</td>
<td>.054</td>
<td>-.131</td>
<td>.333</td>
</tr>
<tr>
<td>1997</td>
<td>-.128</td>
<td>.083</td>
<td>.131</td>
<td>.390</td>
<td>.434</td>
<td>.079</td>
<td>.109</td>
<td>.175</td>
<td>.062</td>
</tr>
<tr>
<td>1998</td>
<td>-.010</td>
<td>.035</td>
<td>.006</td>
<td>-.072</td>
<td>-.027</td>
<td>.067</td>
<td>.210</td>
<td>-.084</td>
<td>-.048</td>
</tr>
<tr>
<td>1999</td>
<td>.154</td>
<td>.176</td>
<td>.908</td>
<td>.715</td>
<td>.469</td>
<td>.077</td>
<td>.112</td>
<td>.756</td>
<td>.185</td>
</tr>
</tbody>
</table>

Based on the inputs, the **Investment Policy Agent** (see Figure 4.18) can decide that the client's investment policy is relatively aggressive. Thus the client can invest in the stock market.

According to the security data as well as other relevant information, this prototype can select a portfolio from these nine securities that satisfies the investor's return level and minimizes the risk (variance). The following is the selected portfolio with an expected average return $r_s = 0.17$.

Based on the Markowitz model, the selected portfolio ($P_{MAR}$) consists of $S_3(25\%)$, $S_4(4\%)$, $S_5(16\%)$, and $S_7(55\%)$. This indicates that the client should invest $25,000 on $S_3$, $4,000 on $S_4$, $16,000 on $S_5$, and $55,000 on $S_7$.

Based on the fuzzy probability model, the portfolio ($P_{FUZ}$) consists of $S_3(8\%)$, $S_4(13\%)$, $S_5(20\%)$, $S_7(57\%)$, and $S_8(2\%)$. This indicates that the client should invest $8,000 on $S_3$, $13,000 on $S_4$, $20,000 on $S_5$, $57,000 on $S_7$, and $2,000 on $S_8$. 

Table 8.3: Portfolios Based on Three Models and Aggregated Portfolio

<table>
<thead>
<tr>
<th>Portfolios</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$S_8$</th>
<th>$S_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{MAR}$</td>
<td>25</td>
<td>4</td>
<td>16</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_{FUZ}$</td>
<td>8</td>
<td>13</td>
<td>20</td>
<td>57</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_{POS}$</td>
<td>12</td>
<td>26</td>
<td>8</td>
<td>23</td>
<td>22</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_{OWA}$</td>
<td>7</td>
<td>21</td>
<td>4</td>
<td>13</td>
<td>38</td>
<td>12</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Based on the possibility distribution model, the portfolio ($P_{POS}$) consists of $S_1(12\%)$, $S_3(26\%)$, $S_5(8\%)$, $S_7(23\%)$, $S_8(22\%)$, and $S_9(9\%)$. This indicates that the client should invest $12,000 on $S_1$, $26,000 on $S_3$, $8,000 on $S_5$, $23,000 on $S_7$, $22,000 on $S_8$, and $9,000 on $S_9$.

The above results show that the Markowitz portfolio has four securities, the fuzzy portfolio has five, while the possibility portfolio has six. It means that the possibility portfolio tends to take a more distributive investment than the fuzzy probability and Markowitz ones. Based on this one still cannot simply say that the possibility model is the best. It is more reasonable to combine these portfolios together in some way in real investment problems.

Suppose the degree to which the aggregation prefers high confidence decisions is 70\%, i.e., $\alpha = 0.7$. Using OWA to aggregate these portfolios with $\alpha = 0.7$, the final portfolio $P_{OWA}$ is obtained, which consists of $S_1(7\%)$, $S_3(21\%)$, $S_4(4\%)$, $S_5(13\%)$, $S_7(38\%)$, $S_8(12\%)$, and $S_9(5\%)$. All the results are summarized in Table 8.3.

### 8.4.4 Empirical Evaluation of the Aggregated Results

At this stage, one important problem is how to verify the aggregated portfolio. There is no systematic way available to answer this question. Instead, some experiments were conducted.

The first experiment conducted was to use the first 12 years (1982 to 1993) return data in Table 8.2 and produce three portfolios based on the three models. Based on the analysis in [133, 134], it is known that the fuzzy model is a direct extension of Markowitz's model, while the possibility model is more reasonable than the fuzzy model. Thus the three portfolios are ordered as $P_{POS}$, $P_{FUZ}$, and
Table 8.4: Portfolios and Variances Based on 12 Years Return Data

<table>
<thead>
<tr>
<th></th>
<th>S₁</th>
<th>S₂</th>
<th>S₃</th>
<th>S₄</th>
<th>S₅</th>
<th>S₆</th>
<th>S₇</th>
<th>S₈</th>
<th>S₉</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>5.15</td>
<td>13.95</td>
<td>19.53</td>
<td>39.75</td>
<td>21.62</td>
<td>0.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pₚ₀ₛ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pₚᵥₜᶻ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pₚₘᵣₐ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pₒₙʷₜₐ</td>
<td>2.85</td>
<td>7.73</td>
<td>21.12</td>
<td>20.77</td>
<td>22.02</td>
<td>25.51</td>
<td>0.18</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.5: Realized Average Returns of the Portfolios (%)

<table>
<thead>
<tr>
<th>Year(s)</th>
<th>Pₚₒₛ</th>
<th>Pₚᵥₜᶻ</th>
<th>Pₚₘᵣₐ</th>
<th>Pₒₙʷₜₐ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.421</td>
<td>7.501</td>
<td>7.554</td>
<td>9.681</td>
</tr>
<tr>
<td>2</td>
<td>27.694</td>
<td>15.665</td>
<td>15.700</td>
<td>22.334</td>
</tr>
<tr>
<td>4</td>
<td>19.684</td>
<td>14.990</td>
<td>15.006</td>
<td>17.593</td>
</tr>
</tbody>
</table>

Pₚₘᵣₐ and α = 0.7 (the degree that the aggregation prefers decisions with high confidence) is chosen when using OWA operator to aggregate the three portfolios. The selected portfolios as well as corresponding risks of investment are shown in Table 8.4. The portfolios in Table 8.4 are also selected with an expected average return rₛ = 17%.

The last 6 years (1994 to 1999) return data in Table 8.2 are used to verify the realized average returns of the four portfolios. The realized average returns of the four portfolios from one year to six years are listed in Table 8.5.

From Table 8.5, one can see that the average returns of Pₒₙʷₜₐ are better than those of Pₚᵥₜᶻ and Pₚₘᵣₐ, and slightly less than those of Pₚₒₛ. The variance (risk or uncertainty degree of the investment) of Pₒₙʷₜₐ is greatly reduced (from 0.30 to 0.18) compared with that of Pₚₒₛ.

To further verify the aggregated portfolio, 12 securities listed in Australian Stock Exchange Limited (ASX) were selected and 12 years average returns (from 1986 to 1997) were collected (see Table 8.6)\(^2\). The data are from [151, 152].

\(^2\)The ASX security codes of Sₚₒₛ to Sₚₒₛ are AKC, AFI, AGL, BPC, CSR, EML, GUD, SMI,
Table 8.6: Returns on Twelve Securities from ASX

<table>
<thead>
<tr>
<th>Yr</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>S9</th>
<th>S10</th>
<th>S11</th>
<th>S12</th>
</tr>
</thead>
<tbody>
<tr>
<td>86</td>
<td>.675</td>
<td>.33</td>
<td>.053</td>
<td>.946</td>
<td>.081</td>
<td>1.026</td>
<td>.42</td>
<td>.198</td>
<td>.405</td>
<td>.249</td>
<td>.75</td>
<td>.08</td>
</tr>
<tr>
<td>87</td>
<td>.648</td>
<td>.93</td>
<td>.864</td>
<td>1.097</td>
<td>.826</td>
<td>.897</td>
<td>.535</td>
<td>.207</td>
<td>1.023</td>
<td>.333</td>
<td>.994</td>
<td>.684</td>
</tr>
<tr>
<td>88</td>
<td>.248</td>
<td>.432</td>
<td>-.157</td>
<td>.656</td>
<td>.515</td>
<td>1.044</td>
<td>.623</td>
<td>.05</td>
<td>.132</td>
<td>.174</td>
<td>.238</td>
<td>.403</td>
</tr>
<tr>
<td>89</td>
<td>-.007</td>
<td>.405</td>
<td>.538</td>
<td>.698</td>
<td>.136</td>
<td>.544</td>
<td>.358</td>
<td>1.396</td>
<td>.468</td>
<td>.271</td>
<td>.355</td>
<td>.467</td>
</tr>
<tr>
<td>90</td>
<td>-.097</td>
<td>-.08</td>
<td>-.168</td>
<td>-.029</td>
<td>-.228</td>
<td>-.115</td>
<td>.243</td>
<td>-.233</td>
<td>-.203</td>
<td>-.251</td>
<td>-.208</td>
<td>.096</td>
</tr>
<tr>
<td>91</td>
<td>.314</td>
<td>.291</td>
<td>.787</td>
<td>.341</td>
<td>.37</td>
<td>.598</td>
<td>.577</td>
<td>.57</td>
<td>.037</td>
<td>.391</td>
<td>.382</td>
<td>.263</td>
</tr>
<tr>
<td>92</td>
<td>.246</td>
<td>.004</td>
<td>.025</td>
<td>-.161</td>
<td>.01</td>
<td>-.148</td>
<td>-.116</td>
<td>-.03</td>
<td>.018</td>
<td>-.127</td>
<td>-.177</td>
<td>-.138</td>
</tr>
<tr>
<td>93</td>
<td>.105</td>
<td>.126</td>
<td>.107</td>
<td>-.091</td>
<td>-.105</td>
<td>-.126</td>
<td>.133</td>
<td>-.013</td>
<td>.011</td>
<td>.005</td>
<td>-.164</td>
<td>-.03</td>
</tr>
<tr>
<td>94</td>
<td>.133</td>
<td>.135</td>
<td>.264</td>
<td>-.17</td>
<td>.058</td>
<td>.112</td>
<td>0</td>
<td>.292</td>
<td>.232</td>
<td>.129</td>
<td>.014</td>
<td>-.006</td>
</tr>
<tr>
<td>95</td>
<td>.385</td>
<td>.433</td>
<td>.543</td>
<td>.042</td>
<td>.167</td>
<td>.293</td>
<td>.675</td>
<td>.646</td>
<td>.55</td>
<td>.62</td>
<td>.42</td>
<td>.361</td>
</tr>
<tr>
<td>96</td>
<td>.053</td>
<td>.105</td>
<td>.319</td>
<td>-.947</td>
<td>.121</td>
<td>-.179</td>
<td>-.487</td>
<td>-.214</td>
<td>.226</td>
<td>-.046</td>
<td>-.299</td>
<td>-.265</td>
</tr>
<tr>
<td>97</td>
<td>-.2</td>
<td>.109</td>
<td>-.046</td>
<td>1.192</td>
<td>.305</td>
<td>-.168</td>
<td>-.06</td>
<td>.258</td>
<td>-.213</td>
<td>-.088</td>
<td>-.111</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 8.7: Portfolios and Variances Based on ASX 8 Years Return Data

<table>
<thead>
<tr>
<th>P</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>S9</th>
<th>S10</th>
<th>S11</th>
<th>S12</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPOS</td>
<td>22.2</td>
<td>26.45</td>
<td>12.06</td>
<td>9.95</td>
<td>0.07</td>
<td>25.65</td>
<td>6.62</td>
<td>0.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFUZ</td>
<td>59.11</td>
<td>12.10</td>
<td>6.19</td>
<td>22.60</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMAR</td>
<td>43.10</td>
<td>35.77</td>
<td>12.58</td>
<td>10.55</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POWA</td>
<td>36.20</td>
<td>14.66</td>
<td>6.68</td>
<td>14.25</td>
<td>3.78</td>
<td>15.87</td>
<td>8.56</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Similar to experiment one, the first 8 years (1986 to 1993) return data were used to generate the portfolios, while the last 4 years (1994 to 1997) data were used to verify. When the expected average return $r_s = 17\%$, the selected portfolios based on the three models and the aggregated portfolio based on OWA with $\alpha = 0.7$ are listed in Table 8.7.

Based on the four portfolios, the realized average returns from one year to four years are shown in Table 8.8.

The results in Table 8.8 are consistency with those in Table 8.5. Thus the same conclusion can be reached. The average returns of $P_{OWA}$ are better than those of $P_{FUZ}$ and $P_{MAR}$, and slightly less than those of $P_{POS}$. The variance (risk) of $P_{OWA}$ is greatly reduced (from 0.26 to 0.15) compared with that of $P_{POS}$.

Finally, different expected average return values (from 10% to 20%) were used

HAH, OPS, PDP, and WYL, respectively.
Table 8.8: Realized Average Returns of the Portfolios Based on ASX Data (%)  

<table>
<thead>
<tr>
<th>Year(s)</th>
<th>( P_{POS} )</th>
<th>( P_{FUZ} )</th>
<th>( P_{MAR} )</th>
<th>( P_{OWA} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.432</td>
<td>9.556</td>
<td>9.351</td>
<td>11.672</td>
</tr>
<tr>
<td>2</td>
<td>30.833</td>
<td>25.035</td>
<td>28.315</td>
<td>28.752</td>
</tr>
<tr>
<td>3</td>
<td>19.584</td>
<td>11.304</td>
<td>8.524</td>
<td>15.463</td>
</tr>
<tr>
<td>4</td>
<td>12.644</td>
<td>4.887</td>
<td>3.919</td>
<td>9.035</td>
</tr>
</tbody>
</table>

To test the four portfolios based on the two sets of return data, the same conclusion was reached.

8.5 Summary

This chapter dealt with the decision aggregation problem in multi-agent systems, which are related to the DECISIONAGGREGATOR role and Aggregation Agent (see Chapter 4).

There are two issues involved in decision aggregation in MASs. One is how to choose appropriate aggregation algorithms for specific application. The other is how to implement these algorithms effectively and efficiently in MASs.

The first issue is application dependent. How to choose suitable aggregation operations was demonstrated through financial investment applications.

For the second issue, we designed and implemented three approaches – the stationary agent approach, the token-passing approach, and the mobile agent approach. The stationary agent approach is the most common one. It is directly derived from the architecture (see Figure 4.11). After the three approaches were compared, it is concluded that the first two are suitable for small-scale MASs, and the mobile agent approach is the most promising and practical approach in large-scale MASs as well as in open environments such as the Internet.

A case study of financial portfolio selection was presented. In this case study, both categories of intelligent techniques (hard computing and soft computing techniques) are integrated in the system.

The empirical evaluation of the aggregated results indicates that the OWA operator is suitable for the financial investment application. To evaluate the aggregated
results effectively and systematically requires further research.
Chapter 9

Concluding Remarks

In this thesis, an agent-oriented methodology for the analysis and design of agent-based hybrid intelligent systems has been tailored. An agent-based hybrid framework for complex decision making based on the methodology has been proposed. Also discussed are the related issues such as the construction of domain-dependent ontology, matchmaking in middle agents, reasoning in decision making as well as decision aggregation. We have demonstrated the framework through some examples in financial investment application. The final step is to summarize the arguments presented in the thesis and reflect on them.

In Section 9.1, all the case studies from the previous chapters are pulled together to construct a prototype for an agent-based hybrid intelligent system for decision making. A comparison of the proposed framework with related work is given in Section 9.2. The framework is then evaluated with particular reference to the prototype in Section 9.3. The conclusions obtained are summarized (Section 9.4), and possible future work is outlined (Section 9.5).

9.1 The Prototype System: Pulling It All Together

We have implemented a prototype system of agent-based intelligent technique society using Java and under the support of JATLite. The application background of the prototype is financial investment planning. In fact, the case studies in this
thesis are derived from the prototype.

Following the analysis and design phases of the proposed methodology, we worked out that the prototype consists of the following agents: one middle (serving) agent, one interface agent, one planning agent, one aggregation agent, four decision making agents (investment policy agent, portfolio selection agent based on the Markowitz model, portfolio selection agent based on the fuzzy model, and portfolio selection agent based on the possibility model), and five service provider agents (financial risk tolerance ability evaluation agent, interest rate prediction agent based on neural network, interest rate prediction agent based on fuzzy logic and genetic algorithm, approximate reasoning agent, and a quadratic programming problem solving agent).

To demonstrate how the prototype works, we first examine the typical scenario for investment. We then present an example to show how the multi-agent financial investment planning prototype system gives advice to the investor.

9.1.1 A Typical Scenario for Investment

When a person wants to invest some money somewhere, he usually comes to the financial investment adviser for advice. The first thing the adviser needs to do is to understand the client's individual circumstances. The adviser may ask the client to provide the following information about himself: his financial position (annual income, total net-worth etc.), age, tax effectiveness etc. Based on the information, the adviser should evaluate the financial risk tolerance ability as well as the client's investment goal.

If the client's primary goal is income, then investments that provide interest or dividend payments regularly and dependably are required. If the primary goal is growth, then investments that are likely to increase in value are wanted so that they may be resold for more than their initial cost. If, however, the primary goal is to avoid risk, then investments that offer the greatest safety of principal and protection from inflation are required. Unfortunately, there is no single investment that simultaneously offers maximum income, maximum growth and minimum risk.

Suppose the adviser suggests the client invest some money in the stock market after evaluating his financial risk tolerance ability. How can one select a portfolio
CHAPTER 9. CONCLUDING REMARKS

for the client under his constraints (risk tolerance level, return rate etc.)? The adviser should gather some information about the stock market. The information includes market data, financial report data, technical models, analysts' reports, breaking news etc. After gathering the information, the adviser then makes a portfolio selection decision based on some models (e.g., the Markowitz model, the fuzzy probability model etc.)

In short, the overall investment advising task has several component tasks: eliciting (or learning) user profile information, collecting information on the user's initial portfolio position, and suggesting and monitoring a reallocation to meet the user's current profile and investment goals.

9.1.2 Example

Suppose the investor provides the interface agent with the following personal information (see Figure 9.1):

- **Amount of money to invest:** $100,000
- **Annual income:** $72,000
- **Total net-worth:** $480,000
- **Age:** 34

**Investment goal:** growth (can be *income, growth*, and *avoid risk* etc. This reflects investor's attitude toward risk.)

![Figure 9.1: Example Input Screen of the Prototype](image)

The interface agent passes the investor's information to the planning agent (see Figure 4.11). The planning agent will delegate sub-tasks to different decision making agents. During the decision making process, decision making agents may
ask different service provider agents (most of them are intelligent technique agents) for help. Finally, the prototype will give its advice to the investor through the interface agent (see Figure 9.2). The whole process (from receiving inputs to giving advice) consists of the following five steps.

![My Advice]

**Figure 9.2: Example Output Screen of the Prototype**

**Step 1:** Determining the investor's investment policy (aggressive or conservative).

To determine the investor's investment policy, the investment policy agent will use fuzzy rules in its knowledge base. One example of a fuzzy rule is as follows:

**If** the investor's risk tolerance is *high* and the interest rate is *falling* and the investor's investment goal is *growth* then the investor can take an *aggressive investment policy*.

The investment policy agent will then ask for help based on its meta-knowledge. Thus, the investment policy agent sends KQML messages using *recommend-one* performative to the serving agent:

(\texttt{recommend-one}

\texttt{sender investment_policy_agent}

\texttt{receiver serving_agent}

\texttt{language KQML}

\texttt{content (ask}

\texttt{ :ability risk_tolerance_evaluation}

\texttt{ :name ?}

\texttt{ :ontology ?}
The serving agent then retrieves its database and replies with an appropriate service provider agent’s name and ontology which has the capability asked for using reply performative. In the prototype, there are a risk tolerance ability evaluation agent based on fuzzy logic and two interest rate prediction agents based on feedforward neural network and genetic algorithm, respectively. The serving agent will choose the interest rate prediction agent based on fuzzy genetic model according to the initial values of these agents’ track records (see Section 6.4.2).

(reply
  :sender serving_agent
  :receiver investment_policy_agent
  :content (:name SC_Agent_FL
    :ontology Financial_investment
  )
)

(reply
  :sender serving_agent
  :receiver investment_policy_agent
  :content (:name SC_Agent_FLGA
  )
)
The investment policy agent then communicates with SC_Agent_FL (for risk tolerance evaluation) and SC_Agent_FLGA (for interest rate prediction) directly.

The risk tolerance evaluation agent (based on fuzzy logic) uses rules such as

If the investor's annual income is high and total net-worth is high and the investor is young then the investor's risk tolerance ability is high;

If annual income > $50,000 then annual income is high;

If age ≤ 35 then the investor is young etc.

Based on the investor's information and the financial risk tolerance model (see Section 4.6.1), the risk tolerance evaluation agent obtains the result that the investor's risk tolerance ability is high. During this process, the risk tolerance evaluation agent needs the help of the approximate reasoning agent.

The fuzzy-genetic based interest rate prediction agent will ask the following parameters as input (see Section 6.4.2): the change of real gross national product ΔGNP, the change of consumer price index ΔCPI, the change of M2 nominal money supply ΔM2, the change of personal wealth ΔW, and the change of previous T-bill rates Δr. All these data need to be gathered by information gathering agents.

Suppose the gathered data are as follows: ΔGNP = 0.50, ΔCPI = −0.10, ΔM2 = 47.80, ΔW = 456.70, and Δr = 0.01. The predicted result is −0.76. Thus the interest rate prediction agent reaches the conclusion that the interest rate will fall.

Combining the results of the two SC agents and the investor's investment goal, the investment policy agent reaches the conclusion that the investor's investment policy can be aggressive.

Step 2: Determining the investment category. The investment category can be the stock market, real estate, fixed term deposit etc. One of the decision making agents based on a fuzzy model is used to accomplish this task. Example rules used are:

If the investor's risk tolerance ability is high and the investment policy is aggressive then the suggested investment category is the stock market;
CHAPTER 9. CONCLUDING REMARKS

If the investor's risk tolerance ability is high and the investment policy is conservative then the suggested investment category is real estate, etc.

Based on the model, the agent gives the suggestion that the investor can invest in the stock market.

Step 3: Delegating information gathering agents to gather stock market information (this step has not been implemented in the prototype). Here we suppose information gathering agents gathered the returns of nine securities in the past 18 years (refer to Table 8.2).

Step 4: Selecting portfolios from these nine securities that satisfies the investor's risk tolerance level and maximizes the return. The selected portfolios (with the expected average return $\tau = 0.17$) based on the three portfolio selection models – the Markowitz model, the fuzzy probability model, and the possibility distribution model are shown in Table 8.3.

Step 5: Aggregating the portfolios based on different models and giving a final portfolio to the investor.

How can one aggregate the portfolios from different models and produce a final one that is the best among those? This is a very tough question to answer. Currently, we aggregated these portfolios based on the OWA aggregation algorithm based on fuzzy logic. Based on the OWA aggregation ($\alpha = 0.7$), the final portfolio consists of $S_1(7\%), S_2(21\%), S_3(4\%), S_4(13\%), S_5(38\%), S_6(12\%),$ and $S_7(5\%)$ (see Table 8.3).

Finally, the prototype provides the investor with the following advice (through the interface agent): invest in the stock market with portfolio $P_{OWA}$ (see Table 8.3 and Figure 9.2).

9.2 A Comparison of the Framework with Related Work

Hybrid intelligent systems are of paramount importance for complex decision making, while agent technology is well suited for modeling hybrid intelligent systems. That is why we proposed the agent-based hybrid framework for complex decision making. Of course, this is not the only framework for this purpose. Thus far, there
is also some other research work involved in this topic (refer to Section 2.3).

One of such attempts is the MIX multi-agent platform [35, 87]. Another such attempt is the PREDICTOR system [36]. In [37], Khosla and Dillon introduce a computational architecture called IMAHDA (Intelligent Multi-Agent Hybrid Distributed Architecture). A more recent attempt is the multi-agent architecture for fuzzy modeling [38]. Delgado et al. proposed a hybrid learning system that combines different fuzzy modeling techniques by means of a multi-agent architecture. In [148], Jacobsen proposed a generic architecture for hybrid intelligent systems, which is based on the conceptual learning agent architecture [3].

Among the above agent-based hybrid frameworks or systems, the MIX, PREDICTOR, and the architecture for fuzzy modeling only integrated very limited soft computing techniques. Both the MIX and PREDICTOR systems are focused on the integration of neural networks and symbolic technologies such as expert systems. The multi-agent architecture of Delgado et al. concentrated on the integration of different fuzzy modeling techniques such as fuzzy clustering, fuzzy rule generation, and fuzzy rule tuning techniques. In MIX and PREDICTOR systems, the way for integrating intelligent techniques into multi-agent systems is to embed the intelligent techniques in each individual agent. The MIX and IMAHDA architectures are inflexible as no middle agent [39] was used. The work in [148] is focused on the micro (intra-agent) level of agents, i.e., the integration and interaction of different components within one agent. The macro (inter-agent) level integration and interaction are ignored.

Compared with the related work, the proposed framework in this thesis has the following crucial characteristics that differentiate this research from others:

- The ability to exchange comprehensible communications (interactions at knowledge level) among or within agents, i.e., both micro level and macro level interactions. Interactions can occur among decision making agents and service provider agents;

- Each decision making agent can easily access all the intelligent techniques and other techniques (service providers) available in the system whenever needed. One service provider agent can also ask other service provider agents for help;
CHAPTER 9. CONCLUDING REMARKS

- The ability to add/delete service provider agents to/from the system dynamically;

- The presence of the serving agent in this framework allows adaptive system organization;

- Overall system robustness is facilitated through the use of the serving agent. For example, if a particular service provider (e.g., SC agent) disappears, a requester agent (decision making agent) can find another one with the same or similar capabilities by interrogating the serving agent.

- The agent-based hybrid intelligent systems can make decisions about the nature and scope of interactions at run-time.

9.3 Evaluation

Complex problems such as financial investment planning are uncertain, dynamic, distributed, and heterogeneous in nature. Decision making on such problems needs the combination of adequate information, rich knowledge, and great skills to use knowledge and information. The framework proposed here is suitable for such tasks and can overcome the shortcomings of traditional hybrid intelligent system frameworks (see Section 3.2). Here, the evaluation of the framework and the work we have done is given by considering the following:

- **Flexibility** In the framework, we introduced a special kind of middle agents — serving agent. The presence of the serving agent in the framework allows adaptive system organization. The intelligent technique society proposed has the ability to add and delete intelligent technique agents (or other service provider agents) dynamically as needed.

- **Robustness** In the framework, if a particular service provider disappears, a requester agent can find another one with the same or similar capabilities by interrogating the serving agent. Thus overall system robustness is facilitated through the use of the serving agent.
• **Reliability** Compared with most currently available MAS frameworks, the matchmaking results of the serving agent (matchmaker) in this framework is more reliable due to the consideration of the agents’ track records. The serving agent has less chance of a single point of failure compared with the OAA Facilitator [18, 19] and InfoSleuth’s Broker [20], since after a requester has been given a list of providers, it can continue its transactions directly even when no serving agent is present. In addition, a requester can cache providers’ contact information and reuse them without resorting to a serving agent every time.

• **Availability of agent capabilities** As any agent under this framework should register and advertise its capabilities to the serving agent, any other agent can easily access all the capabilities available in the system whenever needed.

• **Interoperability** One main obstacle of a meaningful interoperation and mediation of services is the syntactic and semantic heterogeneity of data and knowledge the middle agent does access and receive from multiple heterogeneous agents, information systems and sources. By using domain-specific ontologies, the semantic heterogeneities are resolved among agents in the framework.

• **Inter-agent interactions** The interactions among agents are easy as all agents use the standardized agent communication language – KQML.

• **Platform independent** The prototype was implemented by using Java and under the support of JATLite. It can execute on any platform with a Java Virtual Machine. Thus it is platform independent.

The methodology we introduced is based on the Gaia methodology but different from it. One difference is that our methodology is suitable for the analysis and design of applications in open environments such as the Internet while Gaia is not. Another difference is the skill model and knowledge model in this methodology, which are of paramount importance in modeling complex problem solving and decision making. All the agents in the prototype, which were derived from the methodology, exhibited their behaviors correctly as specified. However, we are
not clear whether and how the adequacy and completeness of the outputs of the methodology can be assessed at this stage.

9.4 Conclusions

Many complex problems have many different components, each of which requires different types of processing. That is, they require hybrid solutions. Increasingly, hybrid intelligent systems combining different intelligent techniques such as expert systems, FL, NN, and GA are proving their effectiveness in a wide variety of real-world problem solving and decision making. Hybrid intelligent systems usually have a large number of parts or components that have many interactions. It is now widely recognized that interaction is probably the most important single characteristic of complex software. Thus hybrid intelligent systems are complex in nature.

On the other hand, agents in multi-agent systems are autonomous and can engage in flexible, high-level interactions from a multi-agent perspective. Multi-agent systems are good at complex, dynamic interactions. Thus the multi-agent perspective is suitable for the modeling, design, and construction of hybrid intelligent systems.

Existing software development techniques (typically, object-oriented) are inadequate for modeling agent-based hybrid intelligent systems. There is a fundamental mismatch between the concepts used by object-oriented developers and the agent-oriented view. Although there are some agent-oriented methodologies such as the Gaia methodology, there is still no specifically tailored methodology available for the analysis and design of agent-based hybrid intelligent systems. For these reasons, we introduced a methodology, which has been specifically tailored to the analysis and design of agent-based hybrid intelligent systems.

The proposed methodology consists of six models - the role model, the interaction model, the agent model, the skill model, the knowledge model, and the organizational model. This methodology differs from other agent-oriented methodologies in its skill and knowledge models. We added these two models to the methodology as they are of paramount importance in modeling complex decision making and
problem solving.

Follow the methodology, an agent-based hybrid framework for complex decision making – the agent-based intelligent technique society – was developed. The framework has several crucial characteristics that differentiate this research from others (see Section 9.2).

Four important issues relating to the framework as well as applying it to financial investment planning have been investigated. These cover the building of an ontology for financial investment, matchmaking in middle agents, reasoning in decision making, and decision aggregation in multi-agent systems.

We demonstrated how to build a domain-specific ontology and how to access it in a multi-agent system by building a financial ontology. Experience with the financial ontology development and a relatively profound analysis of current ontology research have led us to an extended and refined set of ideas regarding the next generation of ontology construction tools. We proposed that the next generation of ontology construction tools should have adapting and reconciling, visualizing ontologies, and detecting inconsistencies etc. nine capabilities. A framework with these nine capabilities is reasonable and actual.

Matchmaking is very important in middle agents. To our knowledge, almost all currently used matchmaking algorithms are only based on the advertised capabilities of providers when doing matchmaking. Those algorithms did not consider the practical outcomes of service provider agents in accomplishing delegated tasks at all. To this end, it was suggested to consider agents' track records in matchmaking. The simulation results has shown that agents' track records have a strong impact on the outcome of matchmaking. With more history information of agents in accomplishing delegated tasks, the proposed algorithm can improve the matchmaking. We also suggested a way to provide initial values for agents' track records. When these two are combined, the matchmaking results of the algorithm are more accurate and reasonable.

Reasoning is one important part of the skill model. Approximate reasoning that can deal with fuzzy information must be employed because much fuzzy information is available in financial applications. As approximate reasoning is very time-consuming, we explored the parallelism in it. The parallel implementation
greatly accelerated approximate reasoning and is suitable for incorporating into agents. The concept of “reasoning with multimedia information” was introduced. To our knowledge, this problem has not previously been addressed. We proposed to use symbolic projection theory for reasoning with still image information. A case study of reasoning with moving average charts in finance was provided. When using symbolic projection to solve the problem, a new concept – introducing benchmark objects – was developed.

There are two issues involved in decision aggregation in MASs. One is how to choose appropriate aggregation algorithms for specific application. The other is how to implement these algorithms effectively and efficiently in MASs. This thesis demonstrated how to choose suitable aggregation operations through financial investment application. We also proposed three approaches – the stationary agent approach, the token-passing approach, and the mobile agent approach to implementing decision aggregation in MASs. After comparing the three approaches, we concluded that the first two are suitable for small-scale MASs, and the mobile agent approach is the most promising and practical approach in large-scale MASs as well as open environments such as the Internet.

Based on the intelligent technique society, we built an agent-based hybrid intelligent system prototype for financial investment planning. In the prototype, there are one serving agent, one interface agent, one decision aggregation agent, one planning agent, four decision making agents, and five service provider agents.

Some experiments were conducted on the prototype. The experimental results show the framework is flexible, robust, and fully workable. All agents derived from the methodology exhibited their behaviors correctly as specified.

9.5 Future Work

As discussed in Chapter 2, at this time, there are two major technical impediments to the widespread adoption of agent technology [7]:

- the lack of a systematic methodology enabling designers to clearly specify and structure their applications as multi-agent systems; and

- the lack of widely available industrial-strength multi-agent system toolkits.
In response to the first impediment, we have tailored a methodology suitable for the analysis and design of agent-based hybrid intelligent systems from the available methodologies. Further work is needed to detail the proposed methodology, by:

- working out a formal semantics for the methodology. We believe that a successful methodology is one that is not only of pragmatic value, but one that also has well-defined, unambiguous formal semantics. It is essential to have a precise understanding of what the concepts and terms in a methodology mean [6].

- providing suitable notations for expressing the expected outputs of the analysis and design phases. We expect standard notations, such as UML (Unified Modeling Language) used in object-oriented software engineering [147], to be rapidly adapted to the needs of agent-based software engineering, as well as new agent-specific methodologies to emerge.

In response to the second impediment, we have proposed a general framework to facilitate the construction of agent-based hybrid intelligent systems and developed a prototype. To facilitate the development of agent-based hybrid intelligent systems quickly, an industrial-strength toolkit is needed, which will be based on the proposed framework. When developing the toolkit, an abstract model of intelligent technique agents is required so as to build the intelligent technique agent library.

The third issue subject to further research is reasoning with multimedia information. Reasoning with multimedia information includes reasoning with still image information, reasoning with video information, and reasoning with audio information, etc. This thesis has only dealt with reasoning with still image information. Further research is needed in this area.

In addition to these three main aspects, plans for future work also include elaborating the algorithm considering agents' track records in matchmaking as well as how to evaluate the aggregated results in decision making effectively and systematically.
Bibliography


