AGENT-BASED OPEN CONNECTIVITY FOR DECISION SUPPORT SYSTEMS

By

HAO LAN ZHANG

MInfoSys, Central Queensland University

MBA, Central Queensland University

A thesis submitted to in fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science and Mathematics

VICTORIA UNIVERSITY

2007

DECLARATION

I, Hao Lan Zhang, declare that the PhD thesis entitled *Agent-based Open Connectivity for Decision Support Systems* is no more than 100,000 words in length including quotes and exclusive of tables, figures, appendices, bibliography, references and footnotes. This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work.

_____ _____

Signature Date

ACKNOWLEDGEMENTS

It would have not been possible for me to finish this thesis without the help from many people. First, I would like to thank my principal supervisor, Professor Clement Leung, for his guidance, support, and valuable advice during the past three years. Our regular meetings guided me towards the right direction. This research work would have not been done without his insightful suggestions.

I am grateful to my current principal supervisor, Professor Yanchun Zhang, for his valuable comments, guidance, advice, support, and encouragement in the crucial stage of finalising my thesis. I would have not been able to finalise this thesis without his help. I also appreciate his advice and help in starting my academic career.

I would like to thank my co-supervisor, Dr. Gitesh Raikundalia, for his valuable comments, guidance, support, and assistance throughout my research study. I appreciate him for giving me the opportunity to work as the research assistant, which helped me gain the research experience.

I wish to thank Angela Rojter for correcting my English writing in my thesis. I highly appreciate her excellent work and friendly help that created a pleasant working environment for me.

Many thanks to the School of Computer Science and Mathematics at Victoria University for providing a high quality research environment, which include: providing me with several travel funds to attend conferences, and offering me the computing facilities for my research experiments. I am grateful to the Head of the School, A/Prof. Peter Cerone, for offering me the school scholarship although I took another scholarship in the end. Thanks also go to Victoria University for offering me the Victoria University Scholarship.

To all the staff members and students in the School of Computer Science and Mathematics at Victoria University, particularly, Dr. Bai-ling Zhang, Janet Grady, Ponnusamy Rajendran, Danh Ho, Mark Mojic, A/Prof. Yuan Miao, Ian Gomm, Michael Grubinger, A/Prof. Neil Barnett, Dr. Iwona Miliszewska, and Anne Venables.

I wish to express my deep gratitude to Dr. Jing He from the Chinese Academy of Sciences for her encouragement, help, and suggestions.

Finally but most importantly, I would like to thank my parents, my sister, Yinghong, and her family. I can do nothing without their support.

ABSTRACT

One of the major problems that discourages the development of Decision Support Systems (DSSs) is the un-standardised DSS environment. Computers that support modern business processes are no longer stand-alone systems, but have become tightly connected both with each other and their users. Therefore, having a standardised environment that allows different DSS applications to communicate and cooperate is crucial. The integration difficulty is the most crucial problem that affects the development of DSSs. Therefore, an open and standardised environment for integrating various DSSs is required.

Despite the critical need for an open architecture in the DSS designs, the present DSS architectural designs are unable to provide a fundamental solution to enhance the flexibility, connectivity, compatibility, and intelligence of a DSS.

The emergence of intelligent agent technology fulfils the requirements of developing innovative and efficient DSS applications as intelligent agents offer various advantages, such as mobility, flexibility, intelligence, etc., to tackle the major problems in existing DSSs. Although various agent-based DSS applications have been suggested, most of these applications are unable to balance manageability with flexibility. Moreover, most existing agent-based DSSs are based on agent-coordinated design mechanisms, and often overlook the living environment for agents. This could cause the difficulties in cooperating and upgrading agents because the agent-based coordination mechanisms have limited capabilities to provide agents with relatively comprehensive information about global system objectives.

This thesis proposes a novel multi-agent-based architecture for DSS, called Agent-based Open Connectivity for Decision support systems (*AOCD*). The AOCD architecture adopts a hybrid agent network topology that makes use of a unique feature called the *Matrix-agent* connection. The novel component, i.e. Matrix, provides a living environment for agents; it allows agents to upgrade themselves through interacting with the Matrix. This architecture is able to overcome the difficulties in concurrency control and synchronous communication that plague many decentralised systems. Performance analysis has been carried out on this framework

and we find that it is able to provide a high degree of flexibility and efficiency compared with other frameworks.

The thesis explores the detailed design of the AOCD framework and the major components employed in this framework including the Matrix, agents, and the unified Matrices structure. The proposed framework is able to enhance the system reusability and maximize the system performance. By using a set of interoperable autonomous agents, more creative decision-making can be accomplished in comparison with a hard-coded programmed approach.

In this research, we systematically classified the agent network topologies, and developed an experimental program to evaluate the system performance based on three different agent network topologies. The experimental results present the evidence that the hybrid topology is efficient in the AOCD framework design. Furthermore, a novel topological description language for agent networks (TDLA) has been introduced in this research work, which provides an efficient mechanism for agents to perceive the information about their interconnected network.

A new *Agent-Rank* algorithm is introduced in the thesis in order to provide an efficient matching mechanism for agent cooperation. The computational results based on our recently developed program for agent matchmaking demonstrate the efficiency and effectiveness of the Agent-Rank algorithm in the agent-matching and re-matching processes.

PUBLICATIONS

Refereed Journal Articles:

1. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G.K. (2008) "Topological Analysis of Agent Networks and Experimental Results based on the AOCD Architecture", *Journal of Computer and System Sciences*, Vol. 74, No. 2, pp. 255 – 278, Elsevier Publication.
2. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G.K. (2006) "Matrix-Agent Framework: A Virtual Platform for Multi-agents", *Journal of System Sciences and Systems Engineering*, Vol. 15, No. 4, pp. 436 - 456, Springer-Verlag Press.
3. Raikundalia, G.K. and **Zhang, H.L.** (2006) "Document-related Awareness Elements in Synchronous Collaborative Authoring", *Australian Journal of Intelligent Information Processing Systems*, Vol. 9, No. 2, pp. 41 – 48.

Submitted Journal Articles:

4. **Zhang, H.L.**, Leung, C.H.C., Zhang, Y., and Raikundalia, G.K. "A New Agent-Rank Algorithm for Agent Matching", **Submitted to** *IEEE Transactions on Systems, Man, and Cybernetics-Part A.*
5. **Zhang, H.L.**, Leung, C.H.C., and Raikundalia, G.K. "Application of Multi-agent Technology to Information System: An Agent-based Design Architecture for Decision Support Systems", **Submitted to** *AJIS* (in December 2006).

Refereed Conference and Workshop Papers:

6. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G.K. (2007) "Performance Evaluation of Agent Network Topologies Based on the AOCD Architecture", *Proc. of 21st International Conference on Advanced Information Networking and Applications Workshops* (*AINAW*), Canada, pp. 605 – 610, IEEE CS Press.
7. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G.K. (2006) "Classification of Intelligent Agent Network Topologies and A New Topological Description Language for Agent Networks", in: *IFIP International Federation for Information Processing, Volume 228, Intelligent Information Processing III, Shi Z., Shimoharar K., Feng D.* (*eds.*), Australia, pp. 21–31, Springer-Verlag Press.
8. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G.K. (2006) "Towards the *Matrix* Concept: A Virtual Platform for Intelligent Agent Application in Decision Support Systems". *Proc. of The 7th International Symposium on Knowledge and Systems Science*, Beijing, China, pp. 115 – 122, Lecture Notes in Decision Sciences 8.
9. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G. K. (2006) "Performance Analysis of Network Topology in Agent-based Open Connectivity Architecture for DSS", *Proc. of 20th International Conference on Advanced Information Networking and Applications (AINA)*, Austria, Vol.2, pp. 257-261, IEEE CS Press.
10. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G. K. (2005) "AOCD: A Multi-agent Based Open Architecture for Decision Support Systems", *Proc. of International Conference on Computational Intelligence for Modelling, Control & Automation*, Vienna, Austria, Vol. 2, pp. 295-300, IEEE CS Press.

Refereed Technical Reports:

11. **Zhang, H.L.**, Leung, C.H.C. and Raikundalia, G. K. (2006) "A Novel Matrix-Agent Connection Design in Multi-agent Based Architecture for Decision Support Systems", *Technical Reports of International Conference on Creativity and Innovation in Decision Making and Decision Support – An IFIP TC8/WG 8.3 Open Conference*, The London School of Economics, London, UK.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Making decisions in the current dynamic environment is complicated and indefinite because of various unpredictable factors and large volumes of information. A solely human-based decision-making process is exceedingly insufficient in terms of computational capability, analytical accuracy, and memory capacity. Therefore, it has become a key issue for modern organisations to have efficient systems and tools to support rational decision-making in problem-solving processes, particularly, for problems with a lack of predefined structure.

A Decision Support System (*DSS*) provides information and data manipulation tools to help make decisions in semi-structured and unstructured situations [3]; it is particularly efficient for dealing with complex problems. Existing DSS applications have been extensively applied to various areas, such as financial analysis, medical diagnosis, military, water resource planning, etc. Unfortunately, there is an obvious trend of a continuous decline of the DSS applications in practice [8, 27, 145], and it motivates a paradigm shift for DSSs [27, 145]. Various problems restrain the development of DSSs, which include: the lack of a standardised structure that allows various DSS applications to cooperate and communicate efficiently, and the lack of an active and intelligent methodology to support decision making in complex situation [27]. Researchers in the DSS area have realised the current situation and suggested several technologies, such as intelligent agents, to tackle existing problems.

Intelligent agents are an emerging technology in DSSs, which have been recognised as a flexible and efficient methodology to facilitate the development of DSSs [6, 145]. In this thesis, the term ‘agent’ refers to ‘intelligent agent’, which is described in [98] as: “autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realise a set of goals or tasks for which they are designed.”

Agents have various properties including intelligent, proactive, reactive, mobile, adaptive, cooperative, customising, flexible, etc. These features allow agents to adapt to the dynamic environment efficiently and solve specific tasks independently. Moreover, a complex problem can be solved through multi-agent cooperation. These

benefits assure the underlying success of applying the agent technology to complex and distributed systems, i.e. DSSs.

The advantages of applying agents in the DSS field have been discussed in numerous literature studies [6, 24, 40, 145, 160], which basically can be summarised as: improving system intelligence (reasoning capabilities), enhancing connectivity and compatibility, improving flexibility and expandability, transforming a DSS from a passive tool to an active collaborator in decision-making, improving the capability of solving complex problems through agent cooperation, and etc. Meanwhile, various agent-based DSS applications have been developed [97, 114]. However, most of the current agent-based DSS applications have several vital limitations, such as the lack of standardisation, the lack of reusability and manageability, and the lack of efficient self-upgrading mechanisms for agents (see Chapter 2, Section 2.3).

This thesis aims to provide a novel agent-based framework for DSSs based on the analysis of the major problems in existing DSSs (see Section 1.1). The proposed multi-agent-based framework, namely the *Agent-based Open Connectivity for DSS* (*AOCD*) framework, provides an open, efficient, and flexible architecture for DSSs.

The AOCD framework design is based on existing agent-based DSS applications and mainly focuses on building an open and efficient framework for DSS applications. The AOCD framework overcomes the compatibility and connectivity problems in most traditional DSS applications, and provides a 'living environment' for agents, an issue which most existing agent-based DSS applications have not addressed. This living environment is provided by the Matrix, which allows agents to acquire information, self-upgrade, perform tasks, travel to other Matrices, and be reused. The AOCD framework offers several key features, such as the hybrid network structure, the Matrix design, the unified Matrices framework, agent self-upgrading mechanisms, etc., which could facilitate the development of agent technology in DSS applications.

The next section provides an analysis of the major problems in existing DSSs and clarifies the motivations of this thesis.

1.1. Problem Description and Motivation

Various problems in DSS applications restrain the theoretical and practical developments in this area. In this section, we analyse the problems in traditional DSSs and existing agent-based DSSs to clarify the motivation of this thesis.

1.1.1 Problems in traditional DSSs

During the past decade, the successful applications of information systems in business processes have declined because of various obstacles. A survey provided by the Standish Group [142] shows that in U.S.A, 23% information systems failed outright and 49% systems were late and (or) went over-budget. Similar cases were also found, not only in U.S.A but also in U.K and other countries. Among the various obstacles, Alter [3] summarizes five major obstacles when developing an information system (especially a DSS) in the real world, which include:

(1) "Unrealistic expectations and techno-hype";

(2) "Difficulty building and modifying IT-Based Systems";

(3) "Difficulty integrating IT-based systems";

(4) "Organisational inertia and problems of change";

(5) "Genuine difficulty anticipating what will happen".

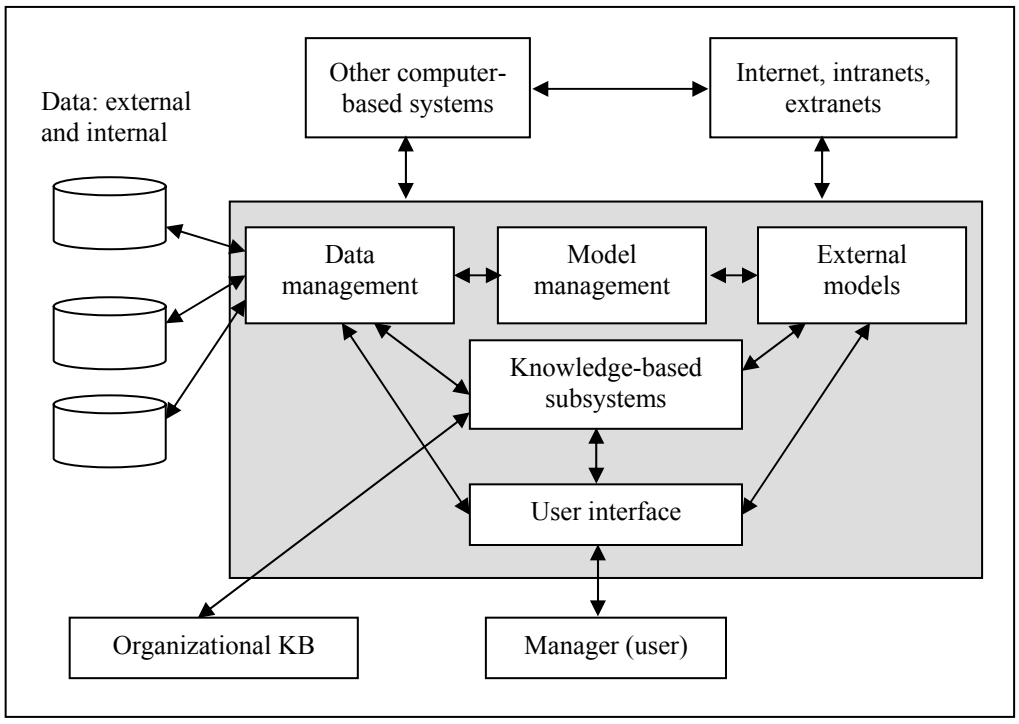


Figure 1.1. A Schematic View of Traditional DSSs [144].

We find that most of these obstacles are caused by the structural problems in current information systems, particularly in DSSs. Figure 1.1 shows a schematic view of traditional DSSs. This figure shows that the system structures of traditional DSSs are closed and that causes various problems, such as lack of compatibility, lack of adaptability, lack of expandability, cost ineffectiveness in System Development Life

Cycle (SDLC), etc. The so-called closed structure is the typical structure that does not provide flexible integration process, such as plug-and-play, for external components. Oppositely, an open structure for DSS supports the plug-and-play integration process.

A recent research survey [9] indicates that there is a widening gap between DSS research and practice. Several researchers have been focusing on identifying the causes of this problem, which is motivating a paradigm shift for DSSs [9, 14, 42]. Based on the previous findings, we discovered an important factor that separates DSS research from practice, that is: the lack of a standardised and compatible structure for various DSS applications. This limitation causes the difficulties in finding an efficient method to evaluate the feasibility, efficiency, and accuracy among various DSS applications. Many efficient DSS design methodologies cannot be recognised in other design frameworks because of their incompatible design structures. Much existing DSS research suffers from this limitation, which affects numerous effective functional designs being extensively applied to various DSS applications.

The discipline coherence problem indicated in the same survey further reflects the above limitation. The survey shows that DSSs have marked disconnects between important sub-fields, such as group support system, intelligent DSS, and data warehousing [9]. Various DSS sub-fields have been established in the past three decades, however there is no such an open and standardised method to easily connect these sub-fields.

In these circumstances, agent technology is suggested as a desirable method to fulfil the demands for a paradigm shift for DSSs. Agent technology can be applied to existing systems in order to transform a DSS into an open system. Multi-agent technology is a superior solution that can enhance a system's connectivity, extensibility, reusability, and manageability. The reasons for the success of multi-agent application in DSS are the features of an intelligent agent, which are: goal-oriented, collaborative, flexible, self-starting, intelligent, mobile, and interactive (see Chapter 3, Section 3.1; and Chapter 5, Section 5.1).

1.1.2 Problems in current agent-based DSSs

Current applications of multi-agent technology in DSS have been mainly focusing on decentralized frameworks. For instance, a multi-agent decision support system in transportation management has been proposed and implemented [114]; the use of intelligent agents in situated systems has been suggested [146], and the Prometheus

methodology for building Belief-Desire-Intention (BDI) based distributed agent systems has been introduced in [116]. However, these systems require highly efficient concurrent control and synchronous communication mechanisms.

Unfortunately, current agent-based DSSs are not efficient enough, particularly, when they deal with large volumes of transactions because most of these systems are based on decentralised structures. Many decentralised multi-agent systems suffer from inefficient manageability [105], particularly, with respect to the difficulties of handling concurrent control and synchronous communication problems. The detailed analysis of agent network topological performances can be found in Chapters 7 and 8.

Moreover, existing agent-based DSSs tend to focus on individual agent or agent group design, such as agent-facilitator-based structure or agent-mediator-based structure. They often neglect the design of the supporting environment for agents or agent groups. This limitation causes the difficulties of agent self-upgrade and agent coordination in many existing DSS applications because a single agent has limited capabilities for acquiring information from a dynamic environment. In other words, an agent-based DSS application is not likely to be efficient in adapting to the dynamic world without offering an open and comprehensive framework for managing and coordinating agents. Therefore, creating an efficient environment to support agents' activities, which include agent cooperation, agent matching, agent upgrade, etc., is the key to an agent-based DSS.

In brief, the major problems exist in current DSSs can be basically summarised as:

(a) The lack of system connectivity (particularly for traditional DSSs);
(b) The lack of a uniform design methodology;
(c) The lack of efficient system manageability;
(d) The lack of a competent supporting environment for agents to perform activities such as self-upgrading, reuse, etc.

1.1.3 Motivations

Based on the above problems in existing DSSs, we summarise the major issues that motivate this research as follows:

(a) The requirement for an open and compatible framework for DSSs;
(b) The requirement for the standardisation of DSS developments;
(c) The requirement for intelligent and active DSSs [6, 85];
(d) The requirement for an efficient structure for communication and cooperation;

The next section further explains how this research contributes to fulfil the above motivations.

1.2. Thesis Contributions and Research Methodologies

1.2.1. Thesis Contributions

The proposed AOCD framework aims to tackle the major problems described in the previous section. The major contributions of this thesis are:

- This thesis suggests an open framework for DSSs, i.e. the AOCD framework, based on a Matrix-agent connection design (see Chapters 3 and 4). The use of Matrix-agent connection design in DSS applications could simplify current intelligent agent development procedures and improve the connectivity of a multi-agent system through adopting the plug-and-play design principle.
- An AOCD-based system solves a complex problem in an active and intelligent way through deploying a number of agents (see Chapters 3, 4, 5). Each AOCD-based agent is able to actively seek the cooperation from other agents when it has difficulties in solving a problem. Thus, more creative and innovative decisions can be produced as a result of such cooperation and interaction.
- The novel Matrix concept proposed in this research provides a standardised environment that allows various service-provider agents to be integrated into an AOCD-based system efficiently (see Chapters 3, 4). The Matrix design is standardised so as to allow external agents to be used in any Matrices in various organisations. This design is able to accelerate the standardisation process for various DSS applications. This research suggests a living environment for agents (see Chapters 4, 5). The Matrix plays an essential role in providing agents with a living environment. The Matrix not only acts as an agent facilitator or mediator that mainly deals with agent coordination, but also incorporates agents into its virtual space in which agents can obtain information, self-upgrade, and be reused.
- In this research study, a set of agent network topological experiments have been carried out based on the AOCD framework to find out the communication and cooperation efficiency of this framework (see Chapters 7, 8). This research also systematically categorises the agent network topologies through incorporating agent mobility and intelligence issues. The selection of an appropriate agent

network topology for a multi-agent system is essential to the success of system design. The hybrid topology adopted in the AOCD framework improves the system manageability and expandability through combining the centralised and decentralised topologies. Current work on agent network topologies is insufficient, which brings this part of work into prominence.

- Several new research mechanisms have been suggested in this thesis including the agent-rank algorithm (see Chapters 6, 8) and the topological description language for agent networks (see Chapters 5, 7). The agent-rank algorithm provides efficient methodology to match a requesting agent amongst numerous service-provider agents. The novel topological description language for agent networks enables an individual agent to perceive the structural information about the network that the agent resides in.

1.2.2. Research Methodologies

To achieve the research objectives, various research mechanisms have been performed and applied to this project. Basically, this research project consists of five major stages, which are listed as follows:

- The first stage includes the problem analysis and literature review. In this stage, the research problems were identified and an extensive review of relevant literature was conducted.
- The second stage mainly focuses on the infrastructure design of the AOCD framework, which includes the analysis of agent network topologies, the evaluation of the agent network topological experimental results, and the design of the Matrix-agent connection. The performance analysis of various agent networks based on different topologies has been carried out in this stage. This analysis helps to conduct the comparison among the common agent network topologies, which enables the system designers to adopt an appropriate agent network topology for their agent framework. Based on the performance analysis, we developed an agent-topology-based experimental program to further evaluate the previous analysis. The design methodologies adopted in this stage underlie the principles of designing the backbone of the AOCD framework.
- The third stage involves the development of the Matrix and the unified Matrices structure. The detailed Matrix design includes the Matrix layer design, the Matrix

control panel design, and the reuse centre design. The unified Matrices structure further extends the AOCD framework, which allows various Matrices to work cooperatively. The Matrix design methodologies emphasise the development of an open environment for agents.

- The fourth stage provides the detailed agent design for the AOCD framework, which includes the agent architectural design, agent interface design, agent communication language, and agent development life cycle. The agent design mechanisms are based on previous research studies and tailored for the AOCD framework, especially for the Matrix-agent connection.
- The last stage focuses on the agent cooperation in the AOCD framework. An AOCD-based agent might constantly seek for an appropriate corresponding agent, thus an efficient agent matching mechanism is vital to the success of the AOCD framework. A novel agent-rank algorithm is developed for agent matching within the AOCD framework. Based on this algorithm, we developed a computational program to evaluate the final ranking results.

In brief, the AOCD framework is based on the research methodologies described in these five stages. Each stage has a high degree of independency but also interrelated with the other stages.

1.3. Thesis Outline

This thesis consists of nine chapters. Following this chapter, Chapter 2, *Literature Review*, reviews the previous studies on traditional DSSs and existing agent-based DSSs. A brief introduction on the DSS development history and the future direction for DSSs are provided. An extensive study on existing agent-based DSSs has been conducted and the limitations of these DSSs have been analysed.

Chapter 3, *Basic Structure of the AOCD Framework*, clarifies the basic concept of the AOCD framework and its major components. The Matrix and unified Matrices concepts are explained. The AOCD network structure is analysed based on the AOCD-based topological experiments. A case study is also provided in this chapter to further explain the efficiency of the AOCD framework.

Chapter 4, *Matrix Design and the Unified Matrices Systems*, describes the detailed design of the Matrix and the unified Matrices structure. The Matrix structural design, which includes its major components and the four-layer design, is introduced. The

unified Matrices structure suggested in Chapter 4 allows various Matrices to work cooperatively. A novel searching algorithm and the super-node structure are deployed in this structure in order to improve the cooperation and coordination efficiency among numerous Matrices.

Chapter 5, *Agent Design Methodologies*, presents the detailed design of the AOCD-based agents, including the agent architectural design, agent communication methodologies, agent capability descriptions, and the AOCD-based agent development life cycle. The AOCD-based agent design adopts previous agent design methodologies and specifically tailors them for the AOCD framework.

Chapter 6, *Agent Matching Mechanisms*, introduces the agent matching mechanisms in the AOCD framework. An effective agent cooperation process is essential for the AOCD framework, thus the agent matching process is particularly important for the AOCD framework. A novel agent-rank algorithm is introduced to enhance the agent matching efficiency, especially in the AOCD framework. The two parts of the agent-rank algorithm are illustrated respectively.

Chapter 7, *Agent Network Topologies and Theoretical Analysis*, provides a systematic analysis of agent network topologies. This is a major contribution of this thesis. Existing topological theories in the multi-agent area are inadequate. Therefore, the topological theories for agent networks are exceedingly important. This chapter classifies agent network topologies and analyses the performance of hybrid topology in the AOCD framework. Furthermore, a novel *Topological Description Language for Agent networks* (TDLA) is introduced in this chapter (the application of TDLA has been described in Chapter 5). This novel language provides an agent with the information about how the agent network, where the agent resides in, is constructed. The use of TDLA allows agents to perceive the information on network structures before they travel to another site or search for a corresponding agent.

Chapter 8, *Experimental Results Evaluation*, illustrates the experimental results based on the AOCD-based agent network topological experiments and the computational program for the agent-ranking algorithm. The experimental results evaluate the network performance of the AOCD framework, and demonstrate the agent-ranking scores according to a specific case.

Chapter 9, *Conclusions*, concludes the findings and contributions from this research. It also suggests the possible directions for future research.

CHAPTER 2
LITERATURE REVIEW

2.1. Introduction

The traditional Decision Support System (DSS) presents a passive form of support that requires users to have good knowledge of how to use the relevant models, data sources, and other tools and take initiative to perform all necessary operations in a DSS effectively [148]. This problem plagues the development of DSS since its early introduction by Gorry and Scott-Morton [65]. Moreover, many other limitations of DSS also hold back its development in the real world (see Chapter 1 Section 1.1). The rising critiques of the inefficient performance of existing DSSs in the dynamic environment motivate a fundamental shift for DSS architecture design [6, 27].

The development of DSS has been through several stages, and a variety of DSS technologies have been applied extensively in many domains. From the earlier DSS software, such as Simscript and DENDRAL to current MicroStrategy and SAP BW, the research and development in DSS have made reasonable progress however future DSS are facing more complex problems, such as constant upgrade, unpredictable external factors, and frequent cooperation with external components. For example, a DSS application for stock market might need to constantly update its sub-functions.

The emergence of intelligent agent technology has been embraced by researchers and experts in the DSS area since intelligent agents can offer many capabilities for solving the inherent disadvantages of traditional DSS. In this chapter, we briefly review the development of traditional DSS methodologies and then focus on the development of existing agent-based DSS. Existing agent-based DSSs are various but the lack of standardised design methodologies has become a major problem that plagues the development of DSS. Thus, we suggest an agent-based open architecture for DSS, which deploys a unique component, i.e. Matrix. This framework not only standardises the design methodologies for different applications but also supports the varieties of individual agent designs. The structure of this chapter is organised as follows. The next section reviews the development history of DSS and indicates the trends of future DSS developments. Section 3 illustrates some existing agent-based DSSs. Section 4 introduces the agent society and the Matrix concepts. The last section provides a brief overview of the major agent design mechanisms.

2.2. Development of Traditional DSSs and Agent-based DSSs

The history of computing programs for decision-making dates back to the early 1960s when various simulation packages, such as Simscript, were developed to solve the modelling problems.

The clear description of the term 'Decision Support System' was introduced by Gorry and Scott-Morton [65] in 1971. They described a DSS as "an interactive computer based system that helps decision makers utilize data and models to solve unstructured problems". This definition is later improved by Alter [4] and Keen and Scott-Morton [86], who further expanded the DSS capabilities to solve semi-structured problems. A recent description of DSS has been given by Alter [3], which describes a DSS is "an interactive information system that provides information, models, data manipulation tools to help make decisions in semi-structured and unstructured situation where no one knows exactly how the decision should be made."

The development of design and implementation technologies in DSS has come through different stages. According to Sen [129] during the period of 1980 to 1994, the Model Formulation System (MFS) architecture became one of the major DSS architectures. MFS architecture applies modelling techniques to integrate components into a DSS framework. In the early 1990s, AI's control-oriented research led to the creation of the control-intensive MFS architecture. Such MFS architectures emphasize hierarchical and opportunistic control paradigms for model formulation [130]. Nevertheless, the MFS architecture focuses less on flexibility and intelligence. Apart from MFS, other novel architectures have been proposed in this period; for instance, Angehrn introduced the conversational framework for decision support, which is the basis of a new generation of active and intelligent decision support systems [6].

Several contributing disciplines in the DSS area have emerged since the 1990s [41]. These emerging areas include: MCDSS (Multiple Criteria Decision Support Systems), GDSS (Group Decision Support Systems), Web-based DSS, and intelligent-agent-based DSS. On-Line Analytical Processing (OLAP) is one of the most popular technologies in the DSS area since 1995. The data warehouse and data mining technologies are essential in many OLAP-based applications. OLAP enables users to retrieve data from the data warehouse for decision-support purposes.

Since the late 1990s, research work in the DSS area has been focusing on Internet and artificial intelligence-based studies. For instance, Chi and Turban's [30]

conceptual framework for a distributed artificial intelligence (AI) system is a typical shift from traditional DSS to AI-based DSS. The applications of agent technology in DSS have been rising gradually in these years especially in the multi-agent area. The idea of multi-agent decision support evolves from Hewitt's actor [73] concept to the modern concept of agents. This modern concept integrates a collection of functionalities, achieved by the interplay of both knowledge about certain problem types and about the environment in which the agent operates. In 1993, Angehrn described a new DSS framework, which utilizes intelligent agents. This approach transforms a DSS into a decision-making arena in which users define and explore their problems under the continuous stimulus of agents [6]. In 1999, the term of 'multi-agent decision support' was suggested [32]. In 2003, the concept of decision station was introduced [147]. After that, the multi-agent decision support system in transportation management has been proposed and implemented [114]. Nowadays, more and more DSSs have implemented or are starting to use intelligent agents in their implementations. Recent technical trends show that future DSSs will become more active, intelligent, and open. Many researchers in the DSS area have realised the significance of implementing agents in DSSs. Based on previous studies on the DSS history [3, 129, 132], the following table lists a brief review of the DSS history.

Table 2.1. Brief Review of DSS Development History (Major mechanisms).

Year	DSS Development Taxonomy	Software/Applications
Pre-1960s	Procedural-code-based algorithms	Operations research, statistics, and forecasting applications.
1960s	*Package* concept for decision-making	GPSS, Simscript
1970s	DSS concept suggested	DENDRAL, MYCIN, SABRE.
1980s	Group DSS, Model Formulation System	RTCAL, Decision conference, GROUP DECISION AID.
1990s	OLAP, data mining, Web-based DSS	Sagent, SAP BW, MicroStrategy.
2000 onwards	Active DSS, Intelligent DSS, Agent-based DSS	(Emerging market….)

Table 2.1 outlines the major stages of DSS development and typical DSS software applications. Although each stage is explicitly classified, several stages are still overlapping with others; for instance Group DSSs were still used in the 1990s but the

major development of Group DSS occurred in the 1980s. This taxonomy indicates the major development period of each mechanism; however the boundaries between any two stages in Table 2.1 are not strictly divided.

In recent years, the term 'Intelligent DSS' has been suggested in much literature work [30, 67]. More and more researchers believe future DSSs need to become more intelligent, active, and open to adapt to the dynamic environment. The emergence of agent-based technologies makes the realization of becoming intelligent, active and open much easier in a DSS. Moreover, the agent-based technologies exhibit the high efficiency in tackling various DSS problems. Undoubtedly, the agent-based DSS research will become a major or dominating area for future DSS.

2.3. Current Agent-based DSS

Existing multi-agent technology has been implemented extensively in different applications in business processes and information systems. Modern business systems such as DSSs are becoming large, open and distributed. Current computing platforms supporting modern business processes are no longer stand-alone systems, but have become tightly connected both with each other and their users [158]. Moreover, current information systems such as DSS, require constant system updates that demand very flexible and reusable system architecture. To become more efficient and effective in a complicated environment, the systems have to be more independent, flexible and intelligent. Agent-based systems provide those technologies that are urgently needed for modern business systems especially DSS.

Multi-agent research is an emerging interdisciplinary research area, which offers a promising future for DSS. The idea of multi-agent decision support has evolved from Hewitt's actor concept [73] to the modern concept of agents, which integrates a collection of functionalities, achieved by the interplay of both knowledge about certain problem types and about the environment in which the agent operates. Agent-based systems provide a way of conceptualising complex software applications that face problems involving multiple and distributed sources of knowledge [102]. Multi-agent systems have the capacity to offer several desirable properties, which have been summarized by Weiss [158] as follows: Speed-up and efficiency, robustness and reliability, scalability and flexibility, cost effectiveness, and reusability. As mentioned in Section 2.2, various DSSs have adopted multi-agent technology in their

applications: for instance, a multi-agent-based DSS for transportation management proposed and implemented in [114]; a multi-agent-based DSS for medicine management has been suggested in [31]; and Gadomski *et al* [57] develop an intelligent DSS for emergency response, etc. Existing agent-based DSSs, such as the above applications, improve system performances compared with traditional DSS. However, we discovered several vital limitations of these systems based on various literature studies and Section 2.

First, most of the above multi-agent-based DSS systems implement either a centralized topology or a decentralized topology [174], which are not efficient and robust enough to handle various problems in the complex environment. On the contrary, the problem can be avoided in our proposed AOCD architecture through using a unique structure that allows both central control and peer-to-peer connection. This structure eliminates the problems of concurrent control and synchronous communication and also enhances the communication efficiency through peer-to-peer connections.

Secondly, existing agent-based DSS applications lack compatibility and reusability [9, 14, 42] since there is no universal design standard. The cooperation and communications between two existing agent-based DSS are not efficient as their design methodologies are various and often incompatible. Thus, AOCD deploys the Matrix as the coordinating component that also provides the virtual living space for agents, instead of continuously using agents to coordinate multi-agent systems [169].

Thirdly, existing agent-based DSS often neglect the supporting environment of agents or agent groups [169]. In these circumstances, most existing agent-based DSSs are unable to provide a fundamental solution to enhance the manageability, connectivity, extensibility, and transferability of a multi-agent system. Thus, the concept of Matrix was suggested [169]. The Matrix concept adopts the agent society concept that allows agents to communicate with their environment and go through their life cycle in their virtual space.

The following sub-sections review some of the major agent-based DSSs to further explain their design methodologies.

2.3.1 Middle-Agent-based DSS

The middle-agent-based model is one of the most common models used in current agent-based DSS applications. In such a model, one (or several) middle agent(s) is

responsible for the cooperation and coordination between agents. These middle agents are also known as coordinating agents or agent facilitators. This is an efficient model in terms of manageability and adaptability. The following two typical middle-agent-based DSSs illustrate the basic design methodologies of middle-agent-based DSS.

In [174], an agent-based framework is suggested for solving complex problems as shown in Figure 2.1. The 'Decision Making Agent' in this framework also called as 'Problem Solving Agent'. In this framework, an agent can join the system through registering its information with the middle agent, or leave the system by removing the registration information from the middle agent [174]. This framework exhibits high manageability in coordinating multi-agents to solve problems. All the agents in this framework need to communicate with the middle agent when they request cooperation from other agents. The planning agent is responsible for generating work plans for the tasks received from the interface agent, which is originally sent by users. The planning agent delivers its work plan to the middle agent for organising problem-solving agents to resolve the request. Based on this framework, agents are able to provide predictive features for decision-making since planning agents and aggregation agents are efficient in combining various factors. In [174], a financial system has been developed to predict the interest rate through using fuzzy logic and genetic algorithms to combine various financial factors such as GNP, CPI, etc.

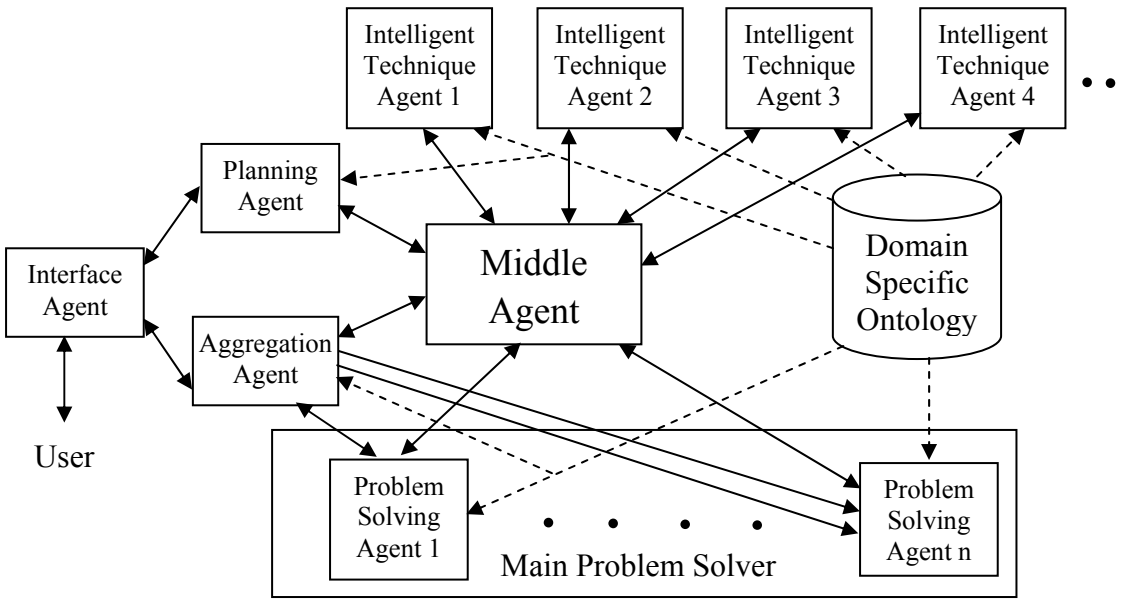


Figure 2.1. General Framework of Agent-Based Hybrid Intelligent Systems [174].

However, there are several crucial limitations in the framework. First, the probability of causing overload in the middle agent is high since the communications between agents mainly rely on the middle agent. Second, the lack of direct communication between problem-solving agents will cause inefficiency in solving a complex problem since the middle agent is involved all the time and that increases the time delay. Finally, the absence of a translation component will limit the variety of agent communication capabilities in the system. Current agent communication languages are various; different applications might adopt different agent communication languages. The above framework does not include a translation function, which could imply further limits to implementing various agent communication languages in this framework.

Another typical middle-agent-based DSS is called MASST (Multi-Agent System for Stock Trading), which employs a middle-layer agent system based on a coordinator agent as shown in Figure 2.2 [97].

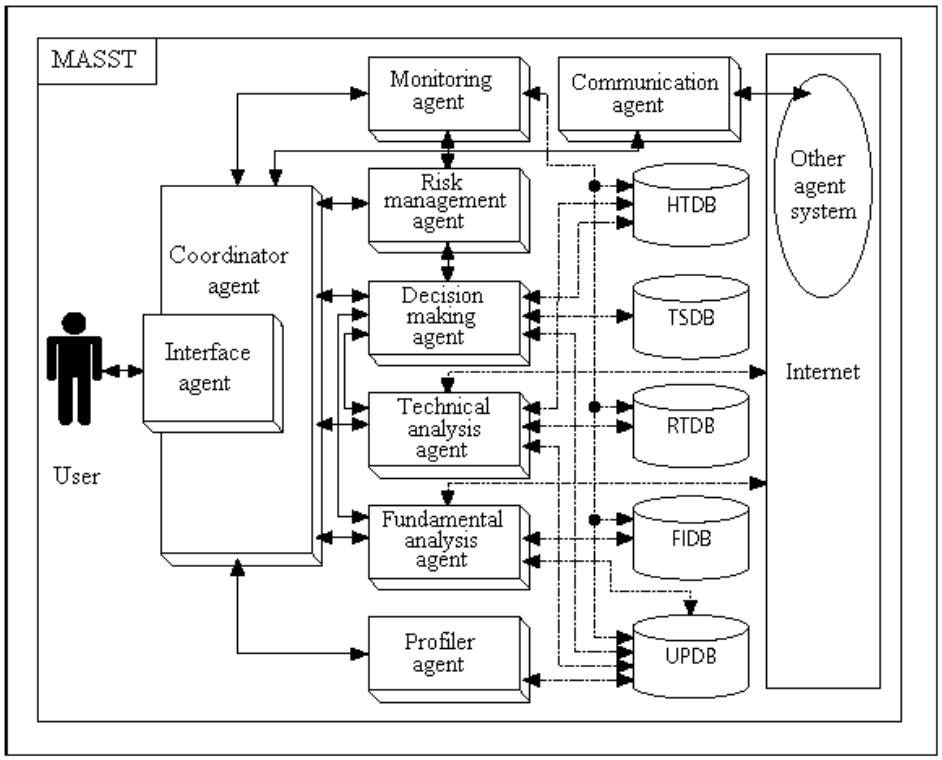


Figure 2.2. MASST Framework for Stock-Trading DSS [97].

The framework of MASST has several similarities with the AOCD framework. First, both of them deploy a coordinating component for agent cooperation, which in the MASST is called Coordinator agent and in the AOCD is Matrix. Second, the agents in MASST not only can communicate with the Coordinator agent but also can

communicate with some other agents. It is similar to partially linked peer-to-peer agent network construction. In the AOCD framework, the connections between agents or Matrix-agent are more powerful, which is due to the combination of the centralised connections and fully linked peer-to-peer connections. Third, both frameworks can communicate with external systems. In the MASST framework, the communication between two different systems is mainly through the Communication agent. Some domain agents also can communicate with the external environment through the Internet. In the AOCD framework, the communication between local and external systems is through three different ways including Matrix-Matrix, agent-agent, and Matrix-agent. The AOCD framework deploys a more flexible and efficient external communication mechanism, which allows each component in the framework to communicate with the external environment efficiently.

Overall, the MASST framework is efficient for solving predictable problems in a specific domain. However, this framework has certain limitations including inefficiency in cooperation with external systems, the overload problem in the coordinator agent, and the absence of a self-upgrade mechanism for the coordinator agent. In the AOCD framework, the Matrix can perform self-upgrade through directly acquire information from the external environment, but the coordinator agent in the MASST framework has no direct connection with any databases or the external environment.

2.3.2 Decentralised Agent-based DSS

Existing decentralised systems have been applied extensively in many areas, such as the online peer-to-peer file sharing applications [77, 124], decentralised DSSs [55], decentralised multi-agent systems [74], etc. However, the applications of the decentralised model in agent-based DSSs have not been widely accepted. In this section, we briefly review existing decentralised DSSs and discuss the limitations of the decentralised agent-based DSS that hinder its development.

Gachet *et al* [55] have suggested a decentralised structure for DSS. Gachet's DSS framework is not agent-based, however, this distributed DSS framework can be efficient through using agent-based technology since agents have strong autonomy, cooperativity, and mobility which are needed by many decentralised frameworks.

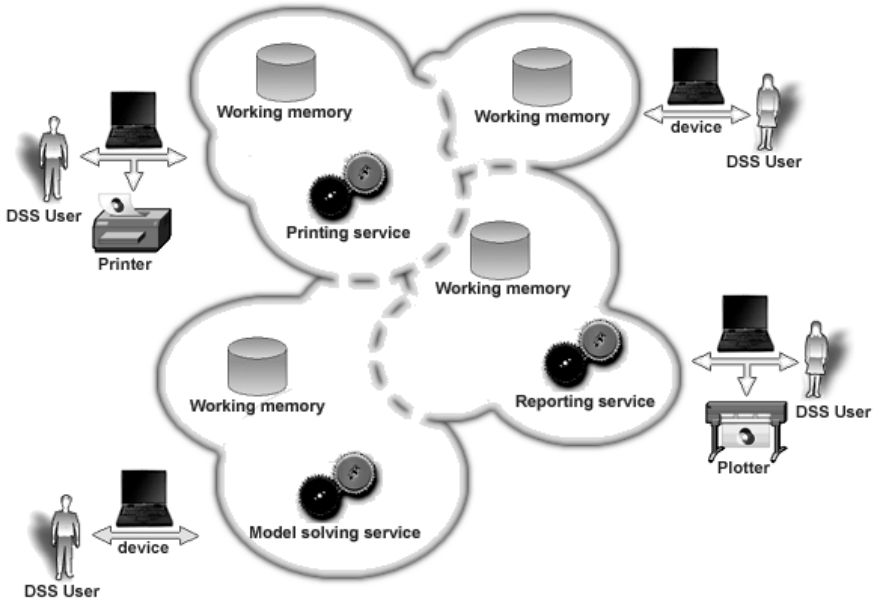


Figure 2.3. Distributed DSS architecture (Revised based on [55]).

A distributed decision support system is described as "a collection of services that are organised in a dynamic, self-managed, and self-healing federation of hard- and software entities working cooperatively for supporting the solutions of semi-structured problems involving the contributions of several actors, for improved decision-making" [55]. In this distributed DSS architecture, the DSS users in different locations can access an external DSS working environment through a network. A *working environment* represents a node (or peer) able to communicate with other nodes (that is, other working environments) in a federation.

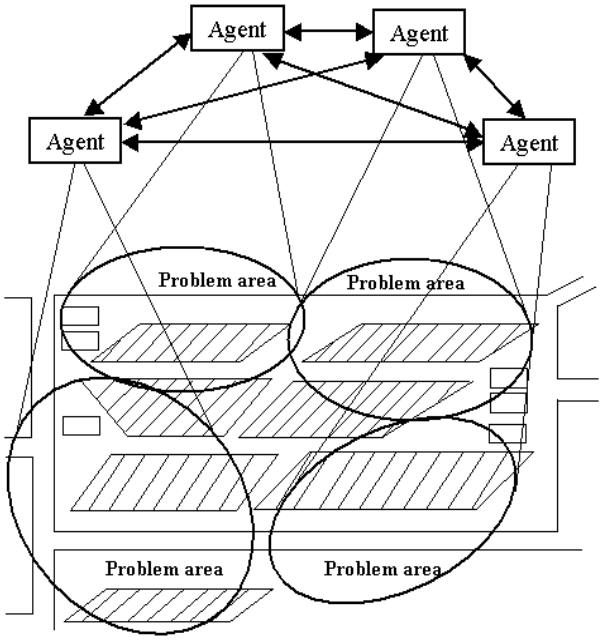


Figure 2.4. A Decentralised Multi-agent Traffic Control Architecture [32].

Numerous agent-based decentralised DSS frameworks have been suggested in previous literature studies, such as a decentralised multi-agent architecture for traffic control [32] and the Agent Grid Based Open DSS (AGBODSS) framework [29].

The decentralised multi-agent architecture for traffic control allows peer-to-peer communication in the traffic control DSS as shown in Figure 2.4. The adoption of the decentralised modes of coordination in this framework enhances communication and cooperation efficiency in the multi-agent framework.

In this decentralised multi-agent framework, each agent is responsible for a specific problem area and is able to cooperate with other agents for efficient traffic arrangements. This framework allows peer-to-peer communication between agents.

The agents in the AGBODSS framework are distributed and allocated in different groups. The agent groups are supported by the agent grid services and other lower level grid services, see Figure 2.5.

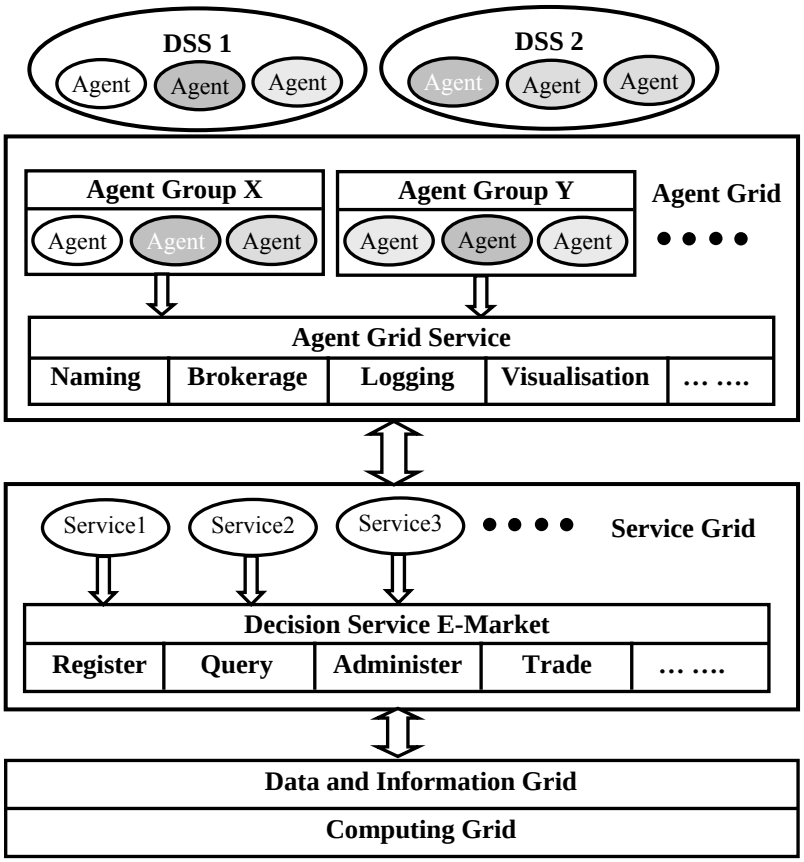


Figure 2.5. The Open DSS Model Based On the Agent Grid [29].

A DSS application in this framework consists of heterogenous agents and each agent has certain capabilities to solve problem in a specific domain. This concept is similar to the AOCD framework, in which a group of agents are organised by a Matrix

to form a DSS application. Notwithstanding the claims of efficiency of the AGBODSS framework, the framework is not mature enough to verify its performance, particularly with respect to the agent cooperation and communication issues.

The above decentralised-DSSs have certain advantages, such as efficient cooperation and communication, superior extensibility, and reliable fault-tolerance. However, most of the decentralised systems are struggling with a major problem that is: the inefficient manageability. There are many factors that affect the manageability in a decentralised agent-based DSS including (*i*) the difficulty in concurrency controls, (*ii*) the communication barrier between two agents (since heterogenous agents may adopt different agent languages yet it is not feasible for every agent to carry a translator that recognises all the possible agent languages), and (*iii*) the inefficiency in searching and matching processes (most of the searching and matching processes are based on individual agents because there is no centralised information for searching or matching in decentralised systems). These factors cause numerous problems for most of the decentralised applications, and sometimes even lead a decentralised system to complete failure.

2.3.3 Web-based Intelligent DSS Utilising Agents

The tremendous Internet revolution changed people's daily life and also inspires many new technologies, such as web-based intelligent DSSs.

The Web-based intelligent DSS applications, such as OLAP, spread quickly to many areas since the Internet has become one of the world's largest networks. The major advantages of developing web-based DSS can be summarised as follows: (*i*) improve cooperation efficiency as it is easy to share information in Web-based systems; (*ii*) improve development efficiency, it is easy to design and maintain a Web-based system since many Web-based software packages are easy to use and the re-coding processes are also simplified; (*iii*) increase implementation feasibility, Web-based technologies offer abundance developing tools, such as Java, HTML, ActiveX, ASP, etc, to support system implementations; and (*iv*) provide superior reusability, the components and sub-systems of a Web-based system can be easily reused in a new system because Web-based technologies provide numerous technologies, like XML, that enable the combinations of hybrid systems.

The advantages of utilising the agent technology in Web-based DSSs have been discussed in the previous literature [94, 153], which include (*i*) enhancing autonomy; improving system reliability (overcoming the vulnerability of Web security technologies), (*ii*) improving intelligence in reasoning processes, and (*iii*) modularising system structures through replacing sub-systems by agents.

Figure 2.6 shows the system structure of a Flexible Web-Based Decision Support System Generator Utilizing Software Agents (FWDSSG) [35]. This framework consists of heterogenous agents; each agent takes responsibilities for several specific tasks and interacts with the other agents through the coordination of an agent manager. The user requests the agent manager to execute the tasks and obtains the results back from the agent manager. The DSS component resources located in various web sites and possessed by a number of agents including model agent, data agent, solver agent, visualisation agent, and control agent. Thus, a complex problem can be tackled through cooperating these component agents.

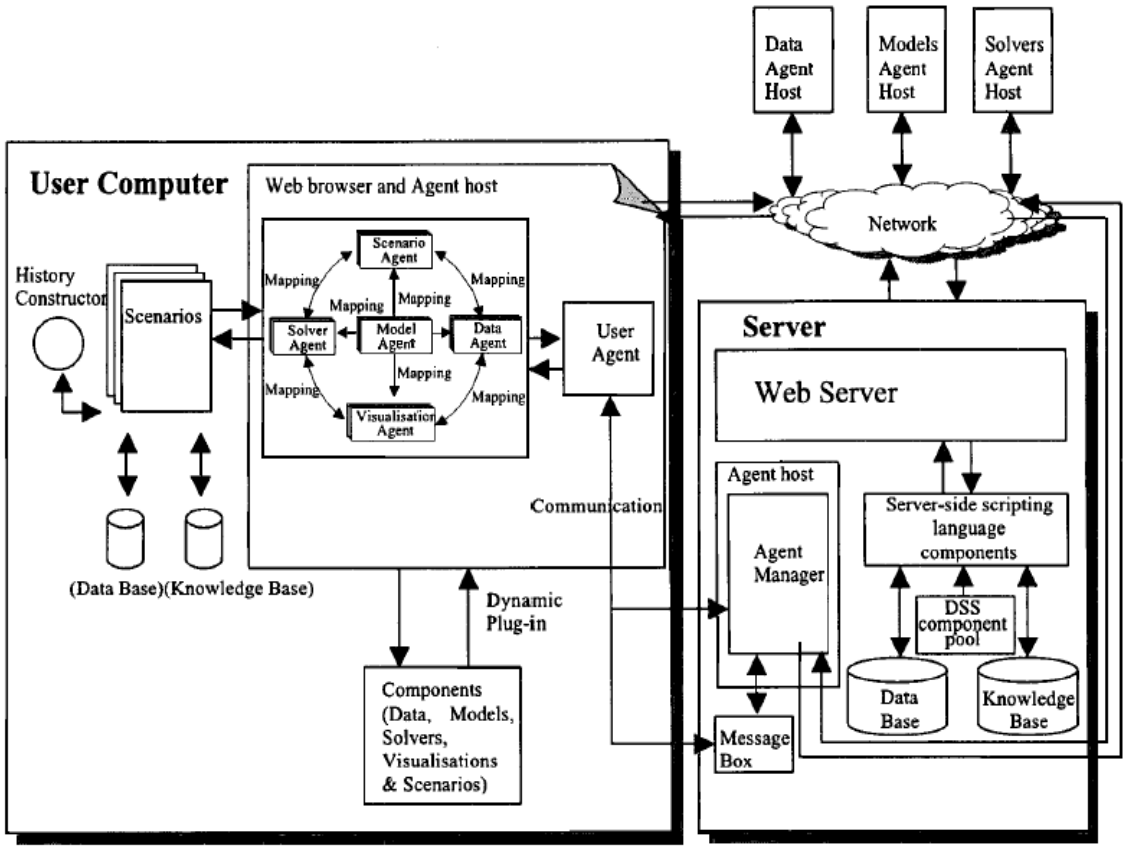


Figure 2.6. System Framework of FWDSSG [35].

Similar frameworks also can be found in [36] and [153]. The rising demands of agent technology in Web-based DSSs indicate this mechanism is becoming a promising direction for future DSS development. However, we argue that the design of utilizing agents in Web-based DSSs needs to overcome the major problems in order to become a mainstream design in the DSS area. Current Web-based DSSs have limited consistency and security. Moreover, most existing Web-based systems are insufficient to solve complex problems as the low performance of computation and communication in Web-based systems.

2.4. The Agent Society Concept and its Evolution

Agent society concept has been suggested in previous works of literature [76, 78]. However, this concept is still preliminary and has not been given a unified definition. To clarify this concept, we begin with the process of problem solving.

In order to solve a complex problem, an agent often cooperates and coordinates with other agents within a multi-agent framework. Therefore, agents do not function alone, rather they exist in an environment where they can interact and cooperate with each other. In [75], an agent society is described as: the societal construct that provides a common domain through which agents can act and communicate. This phenomena is very similar to human society, thus the term ‘society’ has been brought into the multi-agent area. The society of agents can be regarded as the environment where a group of agents reside, and each agent with different roles has social dependency on this environment, i.e. agent society.

Agent societies have several characteristics, which are listed as follows. First, an agent society has strong social commitments, which are the commitments of an agent to another agent. Second, an agent society contains a set of constraints imposing on the behaviours of agents, i.e. *social laws*. Thirdly, an agent society should be an open environment to support heterogenous agents and provide them with various information sources. Heterogenous agents in this thesis represent AOCD-based agents that have similar structures but various capabilities.

Evolved from the agent society concept, several other forms of agent organisations are suggested, such as the agent federation concept [75] (see Figure 2.7) and the Matrix [168]. In the agent federation concept, each agent group is an agent federate. Each agent federate consists of a number of agents and has an agent acting as the

intermediary. As shown in Figure 2.7, the white agent situated at the edge of each federate is the delegated intermediary.

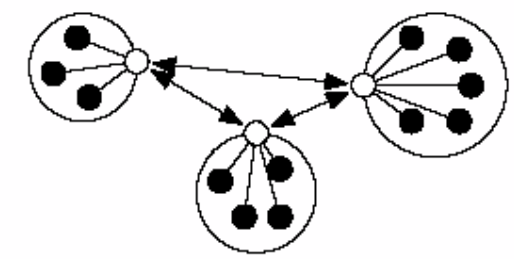


Figure 2.7. An Agent Federation [75].

The Matrix concept extends the agent federation concept. Each Matrix is similar to an agent federate that contains a number of agents. The Matrix plays a role as a virtual space or a virtual platform to agents. The agent federation does not provide such capabilities; it simply groups a number of agents and employs an intermedia agent to coordinate. Agents in the Matrix form an agent community or society in which agents play different roles to solve different tasks. The agent society concept described by Huhns and Stephens [76] does not specify the living environment for agents. Existing research work has already started to develop some agent-based virtual world applications, such as [99]. Nevertheless, the existing work in this area is rather preliminary and has very limited impact on DSS. The use of Matrix-agent connection design in DSS applications could simplify current intelligent agent development procedures. This Matrix concept improves the openness of a multi-agent system through providing a standardized platform, which incorporates the plug-and-play design. The Matrix concept extends the agent society concept by developing some concrete design methodologies and uses a core component, called a *Matrix control panel*. The previous work on the Matrix design includes the AOCD framework design [167], the Matrix control panel design [169] and the agent topological analysis of three agent topologies based on the Matrix-agent connection design [170].

2.5. Agent Communication and Cooperation Design Methodologies

Agent design methodologies are the basis for the AOCD framework. Existing agent design methodologies are various. In this section, we mainly review three aspects in the agent design area, which include the agent communication languages, agent matching methodologies, and agent capability description.

2.5.1. Agent Communication Language

Agent Communication Language (ACL) is expressly tailored to support the collaboration, negotiation, and information transfer required in agent interactions [28]. For the past decade, ACL has been one of the most important issues in multi-agent area. Since 1990, ACLs have been developed based on various standards. Some of the standards make major contributions to the development of ACL, which include Knowledge Query and Manipulation Language (KQML) and Foundation for Intelligent Physical Agents (FIPA) ACL.

As with many natural languages like English, the emergence of ACL is to support communications between entities. The significant difference between natural languages and ACLs is that natural languages support human communications, whereas ACLs support the communications between agents, and ACLs are more abstract. Differing from many communication mechanisms for exchanging information and knowledge, most ACLs have the following two characteristics [92]:

(i) ACLs handle propositions, rules, and actions;

(ii) An ACL message describes a desired state in a declarative language instead of a procedure or method.

ACLs have many specific designs for agents; however we could not ignore the contributions of natural languages to the development of ACLs. All the existing ACLs inherit the basic concept of the semantic logic of natural language. For instance, most ACLs employ subject-predicate-object structure. The future research work in the ACL area still needs to refer to the development of natural language as a major design guideline.

The Knowledge Sharing Effort (KES), which was initiated circa 1990 by the Defense Advanced Research Projects Agency of the US Department of Defense, is often regarded as the origin of ACLs in several literature works [92, 93, 120]. The KSE model adopts a three-part relationship between: an agent, a content-bearing proposition (e.g. it is cloudy), and a finite set of propositional attitudes in an agent related to the proposition (e.g. believing, asserting, fearing, etc.). The KSE model consists of three layers of representation including: specifying propositional attitudes, specifying propositions, and specifying the ontology of those propositions [92].

KQML emerges in the early 1990s as a standardized agent-oriented language, which is often regarded as the most influential ACL. KQML provides a fundamental

architecture for knowledge sharing through communication facilitators (agents), which coordinate the interactions of other agents [47]. A typical KQML message consists of three layers including: content, communication, and message. Figure 2.8 gives two examples of the KQML messages.

```
(ask-one
    :sender       Gates
    :content      (PRICE Microsoft ?price)
    :receiver     Stock-server
    :reply-with   Microsoft-stock
    :language     LPROLOG
    :ontology     NYSE-TICKS)
```

(a) A query from agent Gates about the price of Microsoft stock.

```
(tell
    :sender       Stock-server
    :content      (PRICE Microsoft 114)
    :receiver     Gates
    :in-reply-to  Microsoft-stock
    :language     LPROLOG
    :ontology     NYSE-TICKS)
```

(b) The stock-server's reply.

Figure 2.8. Examples of KQML Messages (based on [93]).

FIPA ACL is another influential ACL for agent communications. FIPA ACL allows agents communicate with each other through messages (communicative acts). FIPA ACL is developed by the Foundation for Intelligent Physical Agents (FIPA) association. The FIPA's goal is to make available specifications that maximize interoperability across agent-based systems, and producing the ACL specification is one of the major missions of FIPA [93].

The FIPA ACL's syntaxes are almost identical to KQML, however FIPA ACL is more powerful in terms of composing new primitives. A FIPA ACL separates the instructions (like register, advertise, recommend, recruit, broker) from the core communication primitives. These instructions are not speech acts in the traditional sense but do have a useful role to play in conversations and provide the capabilities to manage an agent society.

A FIPA ACL message contains a set of one or more message parameters. The selections of specific parameters for effective agent communication will vary according to the situation. The only parameter that is mandatory in all FIPA ACL

messages is the performative, although it is expected that most FIPA ACL messages will also contain sender, receiver and content parameters. More reviews about FIPA ACL language structure can be found in the *Agent Design* chapter or on the FIPA's official website [51]. Compared with KQML, FIPA ACL is generally regarded as a purer ACL when dealing with registration and facilitation primitives since FIPA ACL treats these primitives as requests for action and defines a range of reserved actions that cover the registration and life-cycle tasks [93]. Nevertheless, FIPA ACL seems to have advantages over the KQML so is more likely to become a leading ACL since FIPA ACL is more powerful in terms of composing new primitives and has well defined semantics in the beginning of its design. The AOCD framework adopts an AOCD ACL, which is based on the FIPA ACL with some modifications to adjust to the unique Matrix-agent design. The details of the modifications of the AOCD ACL also can be found in the *Agent Design* chapter.

Several other agent communication languages also have been suggested, particularly the Web-based ACLs such as XML [63], RDF. However, the implementations and developments of these languages for agent systems are still immature, even not close to a formal comprehensive design.

Research work on developing practical and efficient agent communication languages is still very preliminary. The issues, including pragmatics (i.e., the actual use and effect of utterances in social encounters) [108], standardising various ACLs, and developing ACLs to support efficient reasoning processes, are still major problems in the ACL domain. The future directions of the ACL development (including the AOCD ACL) need to tackle these problems in various domains, particularly in Web-based systems.

2.5.2. Agent-Matching Methodologies

An agent-matching (i.e. agent matchmaking) process enables service matchmaking among heterogeneous agents in a multi- agent framework. Utilizing an appropriate agent-matching mechanism in a multi-agent system will enhance the efficiency and effectiveness of agent communication and cooperation.

There are various research studies on agent-matching methodologies and many existing methodologies have been implemented in some applications, such as *SHADE* and *COINS* systems [91], *InfoSleuth* project [53], *A-Match* Web-based matchmaking

system [118], etc.

The InfoSleuth project has been developed to fulfil the needs for semantic interoperability among information sources and analytical tools within diverse application domains. InfoSleuth is an agent-based system designed to integrate heterogeneous, distributed information sources and tools through utilizing common ontologies [11, 53, 109, 110]. A complex problem is solved in InfoSleuth through incorporating heterogeneous agents. InfoSleuth employs various service agents including broker agents, ontology agents, and monitor agents. The broker agents are responsible for matching requested services with agents, and they technically collaborate to implement both syntactic and semantic matchmaking. Figure 2.9 illustrates a general model of an InfoSleuth application [110].

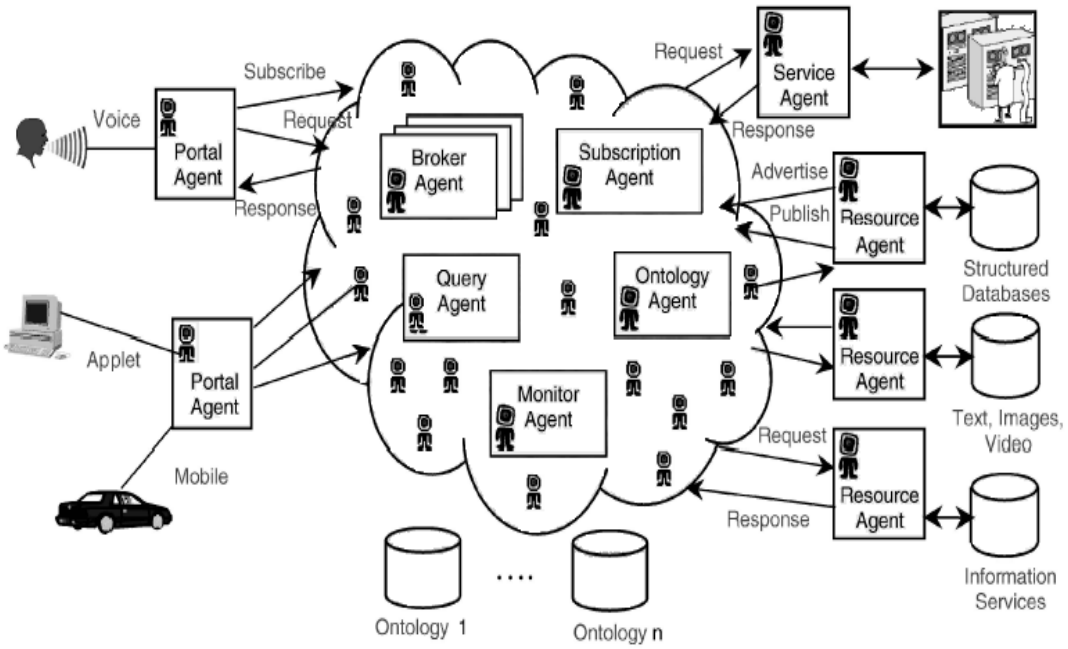


Figure 2.9. A General Model of an InfoSleuth Application [110].

A-Match is a Web-based application for agent-matching processes, which allows human users to find agents that can provide required services. Through the A-Match application, a user can compile a query to an agent matchmaker requesting a specific service. The agent matchmaker matches the user requirements against the advertisements stored in service-provider agents [118, 174].

Several agent-matching methodologies have been introduced in agent matching processes, such as *Language for Advertisement and Request for Knowledge Sharing* (LARKS) [137], DARPA Agent Markup Language (DAML) [139], and *Service Description Language* (SDL) [136]. Most of these methods rely on matching agent-

service-descriptions, also called service advertisements. These descriptions indicate an agent's capabilities and services. In LARKS, there are three matching mechanisms [137, 138]: (*i*) Exact match is the most accurate matching, which requires that both service descriptions are literally equal; (*ii*) Plug-in match is a less accurate matching method in LARKS, in which two matching descriptions are generally similar; (*iii*) Relax match is the least accurate matching method, which determines the similarity of two descriptions by returning a numerical distance value instead of semantically matching the two descriptions.

Most existing agent-matching methodologies are still under developed. Future developments for agent-matching methodologies are facing the challenges of matching in a large-scale multi-agent network. The agent-matching methodologies not only need to consider the accuracy of matching and descriptions based on semantics but also need to consider the efficiency of matching an appropriate agent. A comparative ranking function has been introduced in [89]. However, this early work is based on rather immature agent systems, which do not distinguish P-agent and R-agent. It makes an agent-matching processes extreme complex. In the AOCD framework, a new agent-matching algorithm is introduced to enhance the matching efficiency among heterogeneous agents, especially to reduce the time cost in re-matching processes. This algorithm is based on the Page-Rank algorithm and it distinguishes the P-agent (i.e. service-providing agent) and R-agent (requesting agent) according to the previous identification methods for the heterogenous agents [137, 138].

2.5.3. Agent Capability Description

In a multi-agent system, it is essential for agents to describe themselves to other agents. Existing methodologies for capability description are various, such as EXPECT [61], LARKS [137], ontology fragments (in InfoSleuth) [110], Knowledge Modelling Framework (KMF) [64], Interface Communication Language (ICL) [101], SDL [135], DAML [138], etc. These methods are more or less associated with agent-matching processes, such as LARKS, which have been discussed in the previous section. However, this section places the emphasis on agent capabilities.

EXPECT provides a structured representation of goals and it allows the representation of both general and specialised goals for an agent planning system [61].

The advantage of EXPECT is the ability to use a rule notation to express conditions and effects [2-63]. An example of EXPECT capability description is shown as following (adopt from [2-63]):

```
( ( Name calculate-total-cargo-weight-objects )
   ( capability ( calculate ( obj ( ?w is (spec-of weight ) ) )
       (of  ( ?fms is ( set-of (inst-of object ) ) ) ) ) )
   ( result-type (inst-of weight ) )
   ( method ( sum ( obj (r-weight ?fms ) ) ) ) )
```

An InfoSleuth agent advertises itself to other agents through an advertisement, which contains one or more agent capabilities. The InfoSleuth agents deploy numerous ontology fragments to describe agents' capabilities. In InfoSleuth, the structure for composing individual ontology fragments into capabilities is general; this provides users a uniform view of information and service of the system [110]. These ontology fragments can be easily integrated into an agent's capability descriptions and are also flexible for updating.

As introduced in the agent-matching methodologies section, LARKS mainly deals with matchmaking among heterogeneous agents. In this section, we further review the structure of LARKS. Figure 2.10 illustrates the structure of LARKS.

Context	Context of specification
Types	Declaration of used variable types
Input	Declaration of input variables
Output	Declaration of output variables
InConstraints	Constraints on input variables
OutConstraints	Constraints on output variables
ConcDescriptions	Ontological descriptions of used words
TextDescription	Textual description of specification

Figure 2.10. Structure of LARKS [137].

Every specification in LARKS can be interpreted either as an advertisement or a request; this depends on the sender's purpose. Every LARKS specification must be encapsulated in an appropriate KQML message; and this indicates whether the message content is a request or an advertisement.

KMF is a recently suggested framework, which provides the basis for agent capability description language supporting the automatic, on-demand configuration of agent teams according to stated problem requirements [64]. KMF is a conceptual tool for the solution of the 'bottom-up design problem' in multi-agent systems. There are

three types of knowledge components in KMF including task, domain-model, and capability (see Figure 2.11).

Task	T = <in, out, pre, post>
Capability	C = <in, out, pre, post, asm, kr>
Task-decomposer	D = <in, out, pre, post, asm, kr, st>
Skill	S = <in, out, pre, post, asm, kr>
Domain Model	M = <kr, prop, mk>

Figure 2.11. Three Types of Knowledge Components and Their Main Features [64].

These knowledge components can be re-combined to generate some agent capabilities to fulfil a set of requirements and domain-models. Therefore, the efficiency of reusing agent capabilities can be improved.

Based on the above capability description techniques, the AOCD framework adopts a 'top-down extension tree' mechanism to decompose agent capabilities. This mechanism describes an agent's capabilities through a natural language means. This mechanism provides a general and consistent view of an agent's capabilities, which is what most existing capability description mechanisms lack.

2.6. Agent Software Development Tools

'Intelligent Agent' is no longer a theoretical term since agent software technologies are becoming more mature and practical. More and more agent-based applications have been developed based on different agent software development tools.

Table 2.2. Major Agent Software Tools.

Agent Software Tools	Developer
Cougaar	US Defence Advanced Research Project Agency
FIPA-OS (FIPA-Open Source)	FIPA organisation
Java Agent DEvelopment framework	TILab
JACK Intelligent Agents (JACK)	Agent Oriented Software Inc.

Current agent software development tools for multi-agent system prototyping and application development have been growing very fast in recent years. The major tools for agent software development are listed in Table 2.2.

JADE is currently a leading open source framework for agent software developments. Figure 2.12 shows the distribution of the JADE agent platform over several containers [12]. JADE is composed of eight main packages. Each package

contains a number of Java classes. The JADE platform offers a variety of features, which enable the efficient agent software development processes. Figure 2.13 gives a snapshot of the JADE development GUI [13].

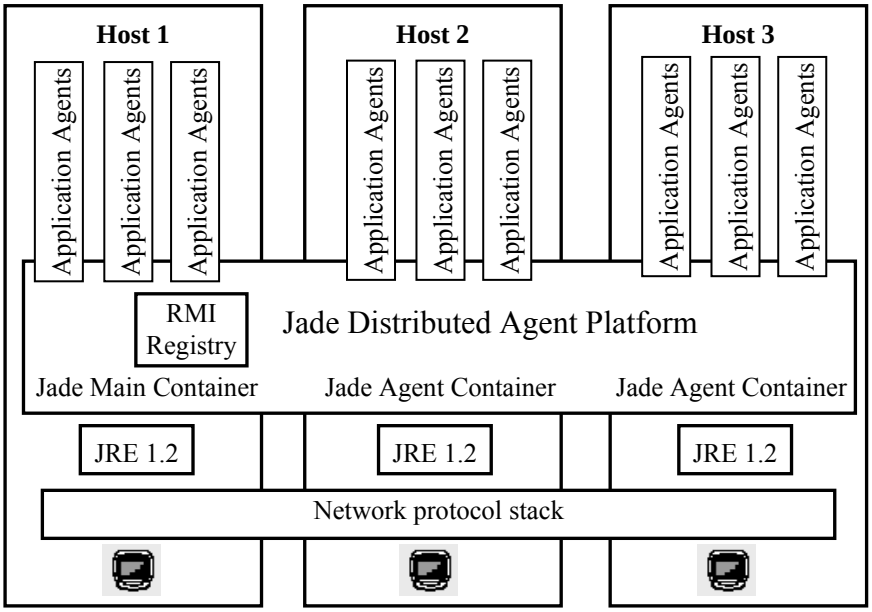


Figure 2.12. JADE Agent Platform Distributed Over Several Containers [12].

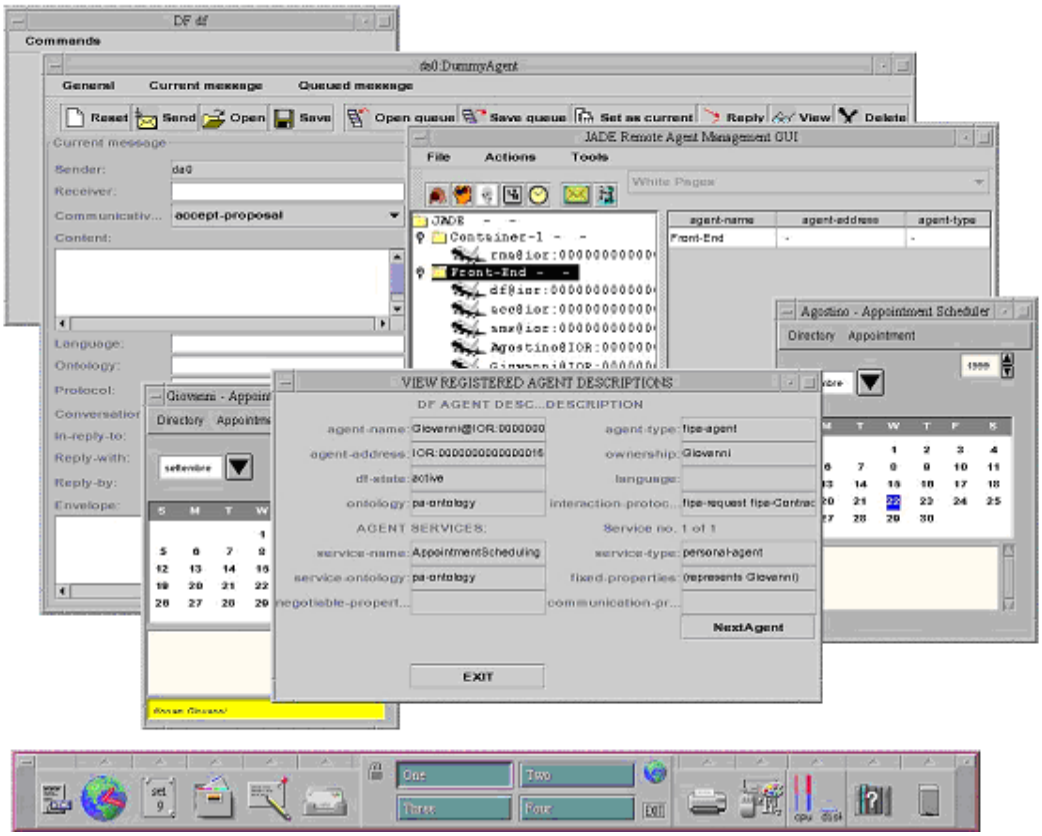


Figure 2.13. Graphical Interfaces of JADE Tools [13].

JACK is also a popular agent software development tool, which is built on top of and integrated with the Java programming language [1]. JACK Agent Language constructs can be categorised as follows: Classes (types), Declarations (#-declarations), and Reasoning Method Statements (@-statements). In addition, each of the JACK Agent Language classes supplies a number of normal Java members and methods that can be made use of in JACK programs [1].

The reasoning process design in JACK is based on the BDI model and there are five BDI event classes in the JACK agent language. Each class provides certain capabilities to handle BDI events.

2.7. Summary

This chapter reviews the history of DSS development as well as the methodologies used in current agent-based DSSs. The trends in the DSS developments indicate future DSSs ought to be intelligent, active, and open. The agent-based DSSs show high efficiency and effectiveness in tackling the various problems in the traditional DSSs, and offer high feasibility to transform a DSS to an intelligent, active, and open system.

Existing agent-based DSSs have been studied extensively. We basically classified them into three major groups including middle-agent-based DSS, decentralised agent-based DSS, and Web-based DSS utilising agents. Most of these agent-based DSSs have certain limitations. For instance, the middle-agent-based DSSs have the overload problem for middle agents; the major problem for the decentralised agent-based DSSs is manageability; the Web-agent-based DSSs are struggling with security and capabilities issues. To address these problems, we propose the AOCD framework that provides a hybrid structure for agent systems. The proposed framework is able to balance the manageability and expandability in a system and maximize the system performance.

Finally, this chapter briefly introduces some existing agent design methodologies including the agent communication and cooperation methodologies, the development of the agent society concept, and the agent software development tools. It gives a background review of the state-of-the-art in the agent design area. In the next chapter, we will introduce the basic structure of the AOCD framework and its major components.

CHAPTER 3
THE AOCD ARCHITECTURE

To tackle the various problems in most traditional DSSs that mentioned in Chapters 1 and 2, such as weak flexibility, weak extensibility, weak compatibility, etc., the AOCD framework deploys intelligent agents to improve system performances. The AOCD framework provides a range of features that support cooperation and coordination between the agent-to-agent connection and the agent-to-Matrix connection. The AOCD framework is based on a hybrid agent network topology, which combines a centralised topology with a decentralised topology through using a unique component, i.e. the *Matrix*. In the AOCD architecture, the core component (i.e. the Matrix) provides agents a living environment within which agents originate, develop, and are contained. Therefore, the term 'Matrix' is suggested and adopted in order to reflect this concept.

This chapter introduces the basic structure of the AOCD framework and illustrates the major components in the AOCD framework as well as providing a case scenario in an organisational application.

3.1. AOCD-based DSSs Vs Traditional DSSs

The AOCD is a multi-agent-based system. Unlike the other multi-agent systems, AOCD deploys a unique component, the Matrix, to assemble agents and coordinate them to solve a complex problem. The AOCD framework is also supported by a set of novel methodologies to ensure its efficient capability and functionality.

Using the Matrix-agent connection design, as Figure 3.1 shows, an external agent can be plugged into an existing system AOCD-based DSS as a component without disturbing the current system structure. In this architecture, a DSS is divided into a set of sub-systems and each sub-system will be represented by at least one agent. The Matrix is a central control panel deployed to connect the different agents. The Matrix in an AOCD is standardized and is independent of the environment that it is used in. External agents are recognised by the Matrix after they are plugged into the system. In this framework, the agents are connected directly and all the agents are also connected to the Matrix. The connections between the agents are established on request whereas

the connections between the Matrix and other components exist permanently unless an agent is removed from the Matrix.

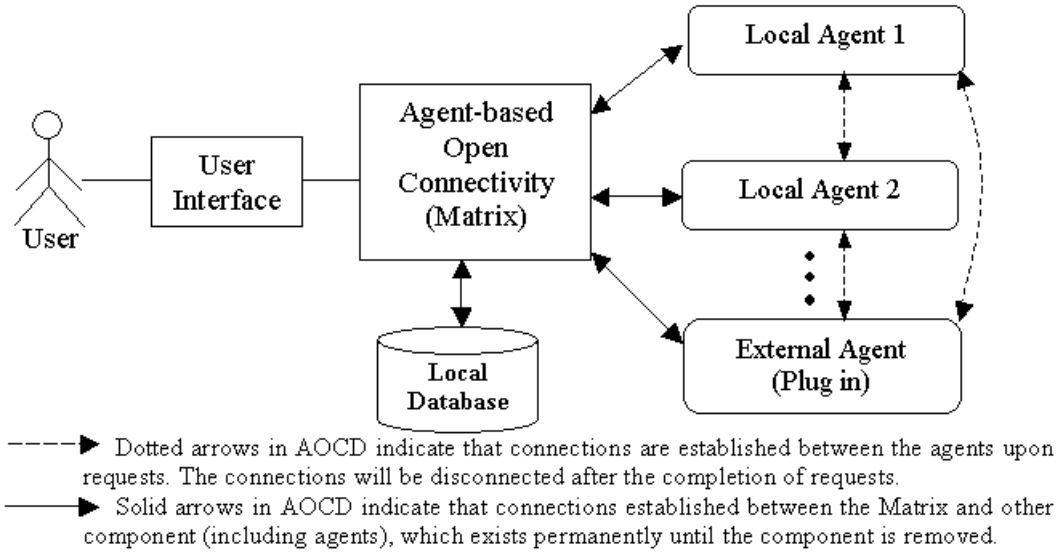


Figure 3.1. A Conceptual View of the Matrix-agent Connection Structure.

As indicated in Chapter 1, the traditional DSSs (see Figure 1.1.) have various problems that hold back the development of DSSs. The inefficient development processes of traditional DSSs discourage the concrete implementations of traditional DSSs in the real world; this problem causes the current DSS field to be disconnected from practice [9]. The AOCD framework aims to provide an open structure to DSSs through utilizing intelligent agents [167]. This novel framework adopts a hybrid structure to connect its subsystems (components).

The AOCD-based hybrid structure combines centralised and decentralised structures, which can increase the system reliability since each component is highly independent and would not be affected by other components' failures [88].

Most traditional DSSs entail high initial costs because they strictly follow traditional system development processes to ensure the system consistency. However, the AOCD framework is based on the independent component-based design. Therefore, the initial costs for the AOCD-based DSSs are much lower than traditional DSSs. Moreover, the subsystems and components in traditional DSSs have a high degree of dependency, which decreases the system expandability and flexibility.

A survey in [48] indicates that the hybrid P2P system has good consistency since it has a central control. However, it is not as good as traditional DSSs because each component in the AOCD-based DSSs is developed independently that might cause the communication gap among various components.

Based on previous studies and experimental results [144, 151, 170, 172], we draw a comparison between the AOCD-based DSS and the traditional DSS as shown in Table 3.1.

Table 3.1. Comparison Between Traditional DSS and AOCD-based DSS.

Characteristics	Traditional DSS	AOCD-based DSS
Compatibility	Poor	Good
Consistency	Good	Modest
Cost-effectiveness	Modest	Good
Expandability	Poor	Good
Flexibility	Poor	Good
Reliability	Good	Good
Reusability	Poor	Good
Performance	Modest	Good

The above table shows traditional DSSs are relatively stabile and consistent since most of the sub-components or sub-systems in these DSSs are coherent. The development life cycle of traditional DSSs is based on the traditional software development life cycle; therefore the relationships and dataflow between the sub-systems are pre-defined in the traditional DSSs. However, this design methodology weakens the other characteristics of traditional DSSs as shown in the above table.

A further comparison has been conducted to compare the performance between AOCD-based DSS and other agent-based DSS based on the literature studies [29, 32, 35, 36, 55, 74, 94, 97, 153, 174] provided in Section 2.3. The following table shows the comparison results.

Table 3.2 Comparisons between AOCD-based DSS and Other Agent-based DSS.

Characteristics	Middle-agent-based DSS	Decentralised agent-based DSS	Web-based DSS utilising agents	AOCD-based DSS
Manageability	Good	Poor	Modest	Good
Concurrency Control	Good	Poor	Modest	Good
Expandability	Modest	Good	Good	Good
Overloading Tolerance	Poor	Good	Good	Modest
System Consistency	Good	Good	Poor	Good

The above results indicate the AOCD-based DSS presents superior system efficiency. In addition, the experimental results carried out in Chapter 8 further prove the AOCD-based framework has superior performance compared with other systems.

In today's dynamic environment, how to adapt to the rapidly changing environment is the priority for most of the organisational applications. An application must be compatible, flexible, and expansible to adjust to the changing environment. These factors are becoming as important as the consistency and stability factors for modern organisations. The AOCD framework overcomes the inexpansible problem in the traditional DSSs through utilising intelligent agents, and this design improves the compatibility, expandability, flexibility, and reusability of the AOCD-based systems. The AOCD framework, a hybrid agent network, offers a superior system performance compared with solely decentralised or centralised systems. The detailed analysis of agent network topologies can be found in Chapter 7. In this chapter, we briefly review the AOCD agent network topology and illustrate the major advantages of the hybrid agent network topology.

Overall, the AOCD framework provides an efficient system structure and superior system performance for DSS applications. It overcomes the major problems in traditional DSSs including the inefficiencies in adopting external components, system upgrade, etc.

3.2. Components in the AOCD Framework

An AOCD system basically consists of four major components, which include a number of agents, a number of Matrices, several databases (or knowledge-bases) that can be accessed by the Matrices, and an AOCD network maintainer for managing the connections between the Matrices and agent transportations. Table 3.3 lists the core components deployed in the AOCD framework and the major sub-components used in each of these core components.

Table 3.3. The Core Components in the AOCD framework.

Core Components	Major Sub-Components
Matrix	Matrix register, Matrix control panel, Matrix learning centre (refer to Chapter 4).
Intelligent Agents	BDI Model, Capability register, Travel control centre, and Domain Specific Component container (refer to Chapter 5).
Databases (Knowledge-bases)	Data, rules, information, etc. (refer to Chapters 4 and 5)
Matrix Network Maintainer	Matrix distribution information centre (refer to Chapter 5).

The Matrix in the AOCD framework provides a living environment for agents. An AOCD-based Matrix normally assembles a group of agents to solve the problems for a specific organisation. In other words, a Matrix with a number of agents represents an organisational DSS application. Beyond the advantages (listed in table 3.1) of utilising a flexible and open structure in the AOCD framework, the unified Matrices structure also supports the cooperation between local agents and external (remote) agents. Figure 3.2 illustrates an example unified Matrices structure.

This unified Matrices structure allows the heterogenous agents in different domains (Matrices) to work cooperatively to solve complex problems without redundantly developing new agents for organisations. For instance, an agriculture department's AOCD-based DSS temporarily needs the population statistics of a specific region for a specific task analysis. It is costly to develop a new agent or several new agents to perform this task. Fortunately, a similar AOCD-based agent has been developed in a statistical company and this agent is also plugged into the company's AOCD-based Matrix. Therefore, the agriculture department could just send their requests to the statistical company's AOCD-based DSS for the results. This framework is cost-effective for the organisations that temporarily need outsourcings. The detailed design of the unified Matrices structure is introduced in Chapter 4.

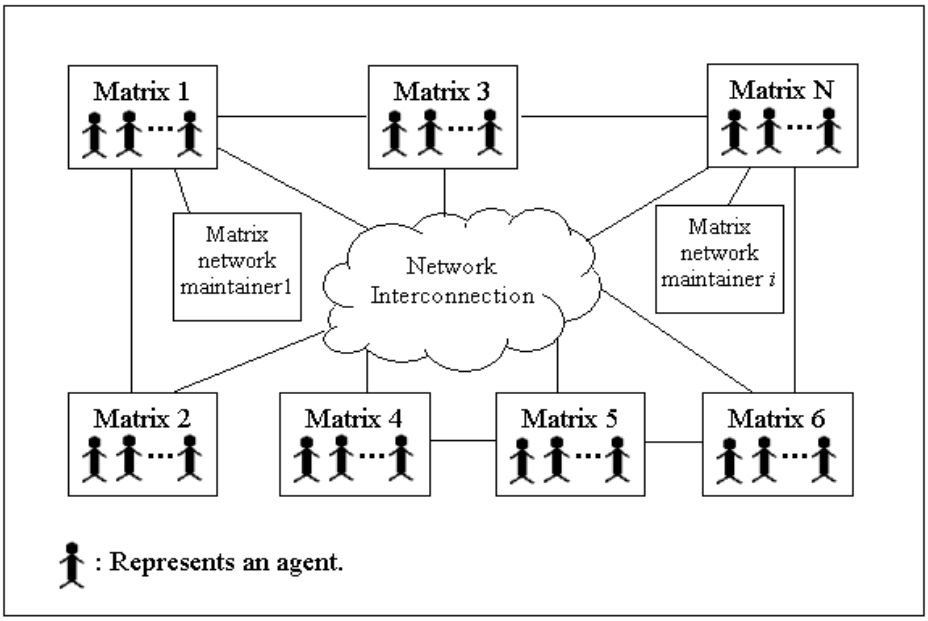


Figure 3.2. A Unified Matrices Structure.

The intelligent agents in an AOCD-based Matrix are the problem solvers, which deal with various problems and user requests within an organisation. Each agent possesses several capabilities that can solve problems and requests in a specific

domain. For instance, an AOCD-based 'Marketing agent' can perform some tasks like analysing current marketing status, obtaining the rivals' marketing strategies, and predicting market trends, and so on. The detailed AOCD-based agent design is discussed in the agent design chapter.

The AOCD databases are the major information sources in the AOCD framework. As mentioned in the agent design chapter, an AOCD agent can possess a small knowledge base for reasoning purposes. However, the knowledge base in an AOCD agent is small and incomprehensive; moreover, it cannot self-update unless connecting to a main AOCD database. An AOCD agent can update its embedded knowledge base through connecting to a major AOCD database. Figure 3.3 shows the process of updating agent 1's knowledge base through connecting with an AOCD database. Both Agent 1 and the AOCD database are connected to Matrix B.

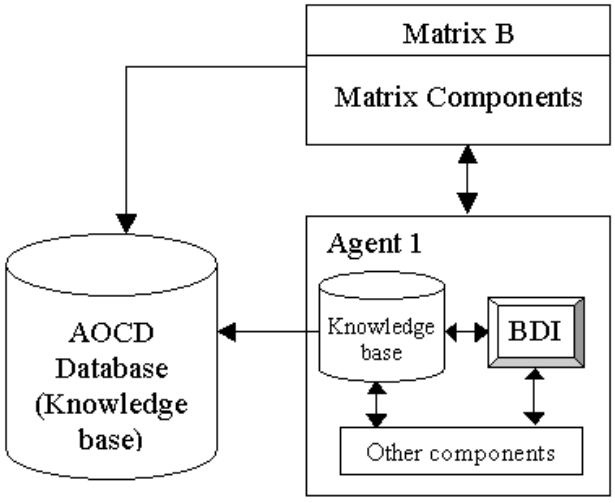


Figure 3.3. Updating an AOCD Agent's Knowledge Base.

The Matrix manages the data flow in the AOCD framework. Normally, agents receive data from their corresponding agents and the Matrix. The direct connection between an AOCD agent and an external knowledge base is allowed but not highly encouraged because the overload problem in the knowledge base. However, agents are allowed to update their internal knowledge base through connecting itself to an external knowledge base when the update involves a large volume of data or the Matrix is overloaded. To improve the security of the AOCD system, an agent must obtain an authorisation from the Matrix before access to the knowledge base.

The Matrix network maintainer is responsible for the coordination among various Matrices. As shown in Figure 3.2, the Matrices network structure is a hybrid network structure. In this structure, a Matrix is connected directly to its familiar Matrices and

is connected to any other Matrices through the Matrix network interconnection (e.g. the Internet). The Matrix network maintainer provides information about nearby Matrices to a requesting Matrix when the requesting Matrix is unable to obtain service from its directly connected Matrices. The Matrix network maintainer contains the information of the names of Matrices, the locations of Matrices, and the domain descriptions of Matrices.

3.3. The Matrix Concept

As introduced in Literature Review Chapter, existing agent-based DSS applications adopt various methodologies for system designs and implementations. Basically, the major agent-based DSSs include the middle-agent-based DSS, decentralised agent-based DSS, and Web-based DSS utilising agents. The AOCD framework is different from existing agent-based DSSs as it deploys a unique component, i.e. the Matrix, to accommodate the problem-solving agents.

An AOCD-based DSS is neither a middle-agent-based (centralised) nor a decentralised agent-based DSS; rather, it is based on a hybrid infrastructure. The Matrices are deployed in the AOCD framework to connect agents and other Matrices. This system infrastructure is proven [170, 172, 173] to be efficient as it balances the expandability and manageability in a system. In the next section, we will briefly discuss the advantages of using the hybrid network structure in the AOCD framework.

The Matrix concept in the AOCD framework suggests a new method for agent cooperation and coordination, which incorporates and extends the agent society concept (see Section 2.4) into the Matrix design. Unlike the middle-agent-based DSS, the Matrix not only acts as a middle-agent or an agent facilitator but also as a living platform for agents. The Matrix provides a small community environment for agents, where agents can self-upgrade and cooperate with other agents. The unified Matrices structure provides a larger and more comprehensive structure that allows the cooperation among various Matrices. This unified Matrices structure extends the agent society concept that heterogeneous agents are distributed in various agent groups (communities) and each group is dominated by a Matrix. These groups, i.e. Matrices, form a large scale of interconnected Matrix network as shown in Figure 3.2.

Unlike the decentralised agent-based DSS, the unified Matrices structure is partially decentralised; each Matrix directly connects to its familiar partners and

connects to other Matrices through the coordination of the Matrix network maintainers. This design provides efficient controls on the cooperation among Matrices and the cooperation between agents inside a Matrix is also highly manageable.

3.4. The AOCD Framework: A hybrid network

The AOCD framework adopts the hybrid topology for the Matrix-agent connections and Matrix-Matrix connections. Among various agent network topologies, we basically summarise the three most common network topologies that are used in distributed networks [55, 105] including: the centralized topology, decentralized topology, and hybrid topology. The detailed analysis of the hybrid topology for agent network will be discussed in Chapter 7 and the performance evaluation also will be provided in Chapter 8. In this section, we will briefly review the major advantages of applying the hybrid network topology in the AOCD framework.

The major characteristics of multi-agent systems are described in [79] as:

- Regular agent cooperation and coordination. To solve a problem, an agent usually needs to have cooperation with other agents. In other words, agent cooperation and coordination in multi-agent systems happen regularly as each agent has limited capability. Therefore, a solely centralised framework will be very inefficient in such a system;
- There is no global control (however central controls in local areas are allowed); Data and information are normally decentralised. A solely decentralised framework is difficult for management, thus central controls in local areas are helpful to improve the manageability in the decentralised framework.

The hybrid topology is highly regarded as an appropriate solution to multi-agent systems as it tallies with the characteristics of multi-agent systems. The hybrid topology provides the most efficient framework for agent communication in the AOCD framework. Based on the previous analysis of the hybrid topology and experimental results on AOCD systems [170, 172], we can summarise the reasons of the adaptation of the hybrid topology in the AOCD framework as follows:

- Efficient processing time and memory consumption (cost-effective),
- Superior manageability,
- Efficient flexibility and extensibility,

- Stability in requests transmission.

The hybrid topology in AOCD is different from the homonymy topology presented in distributed systems [105]. In a distributed hybrid system, agents are partially connected with the other agents. On the contrary, Figure 3.1 shows that each agent in AOCD is able to connect to any other agents directly. However, it is not always necessary to keep all the connections. A dotted line in Figure 3.1 reflects a connection between agents and the connection will be established only when it is required. The Matrix's central control feature improves manageability. In the decentralized topology, the concurrent control and synchronization issues often reduce its manageability. The Matrix's unique design in AOCD maximizes the extensibility of the system.

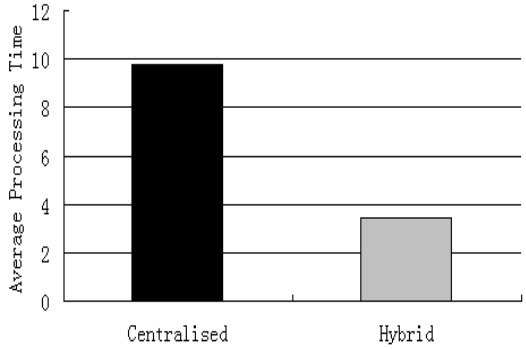


Figure 3.4. Average Processing Time Comparison.

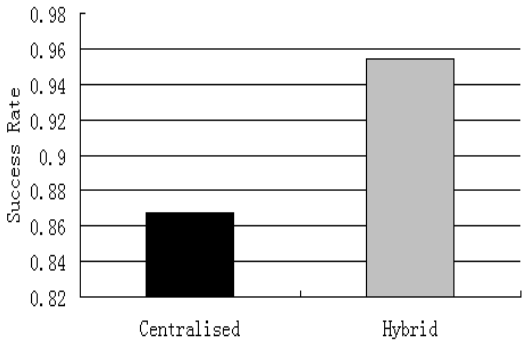


Figure 3.5. Success Rate Comparison.

From the system performance point of view, the hybrid framework offers considerable advantages. Compared with the centralized framework, the hybrid topology reduces the transmission time and relieves the overload problem in the central component. The experimental results introduced in Chapter 8 provide the evidence of the efficiency of the hybrid framework. The experimental study shows that the total time consumption for processing 2917 requests is 28477 seconds in the centralised topological experiments. The average time consumption for one request in the centralised system is 9.76 seconds, whereas the average time is only 3.41 seconds in the hybrid topology (Figure 3.4). In addition, the average success rates of the two topologies are 86.74% (centralised) and 95.45% (hybrid), respectively (Figure 3.5). The success rate of a matching process is calculated by the volume of processed requests (PRV) divided by the total requests generated by the requesting agents (TR). TR also includes the unprocessed requests because of deadlocks. Logically, the more agents participating in the process and the lesser request volume help to increase the

success rates. The experimental design and more detailed experimental analysis can be found in [173], which are currently based on the AOCD framework. The future work will consider generalising the analysis to other agent-based systems.

Generalizing from the above factors, the hybrid topology stands out as the most efficient topology for the AOCD architecture. The hybrid topology in the AOCD framework enables an agent to connect with all the other agents. The connection between two agents is only established when there is a request and will be disconnected after the request is accomplished.

3.5. A Case Study: Agent Technology – A Solution to Modern DSS

Most organisational managers often find it is very difficult to decide whether their organisational information systems need an upgrade or not [3]. There are many factors that plague decision makers about whether or not to upgrade their systems, which, of course, are traditional SDLC-based systems.

The first factor is the enormous time consumption for the development of a traditional information system. For traditional SDLC-based systems, the upgrade period not only includes the time for designing specific components but also includes the time for re-evaluating user requirements, modifying existing system structure (possible re-design when upgrade components affect the overall system processes), integrating a new system to maintain system consistency, and extra outsourcing manpower is often used for communication in every upgrade phase. Meanwhile, the training processes for staff to understand new systems also take time.

The second factor is the high possibility of upgrade failure. As we mentioned in Introduction Chapter, the success rate of developing IT-based systems for business process is disappointing. Managers often face the risk of system failure when developing or upgrading a new system.

Thirdly, the dynamic environment might render their upgrade useless. People are living in a so-called information explosion age, which makes our day-to-day environment very difficult to predict. New technologies, new business opportunities, and new inventions are emerging everyday. In this circumstance, any system upgrade cannot last for long. Therefore, the constant upgrade processes confuse organisational mangers. Certainly, there are many other reasons for the problems of upgrading or re-designing, such as the technology gap between existing and new systems, system

structural inefficiency that results in some unknown information loss between the new and existing system, and human errors that might occur when constantly shifting their familiar interfaces to some unfamiliar interfaces, etc.

The use of multi-agent technology is able to solve the above problems. AOCD is a multi-agent-based architecture, which is open and flexible. To illustrate the effectiveness and efficiency of the architecture we use the following case as an example.

Nowadays, mergers of international corporations happen frequently as the globalisation phenomenon has extended to today's economic section. Hidden behind the encouraging benefits of mergers, most merging corporations have encountered difficulties in integrating various information systems from different mergees. In many traditional information system development frameworks, the process of combining two different information systems, especially DSSs, from two different organizations into one system is time-consuming, costly and inefficient. The traditional development process may involve re-analysing the user requirements, reconstructing the database, redeveloping the code, etc. For instance, during the merger process of the German automaker Daimler Benz with the U.S. automaker Chrysler [3, 17], difficulties were found when integrating Daimler's CAD systems with Chrysler's CAD systems because the two companies' CAD systems were designed based their own company's business and manufacturing processes and their CAD systems were incompatible with each other. Many similar cases can be found when merging two corporations. Using the AOCD architecture can easily solve the integration obstacles that Chrysler and Daimler faced. To simplify the case, we use two fictitious institutes, Institute A and Institute B, to explain the merging process in AOCD-based systems.

Institute A and Institute B are educational institutes. Both institutes have developed two different library systems based on the AOCD architecture. In Institute A, there are five major intelligent agents, which include: human resource (HR) agent, loan and return agent, book sales agent, book circulation agent, and customer info agent (including a small customer database). Each agent is responsible for the information process in its business domain. The HR agent manages current staff information and offers advice to the manager if it is necessary to reduce the labour of the institute. The loan and return agent provides information such as outgoing book information, returned book info, on-hold book information, etc. The book sales agent provides the

information about the best seller, sales statistics, and the market trends, and so on. The customer info agent mainly deals with customer details, customer credibility, current loan status, etc. The book circulation agent is responsible for book stocking, managing new books, managing washout books, etc.

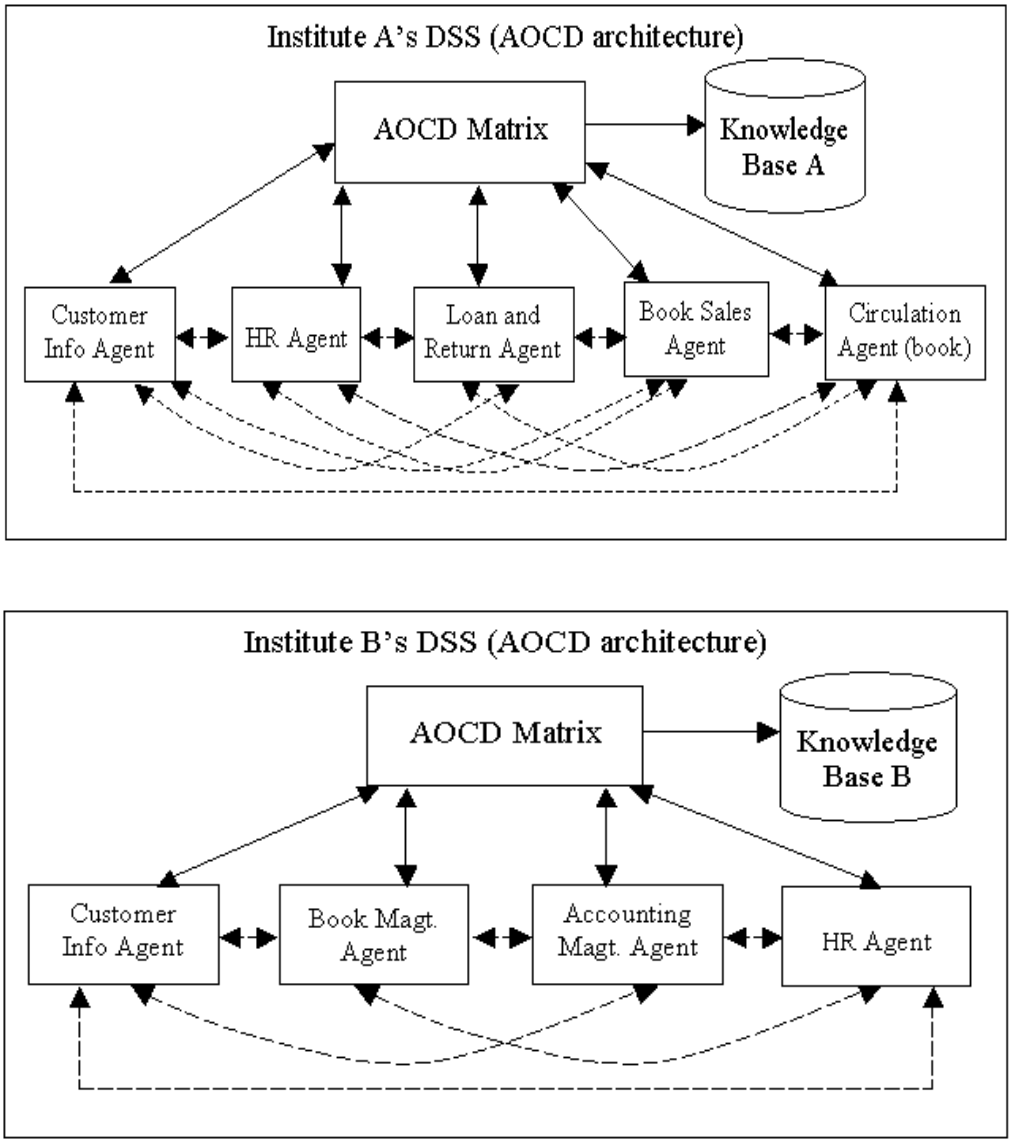


Figure 3.6. Two Institutes' AOCD-based Information Systems (before merger).

Institute B's information system is different from Institute A's as its business strength is financial statistics design. There are four major intelligent agents in Institute B's system including: HR agent, customer info agent, book management agent (including loan and return management, book sales, and book circulation), and accounting management agent. The HR and customer info agents of Institute B are similar to Institute A's. The accounting management agent deals with all the accounting issues, including sales statistics, staff salary, reimbursement, amercement

management, income payments, and so on. The book management agent provides all the functions related to book management issues.

As the strategic alignment would benefit both institutes, the executive boards of the two companies decide to merge the two institutes. There would be many problems to merge the institutes' information systems if they were using traditional information architectures because the headquarters of the two companies are located in two different suburbs and their customer info database systems embedded in customer info agent are incompatible. Now, most of the difficulties can be solved within the AOCD framework. Figure 3.6 shows the two institutes' information systems before the merger. There are some similarities in the two institutes' information systems, the two human resource agents are very similar and Institute B's financial agent can deal with all the financial issues in Institute A's framework.

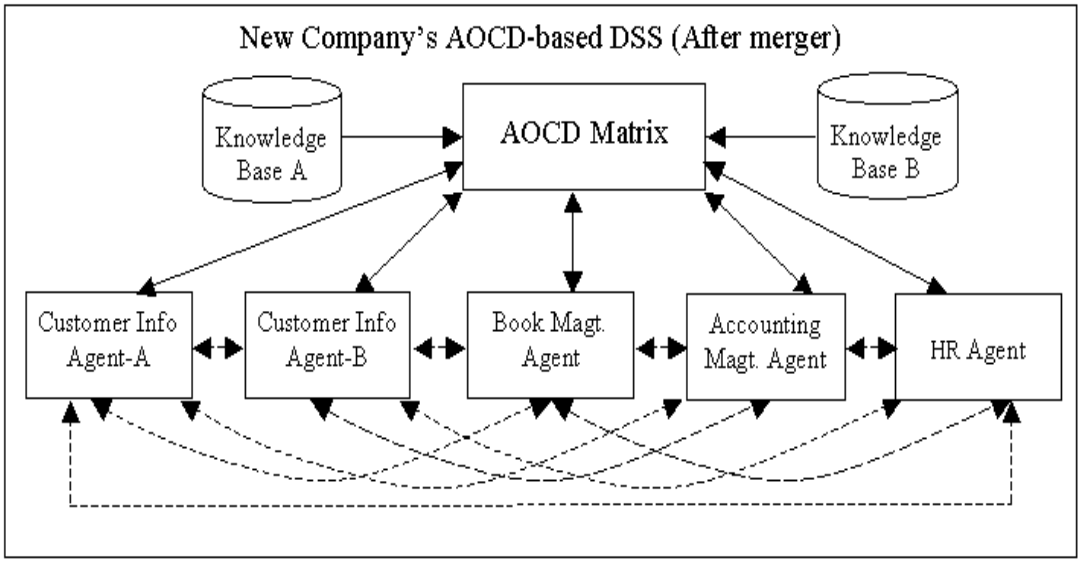


Figure 3.7. New Institute's Information System (after merger).

The Matrix in the AOCD framework is standardized; therefore, the integration process can just use one Matrix. Because the HR agents in two institutes' information systems are very similar, the integration process can use one HR agent instead of using both of them. We also presume that Institute B's book management agent can replace the loan and return agent, book sales agent and book circulation agent of Institute A because the book management agent is able to deal with all the book management issues in Institute A. The integration process is simple and easy because the AOCD design supports plug-and-play. The integrated information system after merger (see Figure 3.7) can support both institutes' business processes. The new

institute's information system retains two customer info agents because the former two institutes' customer info databases were incompatible. The new information system also retains two knowledge bases, which are located at the two headquarters, respectively. The Matrix would be located at one of the headquarters and the agents would be connected to the Matrix through the local area network, the wide area network, and the Internet. Regarding the integration process in practice, there are several options to integrate agents to Matrices. The most common methods include: (1) Build agents as executable programs; and call them through the Matrix user interfaces. (2) Build agents as ActiveX or dynamic link library files (plus a separate knowledge base); and call them from inside the Matrix functions. As mentioned in Chapters 1 and 2, the Matrix design is standardised. In other words, the Matrices provide the same environment for agents. Therefore, the above merge process only takes the agent integration process into consideration. The link between the Matrices maintained by the Matrix network maintenance components could still function or some Matrices can be deported after the agent merger.

As we can see from this case, the difficulties in the integration process that plague many traditional DSS applications are solved in the AOCD framework. In this new framework, there are no complicated re-design procedures involved in the integration or upgrade process and the risk of upgrading failure is minimized. In the AOCD framework, agents can update themselves through updating their *Domain Specific Components (DSC)* (see Sections 4.1.4, 4.2.3, and 5.2.2). The dated DSC items are unplugged and replaced by the new DSC items from the DSC warehouse. In this way, an entire AOCD system could be updated through updating agents. The Matrix register also performs a regular and conventional updating process. The Matrix register will update itself once a new agent establishes the connection with the Matrix or an agent updates its DSC items. The updating process becomes much easier in the AOCD framework than in solely distributed systems since AOCD systems adopt the hybrid structure.

The advantages of using the AOCD framework as a new DSS framework can be summarised as that it has: (*i*) efficient and cost-effective system development life cycle, (*ii*) robust and reliable system performances, (*iii*) scalable and flexible system structures, (*iv*) efficient reusability.

3.6. Summary

The AOCD framework offers a range of new features to accommodate an efficient and effective environment for organisational DSS. This chapter briefly introduces the basic structure of the AOCD framework and its major components.

The major new features in the AOCD framework include the hybrid network structure, Matrix design, Matrix network maintainer, and agent self-upgrade mechanism. The reasons for adopting the AOCD framework as an open DSS structure are various. In brief, we summarise the main points as follows:

1) The AOCD framework achieves the goal of providing an open structure for DSS through utilising intelligent agents. This framework employs the 'plug-and-play' design structure, in which an external agent can be efficiently incorporated into a system through connecting to a specific Matrix.
2) The AOCD framework achieves the goal of balancing expandability and manageability in a system through adopting the hybrid agent network topology. The hybrid network topology is not only adopted in the Matrix-agent connection design but also applied to the unified Matrices structure. The hybrid network topology combines the centralised network connections and the partially connected decentralised network connections in a system, which improves the manageability through the central controls and enhances the flexibility through partial or full direct connections.
3) The AOCD-based agents are able to self-upgrade through connecting their internal DSC containers to the DSC usage centre (see Chapter 4, Section 4.1.4). This design simplifies the complex agent learning processes since the upgrading of an agent's knowledge is based on a relatively small size (compared with the knowledge-base in the Matrix) and specifically focuses on some certain domains.

Overall, the proposed AOCD framework has been proved to be efficient. The experimental results prove that the AOCD-based framework offers efficient system performance. The detailed design methodologies and experimental results will be discussed respectively in Chapters 4, 5, and 8.

CHAPTER 4
MATRIX DESIGN AND THE UNIFIED MATRICES STRUCTURE

The Matrix is a core component in the AOCD framework. The Matrix concept has evolved from the human society structure, which not only concerns individual entities in a system but also provides a living environment for all the entities (agents). The Matrix incorporates three core concepts, which include the agent living environment, the agent life cycle, and the agent relationships. This allows the Matrix to provide information resources to agents and support the agent life cycle. In the Matrix-agent framework, agents are integrated into a Matrix and each agent solves problems in its specific domain. The Matrix distinguishes itself from the agent facilitators and mediators that mainly deal with agent coordination; rather the Matrix incorporates agents in its virtual space in which agents can obtain information, perform self-upgrade, and be reused. A Matrix is normally set up in a computer system as a software application. The Matrix provides an interface for system users, which allows them to configure and manage agents and the Matrix register. The Matrix network maintainer is also a software application, which normally set up in a super-node computer. It allows system managers to configure the unified Matrix structure.

In the AOCD framework, the cooperation between agents is based on a unified Matrices structure. This structure connects numerous Matrices through heterogenous networks such as the Internet, private networks, etc; it also deploys a number of Matrix network maintainers to manage the information about the Matrices. The process of searching a corresponding agent incorporates the keywords matching methodologies in the Matrix network maintainers with the process of applying the Agent Rank algorithm in related Matrices. The Matrix deploys a novel design, i.e. the *Domain Specific Component* (DSC) design, for constructing agent capabilities (see Chapter 5, Section 5.2.2). This design allows agents in the AOCD framework become more flexible and upgradeable.

This chapter is organised as follows. The first section in this chapter introduces the Matrix infrastructure and its core components, including the Matrix control panel, the Matrix register, and the Matrix learning centre. Section 4.2 introduces the

mechanisms used in the unified Matrices framework, which include the Matrices searching mechanism, Matrix network maintainers, and the cooperation process among various Matrices. Section 4.3 briefly explores the Matrix development life cycle and the last section concludes the Matrix design methodologies.

4.1. Matrix Structural Design

Unlike the decentralised agent-based DSS, the unified Matrices framework is partially decentralised; each Matrix can directly connect to other Matrices through the coordination of the Matrix network maintainer.

Based on the three core concepts of the Matrix, we illustrate four fundamental components in the Matrix, which include: (*i*) the *agent society* that provides a virtual space to agents (see Chapter 2, Section 2.4); (*ii*) the *Matrix learning centre* that acquires information (new DSC items) from the external environment; (*iii*) the *Matrix control panel* which is the core part that mainly handles agent matching, requests processing, and resource allocation; and (*iv*) the *DSC Usage Centre* that allows the functional components to be used/reused by agents. Each AOCD-based agent carries a DSC container, which holds a number of DSC items. Each DSC item can perform one (or several) specific task(s) and produce results; and each item can be plugged/ unplugged into/from the DSC container, which offers a flexible structure for upgrading agents' capabilities. These DSC items in an agent represent the agent's problem-solving capabilities. We will introduce the detailed design of the DSC container in Chapter 5. Figure 4.1 shows a conceptual view of the Matrix.

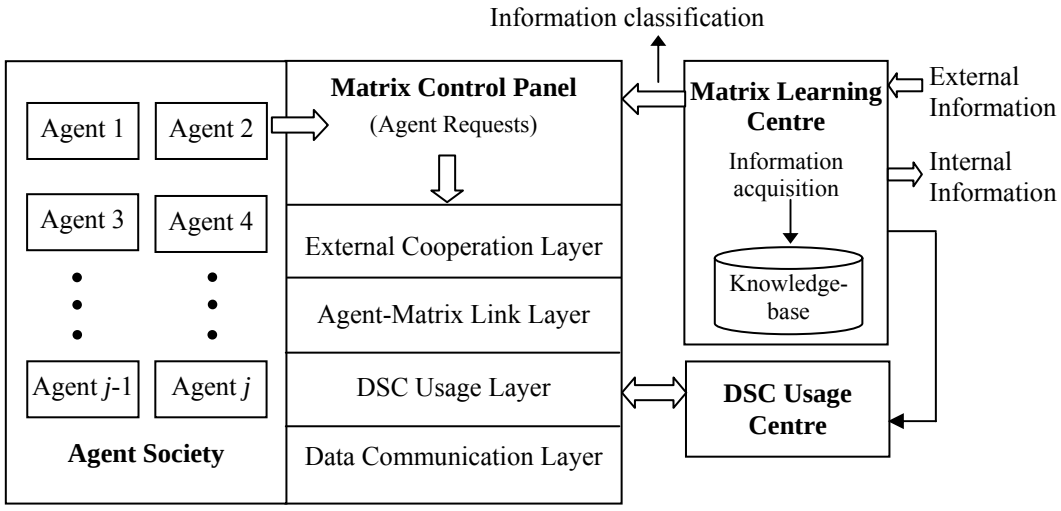


Figure 4.1 Conceptual View of Matrix.

The *Matrix control panel* manages the agent society through establishing relationships with agents. An agent society can be regarded as the aggregation of a set of agent-relationships. Managing relationships between agents is a key issue in this architecture. Other issues, such as actual physical space (or memory consumption), geographic distance, and dimensions of the agent society are not the major concerns here. In an agent society, each agent experiences difference stages in their life cycle, which is introduced in Chapter 5.

The *Matrix learning centre* section of the Matrix basically deals with knowledge acquisition and information classification. This is the major method whereby the Matrix updates its information resources and communicates with the external environment.

The Matrix-Agent connection design is based on a hybrid network topology. The connections between agents and Matrix are centralised, whereas the connections between agents are decentralised. The agent-agent connections are established upon requests and will be disconnected after the completion of requests, whereas, the Matrix-Agent connections exist permanently until agents are removed. The major role of the Matrix is to organise the agents to accomplish the tasks sent by users. The central role managed by the Matrix in the AOCD working process enhances the system's manageability by eliminating concurrent control and synchronous communication problems.

The linkages of a Matrix and agents are through the connection interfaces. A connection interface will be allocated to an agent when the Matrix receives a request of connection from the agent. An agent can connect to the Matrix at any time without disturbing the current system.

The structure of a Matrix consists of four layers: External cooperation layer, Agent-Matrix link layer, DSC usage layer, and Data communication layer. Each layer has a specific role in the Matrix design. The working process of a Matrix is based on the coordination of these four layers. The next section further explains the design of each layer in the Matrix.

4.1.1. Four-Layer Structure of the Matrix

Basically, the Matrix consists of four layers as shown in Figure 4.2.

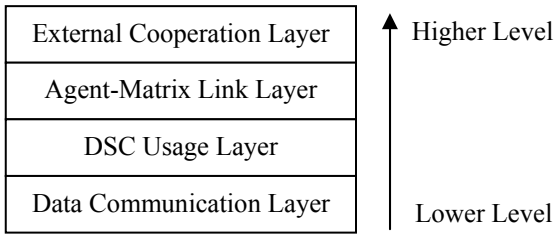


Figure 4.2. Four Layers in the Matrix Design.

The role of each layer in the Matrix is as follows:

- External cooperation layer: This layer allows a Matrix to connect to external entities such as other Matrices. This layer provides system users with user interfaces in order to load external agents in the other Matrices to the Matrix and establish configurations; it also connects the Matrix to its corresponding Matrix. This is the highest level of the Matrix design, which is mainly based on the cooperation between a Matrix and the external environment. Figure 4.3 shows a graphic user interface (GUI) design for the external cooperation layer. The type of external entities can be an external Matrix or an external agent. The external entity address can be an IP address, or a unique semantic address. No matter what the address format that system users input to the address box, the address will be eventually converted to a recognisable and unique address that is stored in a Matrix network maintainer.

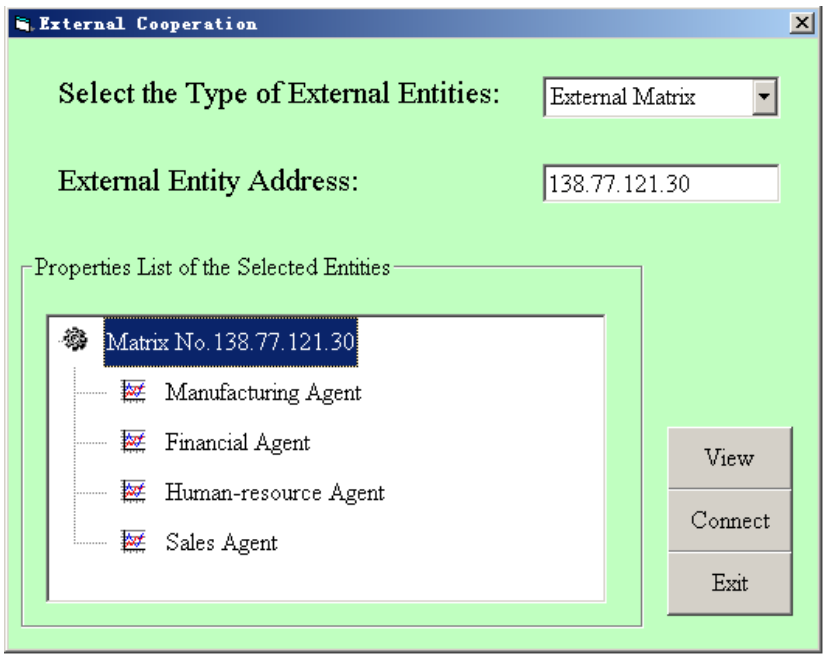


Figure 4.3. GUI Design for the External Cooperation Layer.

- Agent-Matrix Link Layer: This layer enables an internal agent to establish connections with its Matrix to solve a problem. This layer is the second highest level in the Matrix design, which mainly deals with the connections between the Matrix and agents. Before an agent is linked, we need to load this agent into the system through the Matrix. Figure 4.4 gives a snapshot of the GUI design for loading agents to the Matrix.

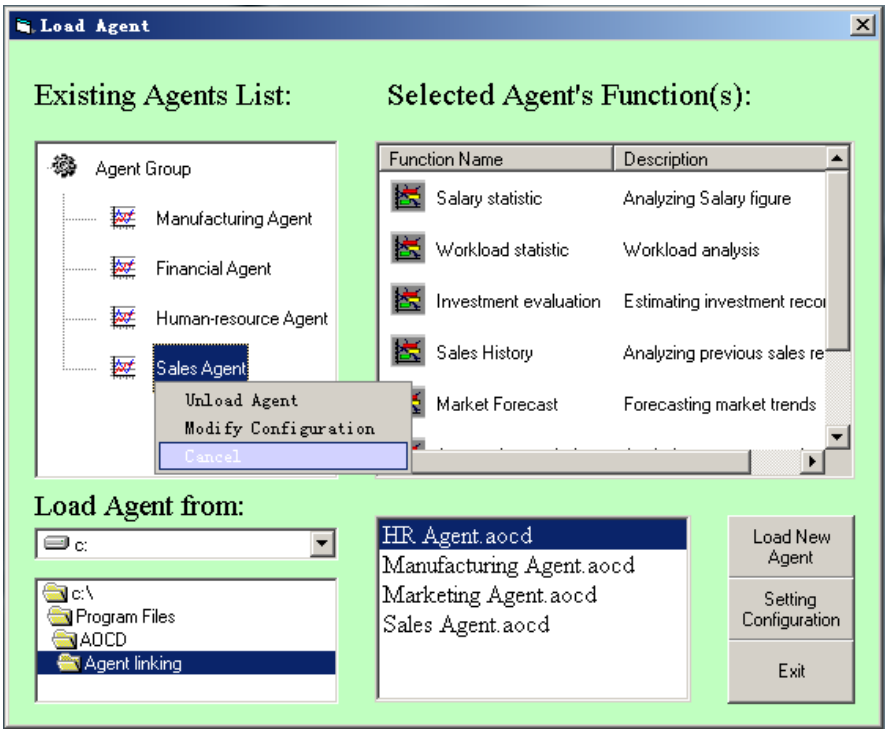


Figure 4.4. GUI Design for the Agent-Matrix Link Layer.

The 'Existing Agents List' shows all the agents that have been plugged into the Matrix. Users can unload agents in this list and modify the agent configurations. The 'Selected Agent's Function(s)' area shows the functions of the highlighted agent in the left 'Existing Agents List'. This interface provides users with the function to load an agent from the local site through the directory path. To load remote agents, system users need to configure the external cooperation layer, which is triggered by 'Setting Configuration' button.

- DSC Usage Layer: This layer provides domain specific components for agents. The DSC items are the elements of agent functionalities. In other words, an agent's capabilities are based on the DSC items that it carries. The detailed description of DSC can be found in Chapter 5. This layer mainly focuses on the usage of agent functional components, which is the foundation of the agent link layer.

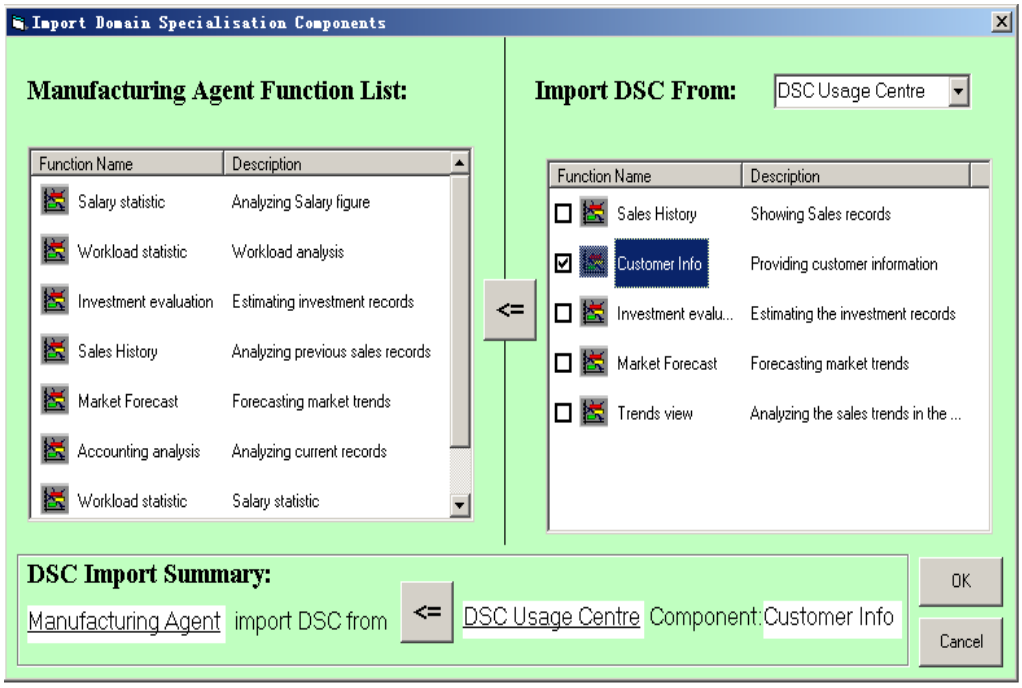


Figure 4.5. GUI Design for the DSC Usage Layer.

There are two methods to import a DSC item to an agent. The first method is the manual import procedure that imports a DSC item through the interface. The second method is the automatic import procedure, which achieved by the agent's self-learning process and the DSC usage centre (see section 4.1.4). This section mainly introduces the manual import method, as shown in Figure 4.5. A DSC item, 'Customer Info', is imported to the manufacturing agent from the DSC usage centre. A DSC item can be imported from three major sources, including the DSC usage centre, an internal agent, and an external entity. The list-box on the left shows all the functions of the manufacturing agent. The 'DSC Import Summary' describes the import result to ensure it is accurate.

- Data Communication Layer: This is the lowest level in the AOCD matrix architecture. This layer provides data processing, data structure formatting, and data storage functions. All the data transmission and computation are based on this level. The interface for the Data communication layer provides the user with a standardized structure of data and enables an agent to cooperate and communicate with the Matrix and other agents.

These four layers are managed by a core component in the Matrix, namely the *Matrix Control Panel*. The next section introduces the detailed design of the Matrix control panel.

4.1.2. Matrix Control Panel

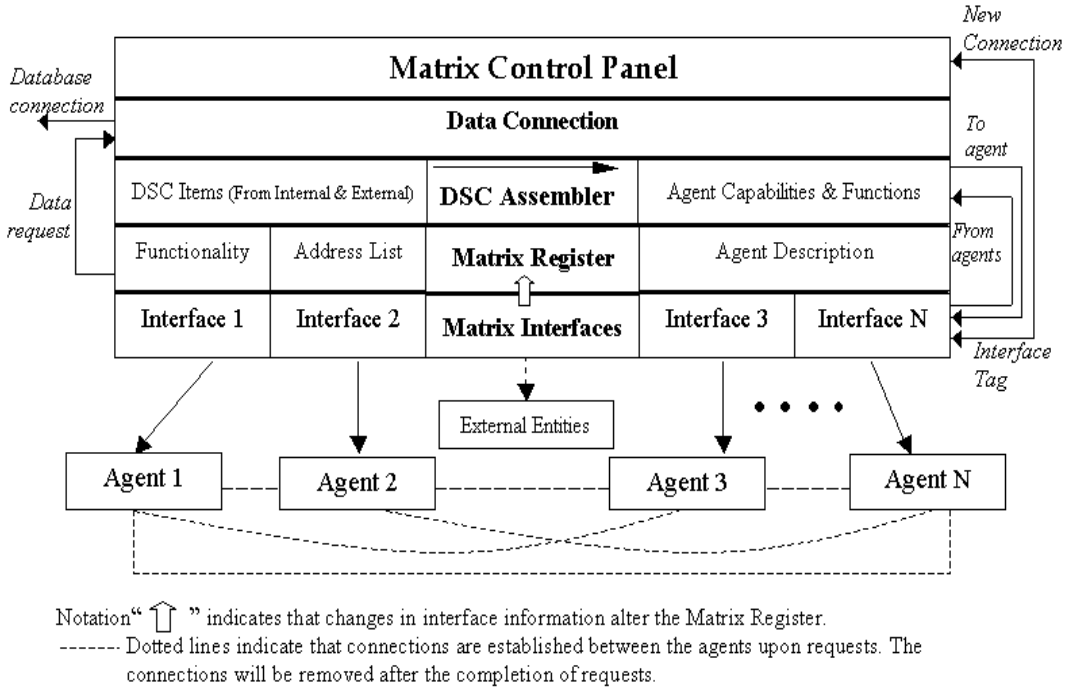


Figure 4.6. Matrix Control Panel Design.

The features of the Matrix control panel have been described in our previous work [173]. The Matrix control panel allows system users to control and manage the activities of the Matrix and agents as shown in Figure 4.6.

The components in the different Matrix layers cooperate with each other to deal with agent requests that are received by the Matrix interfaces. Each agent in an AOCD system is required to register its information to the Matrix register.

The Matrix control panel operation process starts with loading the previous configuration parameters, which contain the information of previous allocated connection interfaces such as the number of connection interfaces allocated in the last system operation. Figure 4.7 shows the operation process of the Matrix control panel. When the connection interfaces have been established, the Matrix control panel begins to receive requests from the agents. There are three types of requests, including the requests for connection interfaces, the requests for corresponding agents, and the requests for DSC items.

Once a request for connection interface is received, the Matrix control panel will change the Matrix register and allocate a new connection interface for the new agent. In the meantime, a scanning process is also performed to ensure the accuracy of the

system. In normal circumstances, the allocation of a new interface is triggered by an agent request. However, there are some extreme situations that may cause error in allocating the connection interface resource. For instance, if requests from the agents occur concurrently or if the system is shut off inappropriately, then the updating Matrix register information process could be interrupted. The failures in the updating process could lead to errors of redundancy or pretermission in reading interface numbers. Therefore, a frequent scan method is deployed in the 'Receiving data from agents' process, which could prevent the system from losing the information for changing the connection status.

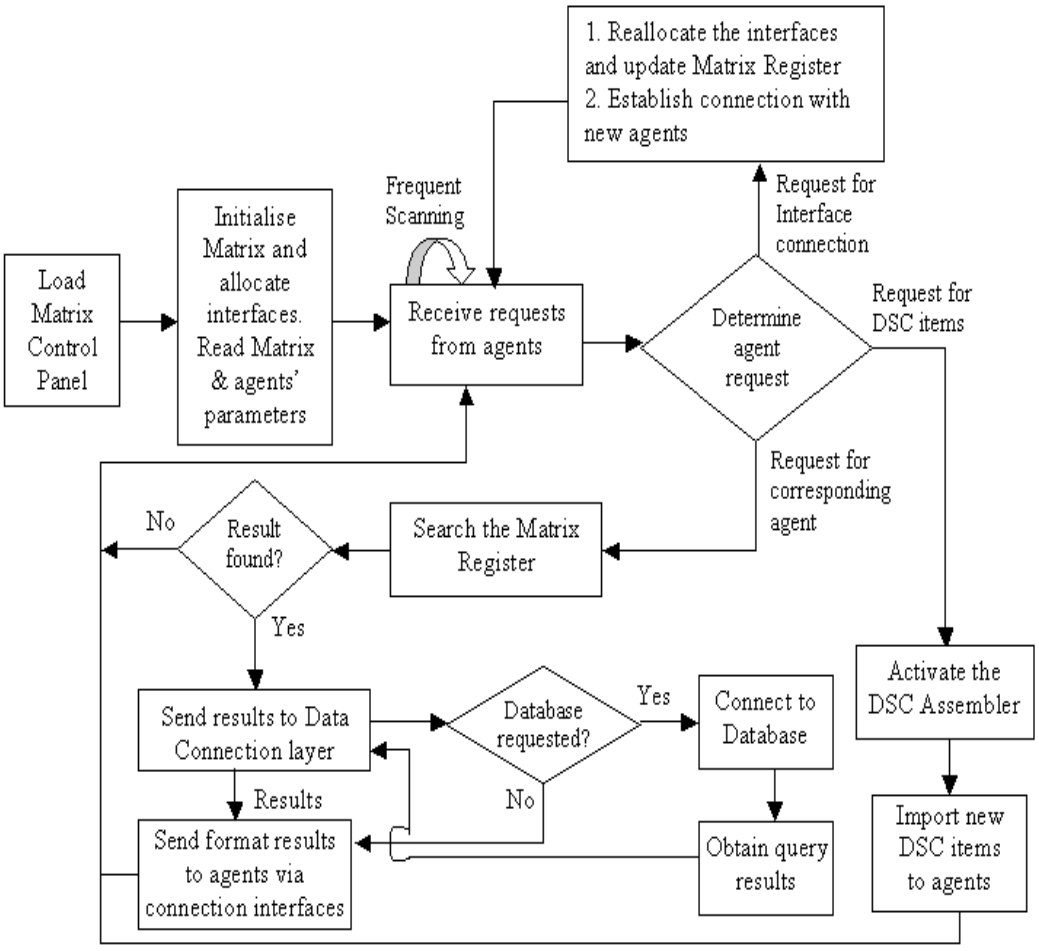


Figure 4.7. Operation Process of the Matrix Control Panel.

If a request for corresponding agents is received, the Matrix control panel forwards the request to the Matrix register for searching a corresponding agent (or agents). If a corresponding agent is found, the result will be delivered to the data communication layer to convert the request data into a set of recognizable data formats. If the requesting agent further requires information from the database before it sends the

request to the corresponding agents, then the Matrix control panel needs to perform the database operation according to the format requests. The database sends the results to the Matrix control panel; and the results are delivered to the corresponding agents via the connection interface.

If a request for updating DSC items is received, the Matrix control panel sends the request to the DSC assembler. The DSC assembler is able to search a DSC item from the DSC usage centre or external entities. If an appropriate item is found then the DSC assembler will import the item to the requesting agent and update the agent's capabilities.

The Matrix control panel is also responsible for managing the agent society inside an individual Matrix system. The concept of an agent society has been introduced in [168]. The agent society provides an agent with DSC items for self-upgrade and agent links for agent cooperation. The Matrix control panel manages the agent society through establishing the Agent-Matrix links. An agent society can be regarded as the aggregation of a set of agent-Matrix relationships. Managing relationships among agents is a key component, i.e. the Matrix register, which provides the Matrix control panel with the registered agents' information for agent relationship management.

4.1.3. Matrix Register

The Matrix register is deployed in the AOCD framework to enhance the efficiency of agent relationship management and agent cooperation and coordination in the Matrix. The Matrix register records the information about all the agents that have been plugged into the Matrix. It plays a primary role in the Agent-Matrix link and DSC usage layer. The major functionalities of the Matrix register include: *(i)* searching corresponding agent based on the descriptions and *(ii)* providing the corresponding agent address to the Matrix control panel.

The Matrix register makes use of a three-column list to store the information about agents, which contains agents' capabilities and data structures. The first column contains agent functionalities, which indicate the agent's domain or expertise; the second column contains agent identity, which indicates an agent's name and IP address; and the third column contains the standardized descriptions of an agent's capabilities in an abstract level. A data attribute can be viewed as an attribute of a table in a relational database.

The Detailed Description column in the Matrix register is standardized, which provides the information for searching a corresponding agent where the agent linking process is automatic.

Table 4.1. An Example of the Matrix Register.

Functionality	Agent Information	Detailed Description
Coffee Sales Statistics	Agent 1 & (220.168.77.10)	A1-Macro: Sales-coffee-daily
Tea Sales Statistics	Agent 1 & (220.168.77.11)	A1- Macro: Sales- tea-daily
Coffee Stock Management	Agent 1 & (220.168.77.18)	A1- Macro: Stock-coffee-daily
Tea Stock Management	Agent 1 & (220.168.77.20)	A1- Macro: Stock-tea-daily
Current Market Analysis	Agent 1 & (220.168.77.25)	A1- Macro: Market-daily
Staff Info Management	Agent 2 & (220.168.77.30)	A2-Armstrong: Staff-Info
Staff Salary Management	Agent 2 & (220.168.77.35)	A2- Armstrong: Staff-salary
Accounting management	Agent 3 & (220.168.77.40)	A3- Armstrong: Accounting-monthly

In Table 4.1, the functionality column contains the information on the agent's functionalities, which provides agent capability information for the agent matchmaking process. The agent information column contains the information on the agent's name and IP address. The Detailed Description column consists of four major items: organization/department, data table name, data attribute, and description control (if applicable). For instance, in A1-Macro: Stock-coffee-daily. A1 is the agent name, Macro is the department, Stock is the table name, coffee is the data attribute, and daily is the description control. Table 4.1 shows an example of a Matrix register.

The maintenance of the Matrix register relies on the information provided by the Agent-Matrix link layer. As shown in Figure 4.6, the changes of interface information in the Agent-Matrix link layer alter the information stored in the Matrix register. The procedures of updating the Matrix register information can be briefly described as follows: once a new interface is allocated to a new agent, the Agent-Matrix link layer will send a message to the Matrix Register in order to update the current information. After being advised by the Agent-Matrix link layer, the Matrix Register will be updated through inserting the information about the new agent.

4.1.4. The DSC Usage Centre and the Matrix Learning Centre

The Matrix learning centre provides information resources to agents. This component distinguishes the Matrix from existing middle agents or agent facilitators that mainly play as the role of an agent coordinator. The Matrix learning centre

upgrades the agents' functionalities through acquiring information from the external environment and sending the acquired information in DSC formats to the DSC usage centre. The DSC usage centre manages the unused and active DSC items for agents to update their capabilities. Each DSC item is executable and has predefined inputs and outputs.

Input and output of the DSC items

The input and output information of each DSC item contain information as shown in Figure 4.8.

Input Section: [Input 1: String (Compulsory); Input 2: Integer; Input 3: Float; Input 4: Float (Compulsory).]
Output Section: [Output 1: String; Output 2: Float.]

Figure 4.8. Inputs and Outputs in a DSC item.

In the input section, the information includes the input content, data type, and constraints. The input content indicates the value of the inputs such as a string or a number. The data type indicates the input content's data type, which include string, integer, float, double, etc. The constraint indicates whether the input content can be null. If the constraint value is 'compulsory' then the input content must not be null, otherwise the input content can be null. The output section consists of the output content and data type; they basically have the same meaning as the input section. The validation of the meaning of an input has been pre-processed in the Matrix searching and agent matching process. Therefore, the DSC items inputs generally align to the pre-defined input constraints. Even if an input has a valid structure but is meaningless, it will still be processed, and the output will be meaningless as well. Therefore, the DSC assembler adopts an evaluation process to improve the validity of inputs.

The DSC assembler evaluates the suitability of a DSC item for an agent through comparing the inputs and outputs from both sides. The comparison between a DSC item and an agent's request is mainly based on the data types and the number of inputs and outputs. For instance, an agent requests a DSC item that can provide two main outputs including the total sales amount and the average salary. Then a selected DSC item must provide these two outputs. The following procedures show the comparison process:

(a) **Step 1:** If the agent's request does not specify inputs then skip to the next step, otherwise we compare the length of the inputs of the agent's request and the DSC item. We eliminate all the non-compulsory inputs in both the DSC item and the agent's request. If the length of the compulsory inputs (LC) of both sides is equal then we continue the comparison process, otherwise the DSC item is not appropriate for fulfilling the agent's request. The following example illustrates the verification process.

Agent's request inputs:	1	0	0	0	1	⇨	Agent's compulsory inputs:	1	1
DSC item inputs:	1	1	0	0	1		DSC compulsory inputs:	1	1	1

'0' denotes that the input is non-compulsory; '1' denotes that the input is compulsory.

From the above example, we have: $LC_a = 2$ and $LC_d = 3$, where LC_a denotes the length of the compulsory inputs of the agent's request; LC_d denotes length of the compulsory inputs of the DSC items. Here, $LC_d > LC_a$. Therefore, the selected DSC item is not appropriate for the agent's request.

If $LC_d = LC_a$, then we compare the data types of the compulsory inputs from both sides. If $DIA_i = DID_i$ then go to the next step, otherwise the selected DSC item is not appropriate for the agent's request. DIA_i denotes the input data types of the agent's request; DID_i denotes the input data types of the DSC items; i is from 1 to LC_d, with the incremental change = 1.

(b) **Step 2:** If the agent's request does not specify outputs then the verification process is accomplished. Whether the DSC item is appropriate for the agent's request is based on the semantic matching of the capabilities descriptions between the two sides, and the comparison process performed in the previous step. If the agent's request specified the output, then we compare the length of the outputs of both sides. The outputs do not distinguish the compulsory and non-compulsory values; therefore, it is not necessary to perform the elimination procedure of non-compulsory values.

If $LP_a \neq LP_d$, then the DSC item is not appropriate for the agent's request. If $LP_a = LP_d$, then we compare the data types of the outputs from both sides. If $DOA_i = DOD_i$ then the DSC item can be selected as an appropriate item for the agent's request, otherwise the selected DSC item is not appropriate for the agent's

request. LP_a denotes the length of the outputs of the agent's request; LP_d denote the length of the outputs of the DSC item; DOA_i denotes the output data types of the agent's request; DOD_i denotes the output data types of the DSC items; i is from 1 to LC_d, with the incremental change = 1.

The Structure of the DSC usage centre

The DSC usage centre consists two sections, which include the front section and the backup section. The front section extracts the DSC items from the back section, which are the most frequently used DSC items. This two section-based structure adopts the methodology of the CPU cache design in operating systems, which stores copies of the data from the most frequently used main memory locations. The reason for dividing the DSC usage centre into two sections is that: the number of the DSC items in the DSC usage centre could be large; however, there are only a number of DSC items are used frequently within a certain period. Therefore, it can improve the system efficiency through deploying a section with a relatively smaller size, which contains the most frequent used items within a certain period.

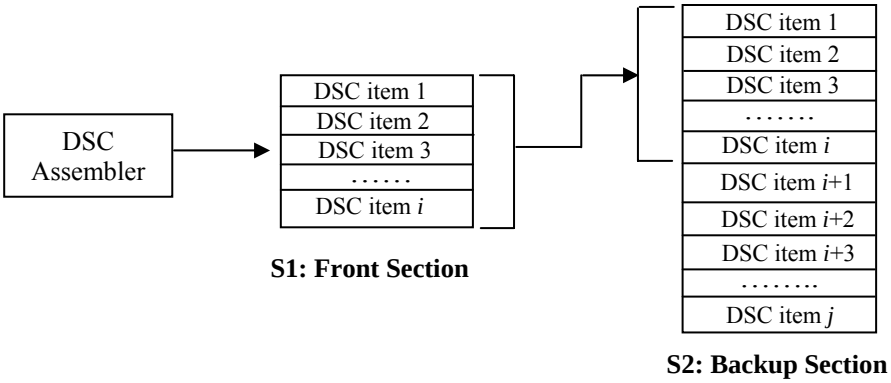


Figure 4.9. The Two Section-based Structure of the DSC Usage Centre.

Figure 4.9 illustrates the structure of the two section-based structure of the DSC usage centre. *S1* denotes the foregrounding section; *S2* denotes the item backup section; all the DSC items in *S1* can be found in *S2*, which can be expressed as: $S1 \subseteq S2$. The DSC items stored in *S1* are obtained from *S2*, which are the most frequently used items in *S2*. The DSC assembler is responsible for searching a DSC item and importing it to the requesting agent. The following calculation processes illustrate how to extract the DSC items in *S2* and store to S1.

(a) **Step 1:** Calculating the recent visit factors in S2 within time period $[t_a, t_b]$.

$$V(t_i) = \begin{cases} 1, \ \textit{the DSC item is visited}, \\ \\ 0, \ \textit{the DSC item is not visited}. \end{cases}$$

where t_i is the time and $V(t_i)$ denotes whether a DSC item is visited at time t_i; i is a variable, which denotes different visiting time within a time period.

The total number of visits to a DSC item is calculated as:

$$n = \sum_{i=a}^{b} V(t_i), \text{ and } V(t_i) = 1. \qquad \text{(Eq. 4.1)}$$

where a, b denote the starting point and the ending point of the time period respectively; n is the total number of visits to a DSC item within the time period $[t_a, t_b]$; t_a is the starting point; t_i is the time point that the DSC item is visited within the time period. If a DSC item is visited at time t_i within $[t_a, t_b]$, then this DSC item is the i^{th} visit to the DSC item and $1 \le i \le n$. For instance: there are a total of 100 visits within $[t_a, t_b]$, i.e. n = 100. The DSC item is visited at time t_{20} ($t_a \le t_{20} \le t_b$) and is the 20th visit to the DSC item, then $t_i = t_{20}$.

Therefore, a DSC item's recent visit factor can be calculated as the following:

$$\begin{aligned} E_i &= \sum_{i=1}^{n} \left(V(t_i) + \frac{t_i - t_a}{t_i} \times V(t_i) \right)^{\log_n (\frac{t_i}{t_a} \times n)} \\ &= \sum_{i=1}^{n} \left((2 - \frac{t_a}{t_i}) \times V(t_i) \right)^{\log_n (\frac{t_i}{t_a} \times n)} \\ &= \sum_{i=1}^{n} \left((2 - \frac{t_a}{t_i}) \times V(t_i) \right)^{(\log_n t_i - \log_n t_a + 1)} \end{aligned} \qquad \text{(Eq. 4.2)}$$

where E_i denotes the recent-visit-factor value of a DSC item at time t_i, which indicates a DSC item's usage within the period $[t_a, t_b]$; n is the total number of visits to the DSC and $1 \le n$, (Eq. 4.1, 4.2 only calculate the recent-visit-factor value when there is visit to the DSC item, if $n = 0$ then $E_i = 0$.); the exponent in Eq. 4.2, i.e. $(\log_n t_i - \log_n t_a + 1)$, is to enlarge the recent-visit-factor value. If t_i is more recent then its E_i value is greater, meanwhile *log* function is used to limit the exponent value. Eq 4.2 indicates that: when t_i increases (more recent) then E_i increases. In other words, if a visit to the DSC item is more recent then its E_i is

greater. The AOCD system developers or users can define the time period (i.e. [t_a, t_b]), which is configurable, to initialize and update the DSC items in *S1*.

The DSC items in *S2* are ranked according to E_i. The DSC items with higher E_i scores are listed on the top of *S2*; *S1* extracts a number of the DSC items that are listed on the top of *S2*. The next step calculates the number of the DSC items that *S1* extracts from *S2* (i.e. the size of *S1*).

(b) **Step 2:** Calculating the size of *S1*.

The size of *S1* is based on the AOCD system's DSC item usage frequency; it should also take the miss-rate into consideration. We use a dynamic alteration method to determine *S1*'s size. An empty *S1* extracts the maximum number of the DSC items with top E_i values from *S2* within *S1*'s predefined size limit. After a period ([t_0, t_s]) of running (initial running period), some DSC items in *S1* will be replaced by the items in *S2* and some will be removed from *S1* because of low-usage-efficiency (refer to the next section). Thus, the total number of DSC items in *S1* defines the preliminary size of *S1*, i.e. *A1*, after the initial running period. We calculate the total number of DSC items in *S2* as its size, i.e. *A2*.

Once E_i, *A1*, and *A2* are calculated, the DSC usage centre starts to dynamically alter *S1*'s size according to the miss-rate of searching DSC items in *S1* within a user defined period, i.e. the *alteration period*. Figure 4.10 illustrates the dynamic alteration process.

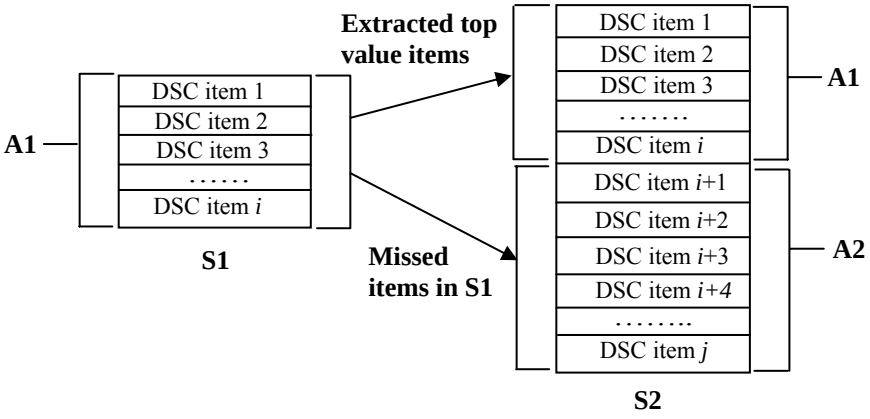


Figure 4.10. Dynamic Alteration Process.

In the dynamic alteration process, *A1* is affected by E_i values and the miss-rate of searching DSC items in S1. The following inequality describes the calculation process.

$$\frac{\sum_{x=1}^{A1} E_x}{\sum_{y=1}^{A2} E_y} \approx \frac{A1}{(A2 - A1)} \times \frac{1}{ms}$$

$$\Rightarrow \quad A1 \approx \frac{\sum_{x=1}^{A1} E_x \times (A2 - A1) \times ms}{\sum_{y=1}^{A2} E_y}$$

$$\Rightarrow \quad A1 \approx \frac{ms \times \sum_{x=1}^{A1} E_x \times A2}{\sum_{y=1}^{A2} E_y + \sum_{x=1}^{A1} E_x \times ms} \quad . \qquad \text{(Eq. 4.3)}$$

where *ms* denotes the miss-rate of searching DSC items in *S1* within the alteration period (this period can be various according to different system applications), and it is always ≤ 1; E_x and E_y denote the E_i values of the DSC items in *A1* and *A2* sections, respectively.

This inequality indicates that: A1 should be approximately equal to (*A2-A1*) multiplied by the ratio of the total E_i values of all the DSC items in A1 section to the total E_i values of all the DSC items in (*A2-A1*) section, and the miss-rate. This is because the portion of the DSC items' E_i values (*A1* section) in the DSC items' E_i values (*A2-A1*section) should be greater than the ratio of A1 to (*A2-A1*); and the miss-rate reflects this margin.

If initial *A1* does not satisfy this inequality, then the DSC usage centre needs to alter *A1* until it approximately equals the right side of Eq. 4.3. The system users can define the alteration period based on different application requirements. For instance, the alteration period could be longer in some applications because their DSC items require more processing time. Moreover, the process is dynamic, which allows the DSC usage centre to alter the *S1* size regularly according to the computation results based on Eq. 4.3.

The above two steps explain the operating process of the DSC usage centre. The system users can configure the DSC usage centre through adjusting the processing time and the actual database size of the DSC usage centre.

Reuse and dismissal of the DSC items

The DSC items in the DSC usage centre that have low usage efficiency will be sent to the Matrix learning centre for reuse or dismissal. A DSC item is regarded as low usage efficiency when this DSC item is constantly on the bottom of *S2* within a period of time [t_s, t_{re}], which is called the critical period for removal and it can be configured by system users.

The usage efficiency is calculated through combining the probability of the visits to a specific item with the recent-visit-factor value E_i. We have assumptions as follows:

(i) There are a total of *x* DSC items in an agent.

(ii) All the DSC items in the agent have been visited the total of *m* times within period [t_a, t_b].

(iii) There is no concurrent visit to the DSC items. In other words, only one DSC item can be visited at the same time.

Value *m* is based on Eq. 4.1. Therefore, $m = \sum_{i=a}^{b} V(t_i)$, and $V(t_i) = 1$. Based on Eq. 4.2, the average value of all the DSC items' E_i values, i.e. $AVG(E_i)$, within [t_a, t_b] is:

$$AVG(E_i) = \frac{\sum_{i=1}^{m}\left(\left[(2-\frac{t_a}{t_i})\times V(t_i)\right]^{(\log_n t_i - \log_n t_a + 1)}\right)}{m} \qquad \text{(Eq. 4.4)}$$

The standard deviation [149] of all the DSC items' E_i value is:

$$SDV(E_i) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\left((2-\frac{t_a}{t_i})\times V(t_i)\right)^{(\log_n t_i - \log_n t_a + 1)} - AVG(E)\right)^2}$$

$$= \sqrt{\frac{1}{m}\left(\sum_{i=1}^{m}\left((2-\frac{t_a}{t_i})\times V(t_i)\right)^{2(\log_n t_i - \log_n t_a + 1)}\right) - \left(2AVG(E)\sum_{i=1}^{m}\left((2-\frac{t_a}{t_i})\times V(t_i)\right)^{(\log_n t_i - \log_n t_a + 1)}\right) + m(AVG(E))^2}$$

$$= \sqrt{\frac{1}{m}\left(\left(\sum_{i=1}^{m}\left[(2-\frac{t_a}{t_i})\times V(t_i)\right]^{2(\log_n t_i - \log_n t_a + 1)}\right) - m\left(AVG(E)\right)^2\right)}$$

$$= \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left((2-\frac{t_a}{t_i})\times V(t_i)\right)^{2(\log_n t_i - \log_n t_a + 1)} - (AVG(E))^2} \quad \text{(Eq. 4.5)}$$

We consider removing the DSC item with an E_i value far below $AVG(E_i)$. Therefore, the lowest E_i value is:

$$E_k = MIN(E_1, E_2, ...E_m). \quad \text{(Eq. 4.6)}$$

where dataset ($E_1, E_2, \ldots E_m$) denotes all the DSC items' E_i values within the period [t_a, t_b]; E_k denotes the lowest E_i value in the dataset. If we have:

$$\frac{|E_k - AVG(E)|}{SDV(E)} \geq 3. \quad \text{(Eq. 4.7)}$$

We define this DSC item as low usage efficiency. This definition is based on *Chebyshev's* theorem [54]:

"If μ and σ are, respectively, the mean and the standard deviation of the distribution of the random variable *x*, then for any positive constant *k* the probability that *x* will take on a value which is at most $\mu - k\sigma$ or at least $\mu + k\sigma$ is less than or equal to $1/k^2$."

This theorem is also expressed as: $P(|x-\mu| \geq k\sigma) \leq 1/k^2$. According to *Chebyshev's* theorem, at least 89% of the E_i values disperse within 3 standard deviations from the $AVG(E_i)$ value. Therefore, any value greater than this range is considered as an outlier that has low-usage-efficiency.

If a DSC item in a DSC usage centre is identified as a low-usage-efficiency item, then it will be sent to the Matrix learning centre. These low-usage-efficiency DSC items can be discovered and re-deployed by other Matrices, or can be updated by the Matrix learning centre through information updating.

The AOCD-based agents also use this evaluation methodology to identify whether a DSC item is low-usage-efficiency. If a DSC item is identified as low-usage-efficiency in an agent then it will be unplugged from the DSC container and sent to the DSC usage centre. This reuse and dismissal method is also used to remove the low-efficient items in the foregrounding section of the DSC usage centre; it can reduce the section size and improve the system efficiency.

The Matrix learning centre

The process of converting the information in the Matrix learning centre to the DSC usage centre is based on two methodologies: the automatic conversion method and the human-participation conversion method. At the current stage, the human-participation conversion process is the major method in the AOCD framework.

The AOCD system users and developers can update their system through developing a number of new DSC items according to their specific requirements and the available information in the Matrix learning centre. Figure 4.11 shows the information updating process in the Matrix learning centre.

The Matrix learning centre updates its information mainly through updating its database (knowledge-base) and the DSC item information. For instance, a public library implements the AOCD-based system. The library takes in new books through various methods; and one of the major methods is online borrowing from a state library. Once the state library updates the loaning book information for the public library online, then these books listed online will be delivered to the library on the next day. Therefore, this library's Matrix learning centre can acquire the new book information from the state library's website, and update its database for its agents and Matrices. The Matrix learning centre can update the information about agents and Matrices once their statuses are changed. For instance, if an agent changed its name then the Matrix learning centre can capture this change and update information for the corresponding agents and Matrices.

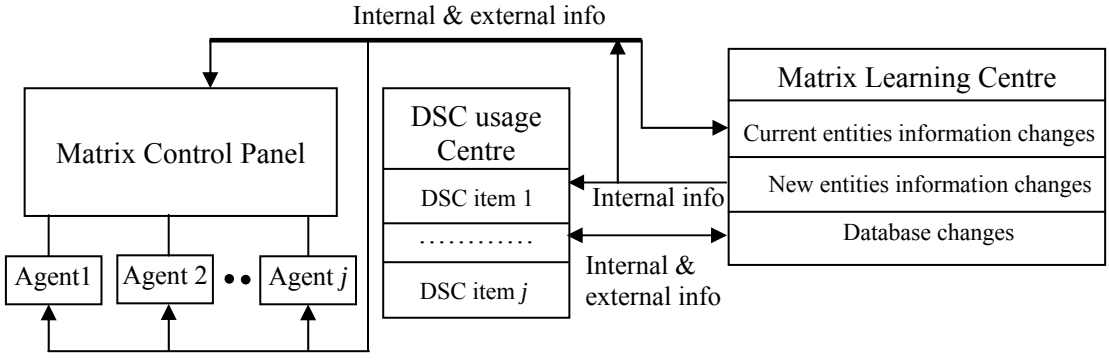


Figure 4.11. Information Updating Process of the Matrix Learning Centre.

The DSC items in the DSC usage centre can also be updated automatically through receiving information from the Matrix learning centre. For example, a DSC item is connected to the database that is located in the Matrix learning centre. Once the

Matrix learning centre updates the database then the DSC item will be updated automatically.

As shown in Figure 4.11, the Matrix learning centre mainly acquires three types of information including: (*i*) the information changes on current entities, which are based the existing entities' information changes (agents, Matrices, DSC items, etc.), such as the change of an agent's name, the change of a Matrix's capabilities, etc. (*ii*) the new entities information, which includes the information about a new developed DSC item, a new agent, etc. (*iii*) the database changes, which include the external data updates, such as online transaction information update, external data inputs, etc.

4.1.5. Interfaces Between the Matrix and Agents

The process of allocating an agent-Matrix interface involves the following procedures: the dynamic detection for interface allocation; updating the agent capability registration in the Matrix register; and establishing links between agents and the Matrix.

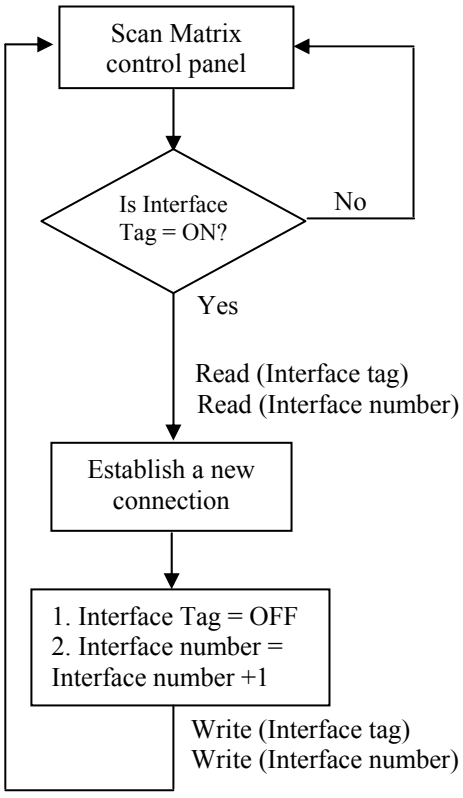


Figure 4.12. Agent Interface Allocation Process.

We adopt a flexible method of designing Matrix-agent connections, where the number of interfaces provided by a Matrix to connect to agents is unlimited. We use a

Dynamic detection method to identify the number of interfaces generated by the Matrix.

In the Dynamic detection method, an *Interface Tag* is deployed to indicate whether the Matrix has received a connection request. Once the Matrix control panel detects a new connection, a new interface will be allocated to the connection. The Matrix control panel examines the interface tag and allocates interfaces to agents according to the value of the interface tag when the AOCD system starts to load the Matrix control panel. A new connection request changes the value of the interface tag that triggers the re-allocation of the connection interface resources. Figure 4.12 shows the process of allocating the interfaces to the agents. The configuration of scanning frequency in the dynamic detection method depends on the system transaction volume. Users can adjust the frequency rate in the Matrix control panel to set the optimum rate for a specific system.

4.2. A Unified Matrices Structure for Matrices Cooperation

The AOCD framework can solve a complex problem through the cooperation and coordination among the numerous Matrices. As mentioned in Chapter 3, a complex problem is usually solved through agent cooperation within a Matrix; however, in many cases the problem cannot be solved in the Matrix alone. The external agents are often required to participate in the problem-solving process. Therefore, the cooperation among various Matrices is necessary for the problem-solving process in the AOCD framework.

Moreover, The Matrix-agent connection is the combination of the centralised topology and the peer-to-peer topology. The bottleneck problem rises when the framework expands to a large scale because of the limitation of the centralised topology. In other words, the Matrix-agent connection is only suitable for a small-scale network because the central component, the Matrix, may encounter an overloading problem when many agents try to connect to the Matrix. To solve the bottleneck problem, we introduce a *unified Matrices structure*. In the unified Matrices structure, a number of Matrices are connected through the Matrix interface layer as shown in Figure 3.2 in Chapter 3.

This unified Matrices structure deploys a key component, namely the Matrix network maintainer, to manage the cooperation and coordination among the Matrices.

4.2.1. An Overview of the Unified Matrices Structure

Each Matrix in the AOCD framework is connected by a number of agents that can deal with various problems in different domains. However, not every request can be solves by the internal agents in a Matrix. Once a request cannot be solved in a Matrix, the Matrix will forward the request to its nearest or most familiar Matrix (see section 4.2.2) for further processing. This is the fundamental motivation of establishing the unified Matrices structure in the AOCD framework. There are two types of connecting requests. The first type of request is from agents for Matrix-agent connections. The second type of request is from the Matrix for Matrix-Matrix cooperation.

The connections between Matrices are established through the external cooperation layer in each Matrix. The external cooperation layer not only connects the Matrix to external agents but also provides interfaces with other Matrices. The connections between agents and a Matrix are direct and centralised, whereas the connections between Matrices are based on some common carriers, such as the Internet or a local area network. To avoid the bottleneck problem that might happen in this structure, there is only one connection allowed existing between any two Matrices at one time. The Matrix network maintainer is deployed to manage this unified Matrices structure. The detailed design of the Matrix network maintainer and the searching algorithms are introduced in section 4.22 and 4.23 respectively.

The major advantages of using this structure are summarised as follows:

(1) *Improvement of the communication and transmission efficiency*. In this structure, the Matrix-agent connection is limited to a small scale, which reduces the communication and transmission volumes. Therefore, the bottleneck problem can be minimised. The experimental results show the efficiency of time consumption in the hybrid structure when concurrent problems occur (see Section 8.1.4).

(2) *Minimisation of system development overheads*. The unified Matrices structure allows the Matrices to share their agents for solving various problems; it could highly reduce the system development costs for many organisations and departments as they can use others' agents within the AOCD framework. For instance, Department A and B deploy two AOCD-based Matrices in their own departments, respectively. Each department's Matrix has a number of agents and these agents have very different but complementary functionalities. Through the

unified Matrices structure, these two departments can solve problems cooperatively without redundantly developing agents that are similar to each other.

(3) *Minimisation of the concurrency problem*. The number of agents connecting to a Matrix is limited, which results in the minimisation of the concurrency problem.

(4) *Improvement of problem-solving efficiency*. The allocation of agents to a Matrix is based on the specific requirements of an organisation and the capabilities of each agent are registered to the Matrix. The functionalities and capabilities of a Matrix can be identified easily. This structure is efficient to identify which Matrix can deal what kind of problems and search a corresponding Matrix.

4.2.2. Matrix Network Maintainer

The Matrix network maintainer provides the information about the Matrices' distribution and their functionalities. Figure 4.13 gives a schematic diagram of the Matrix network maintainer.

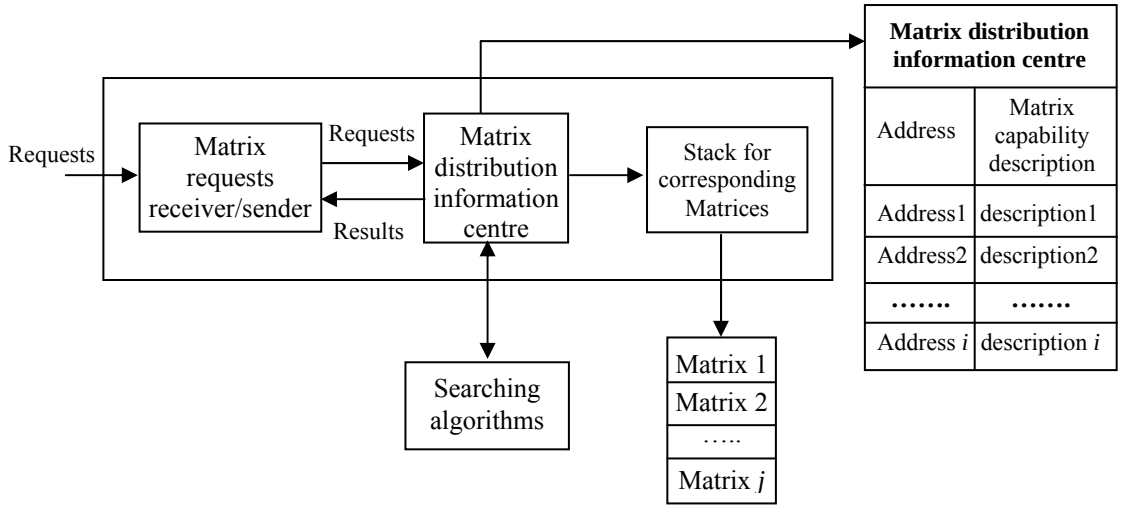


Figure 4.13. A Schematic Diagram of the Matrix Network Maintainer Design.

The following figure illustrates the working process of the Matrix network maintainer to further explain the Matrix network maintainer design.

The Matrix distribution information centre is a major component of the Matrix network maintainer, which contains the information on the addresses of the connected Matrices and the general Matrix capabilities descriptions for the searching service-provider Matrices. An AOCD framework normally has a number of Matrix network maintainers and each Matrix network maintainer is managed by a Matrix with relatively superior computing performance.

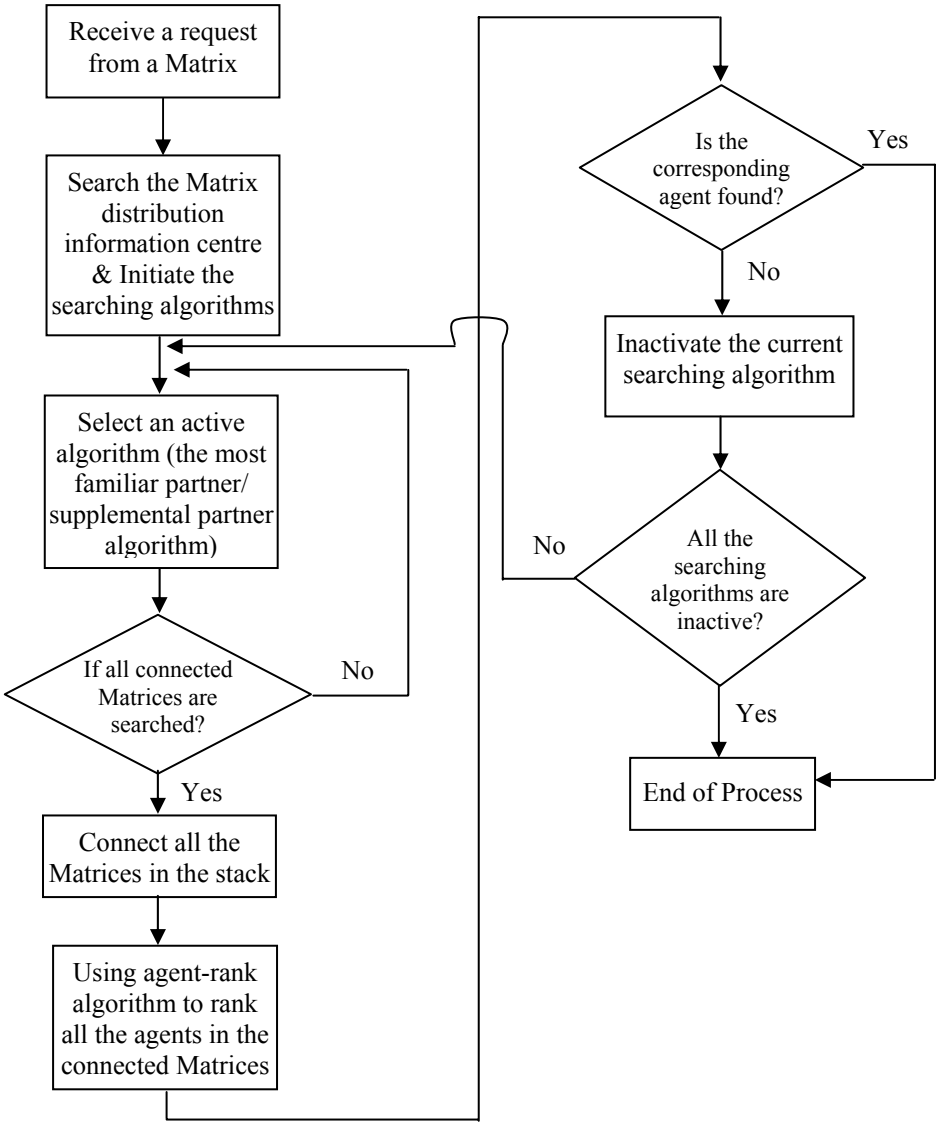


Figure 4.14. Working Process of the Matrix Network Maintainer.

4.2.3. Matrix Searching Algorithms for Service-Provider Matrices

The process of searching a service-provider Matrix according to a requesting Matrix in a unified Matrices structure follows two principle guidelines [168], including the *Most familiar partner* method and the *Supplemental partner* method. When dealing with a request for cooperation (searching for external Matrices), we often encounter two situations: one situation is to search for a partner Matrix that has similar capabilities to the requesting Matrix. The other situation is to find a partner Matrix that can provide some services that the requesting Matrix does not have. For instance, when an accounting agent in a company's Matrix system requires last year's regional taxation statistics, then it would look for another accounting agent that could provide regional statistical data, and the cooperation between these two agents might

happen regularly. In this case, we use the most familiar partner method. Another example is as follows. There is a huge natural disaster in the region where this company is located and the company's accounting agent requires a climate forecast report to analyse: (1) whether it is necessary to inject more funds to strengthen the company's buildings and (2) the amount of funds. In this situation, the requesting Matrix is looking for a very unfamiliar service provider Matrix, which has very different functionalities. In this case, we use the supplemental partner method. Normally, the Matrix network maintainer utilises both algorithms respectively when one of the algorithms cannot satisfy the requests from Matrices.

Most familiar partner method

We use a Matrix capability description table to describe the functionalities and capabilities of a Matrix as shown in Table 4.2.

Table 4.2 Matrix Capability Descriptions.

Agent Functionality Keywords	Agent ID	Domain
Salary statistics, superannuation expenses, facility maintenance, etc.	Agent 1	Accountant
Warehouse stocks, sales condition, transportation expenses, etc.	Agent 2	Accountant
….	….	….
Market share history, market trends, major rivals' products, etc.	Agent *i*	Marketing

This table contains the information on all the connected agents' functionalities and the domain information, which indicates the major areas that an agent is responsible for. Each agent's functionality is described by several keywords and these keywords also represent the functionalities and capabilities of the Matrices. Normally, there are some intersections among the Matrices' capabilities as Figure 4.15 shows.

In Figure 4.15, we aggregate all the keyword descriptions of the other Matrices, which intersect with Matrix 1's keyword descriptions. If there is one keyword intersection between the requesting Matrix and a corresponding Matrix, we add a score, namely the *intersection score*, for these two joined Matrices, expressed as $R_{i \to j}$,

which means Matrix i and j intersect and their intersection score is $R_{i\to j}$. If a corresponding Matrix has the most keyword intersections with a requesting Matrix (compared to other Matrices), then this Matrix is regarded as the most familiar partner of the requesting Matrix.

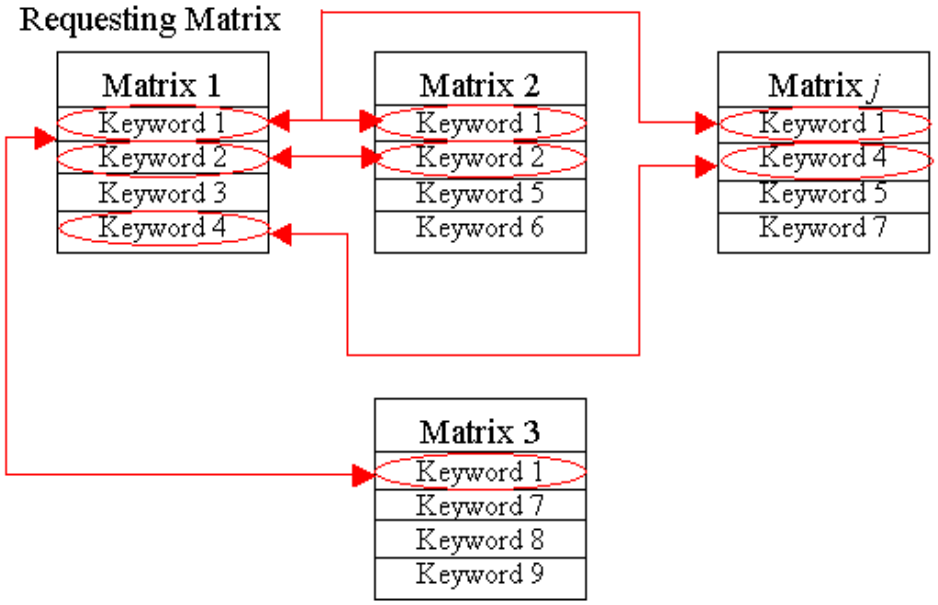


Figure 4.15. Intersections of Matrices' Capabilities [168].

When there are two or more corresponding Matrices having the same number of keyword intersections with a requesting Matrix, we select the Matrix that has keyword intersections least shared by other Matrices as the most familiar partner. Thus, the calculation of the most familiar partner method can be expressed as follows. In the following example, we search a most familiar partner of Matrix 1.

Step 1. $S_i \in \Omega, \quad K^j \in S_i$

where i = 1, 2, ….n. S_i denotes a set of keyword aggregation of Matrix i; Ω denotes the total aggregation of all keywords in all Matrices. j = 1, 2, ….m; j denotes the keyword number in a set; K^j denotes the keyword j in a keywords set; Φ denotes empty set.

$$(S_1 \cap S_j \neq \Phi) \to (R_{1\to j} = \left|S_1 \cap S_j\right|)$$

$$(S_1 \cap S_j = \Phi) \to (R_{1\to j} = 0) \qquad \text{(Eq. 4.8)}$$

where S_1, S_j denote the keywords set 1, j of Matrix 1, j, respectively. $R_{1\to j}$ denotes the intersection score of Matrix 1 to Matrix j.

Step 2. If $R_{1\to j}$ is the maximum value among all the other intersection scores with

Matrix 1, then Matrix *j* is the most familiar partner of Matrix 1 and the most familiar partner process is completed. If $\exists(R_{1\to j} = R_{1\to g})$ then go to Step 3 ($R_{1\to g}$ denotes the intersection score of Matrix 1 to Matrix *g*).

Step 3.

$$(K^i \subset (S_1 \cap S_j \cap ... \cap S_x)) \to (R_{1\to j} = R_{1\to j} + 1/M_i) \quad \text{(Eq. 4.9)}$$

where M_i denotes the total number of the Matrices (including the Matrix 1) that have the same intersections with Matrix 1. S_x denotes a keyword set of Matrix *x* that intersects with S_1 and S_j at keyword K_i.

$$R_{1\to j} = \sum_{i=1}^{\#S_1} (R_{1\to j} + 1/M_i) \quad \text{(Eq. 4.10)}$$

where $\#S_1$ denotes the total elements in Set 1.

$$(K^i \subset (S_1 \cap S_g \cap ... \cap S_y)) \to (R_{1\to g} = R_{1\to g} + 1/Z_i) \quad \text{(Eq. 4.11)}$$

where Z_i denotes the total number of Matrices (including the Matrix 1) that have same intersections with Matrix 1. S_y denotes a keyword set of Matrix *y* that intersects with S_1 and S_g at keyword K_i.

$$R_{1\to g} = \sum_{i=1}^{\#S_1} (R_{1\to g} + 1/Z_i) \quad \text{(Eq. 4.12)}$$

If $R_{1\to j} < R_{1\to g}$ then Matrix *g* is the most familiar partner of Matrix 1 and the process is over. If $R_{1\to j} = R_{1\to g}$ then go to Step 4.

Step 4. Choose the nearest neighbour as the most familiar partner of Matrix 1. If there are two or more nearest neighbours then randomly choose a Matrix among them.

Supplemental partner method

In the supplemental partner method, we choose a service-provider Matrix of a requesting Matrix that has minimum keywords intersections with the requesting Matrix. The procedures of the supplemental partner method are similar to the most familiar partner method. However, the supplemental partner method seeks a service provider Matrix that has a minimum intersection score. We still use the same example to search a supplemental partner for Matrix 1.

Step 1. It is the same procedure as in the most-familiar-partner method.

Step 2. If $R_{1\to j}$ is the minimum value among all the other intersection scores with Matrix 1, then Matrix *j* is the supplemental partner of Matrix 1 and the supplemental partner process is completed. If $\exists(R_{1\to j} = R_{1\to g} = 0)$ then choose a nearest partner among these same intersection score Matrices. If there are two or more nearest neighbours then randomly choose a Matrix among them.

If $(R_{1\to j} = R_{1\to g} > 0)$ then go to Step 3 ($R_{1\to g}$ denotes the intersection score of Matrix1 to Matrix *g*).

Step 3. It is similar to the most familiar partner method. The difference is that the supplemental partner method seeks a minimum intersection score.

$$R_{1\to j} = \sum_{i=1}^{\#S_1}(R_{1\to j} + 1/M_i)$$

$$R_{1\to g} = \sum_{i=1}^{\#S_1}(R_{1\to g} + 1/Z_i) \qquad \text{(Eq. 4.13)}$$

If $R_{1\to j} > R_{1\to g}$ then Matrix *g* is the supplemental partner of Matrix 1 and the process is over. If $R_{1\to j} = R_{1\to g}$ then go to next step.

Step 4. Choose a nearest partner among these same intersection score Matrices. If there are two or more nearest neighbours, then randomly choose a Matrix among them.

The most familiar partner and supplemental partner methods are basically for matching capabilities and functionalities among Matrices. For more specific agent capability search and agent relationship management, we use a novel *Agent-rank* algorithm. The agent-rank algorithm is the most efficient means to search a service provider for a request. The most familiar partner and supplemental partner methods help to narrow the searching range for the agent-rank algorithm. These two methods based on Matrix matching are complementary; the combination of these two methods could enhance the matching efficiency and accuracy. Nevertheless, the agent-rank algorithm is essential for searching a specific service provider (agent) of a request.

4.2.4. Super-node Model in the Matrices Cooperation

The unified Matrices structure introduced in the previous sections has its limitations on central control and fault-tolerance. These limitations lead to the circumstances of low efficiency in the matching process, massive traffic chaos in the cooperation process and vulnerability to partial system breakdown.

A super-node model therefore is proposed to tackle the above problems. The super-node model [48] has been extensively applied to many online systems, such as KaZaA [84], SkypeNet [133], etc. In a super-node model, a number of nodes are selected as super nodes, which manage a limited number of other nodes. The major advantages of the super-node model include reducing time and bandwidth for search, enhancing manageability, and improving load balancing [48]. This model avoids mesh peer-to-peer connections, which minimizes the probability of the occurrence of concurrency. The concurrency problem often causes repetitions of communications, which increases network traffic volume. As the decrease of the concurrency in the super-node model, the traffic volume also decreases. Therefore, the network performance is improved. Figure 4.16 shows an example of a super-node-based network.

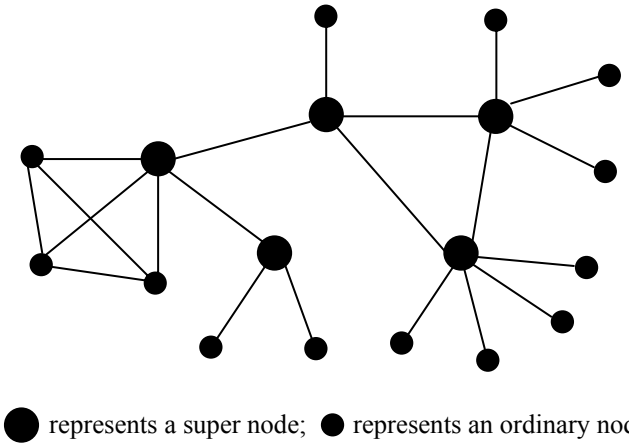


Figure 4.16. An Example Super-node-based Network (based on [48]).

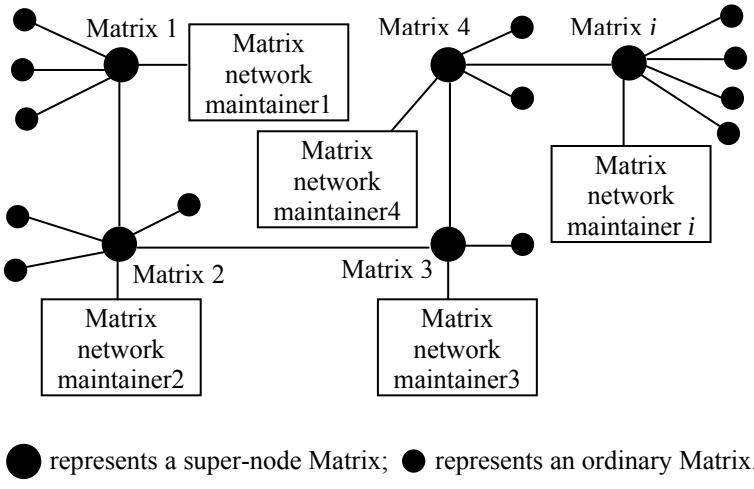


Figure 4.17. A Super-node Model in the Unified Matrices Structure.

The application of the super-node model in the Unified Matrices Framework enhances the fault-tolerance capability and the system manageability. In the unified Matrices structure, the super-nodes are those selected Matrices that play a coordinator role in processing the other Matrices' requests. These super-nodes receive the requests from the other Matrices and search the corresponding Matrices.

The criteria of determining whether a computing terminal is suitable to be a super-node are quite simple [166], which include (*i*) it must have a high bandwidth connection, (*ii*) it needs to have a reasonable computing capability and (*iii*) it should be flexible and efficient in entering and exiting a network. Generally, each super-node Matrix is directly connected to a Matrix network maintainer as shown in Figure 4.17.

In Figure 4.17, the Matrices that are connected to Matrix 1 search their corresponding Matrices mainly within the small network managed by Matrix network maintainer 1 and super-node Matrix 1. In other words, a request from the Matrices group controlled by Matrix 1 are firstly dealt within the group; the request is sent to other Matrices groups only when Matrix network maintainer 1 cannot find an appropriate Matrix for it.

4.2.5. Security Control in the Agent-Matrix Link Layer

The security issue is an ongoing research area for the AOCD framework. At current stage, the AOCD framework adopts the *Global Access Control* (*GAC*) mechanism [140] in the Agent-Matrix link layer. The security procedures of establishing an agent-Matrix link are described as follows:

In the first step, an agent sends a connecting request and its identification information to the Matrix control panel for authorisation.

In the second step, the Matrix register deploys an access authorisation table, which contains a set of authorised agents $A = \{ag_1, ag_2, ag_3, ...ag_n\}$; each agent has a specific access policy that defines the agent's purview of accessing the Matrix control panel and cooperating with other agents.

The final step verifies the requesting agent's information based on the access authorisation table and determines whether granting the admission to the requesting agent and the level of admission.

The security control method in the AOCD framework only addresses the agent-Matrix connection level at this stage. Nevertheless, the future work will extend the security issue to the agent-to-agent connection level.

4.3. Matrix Development Life Cycle

The development life cycle of the Matrix in the AOCD framework is standardised, which means all the Matrices go through the same development procedures regardless of different organisations and applications. In other words, all the Matrices in the AOCD framework are same and they can be applied to any organizations without considering the organisational environment. Readers might doubt about the feasibility of this design in terms of complexity, specialty and variety of different systems. However, it can be sure that the standardised Matrix design would not affect the system variety and specialty because the functionalities of AOCD systems are fulfilled by various service-agents. Therefore, this standardised Matrix design pattern highly enhances system efficiency and compatibility in the AOCD framework. Similar to most of the general software packages in the market, the Matrix development life cycle basically adopts and modifies the traditional system development life cycle and consists of five major stages [168], which include general user requirement analysis, framework design, coding and documentation, testing and maintaining, and implementation and evaluation.

In the first stage, the Matrix performance evaluations based on different applications are analysed. End-user requirements and comments on the Matrix design, which include user interface design, Matrix design consistency, etc., are also analysed.

The framework design stage involves the key-component-modularisation, relationship diagram of modules, communication language specifications for the Matrix, specifications of module-connection, and specifications of major changes to the previous version based on the first-stage analysis (if applicable).

The coding and documentation stage involves individual module coding, such as the Matrix interface module, the Matrix register module, the Matrix control panel module, the Matrix network maintainer module, the DSC usage centre module, etc. The documentations for each module's coding procedure are also required.

In the testing and maintaining stage, every new version of a Matrix is required for testing to ensure the AOCD system's accuracy and efficiency. The results from the testing process are applied to the maintenance process to correct possible errors and maintain the system consistency.

In the final stage, the finalised Matrix packages will be installed to end-users' terminals or servers. The installation and configuration of the Matrix packages can be

operated by end users without assistance of any professionals. The evaluation process is based on user-define requirements and current Matrix performance.

4.4. Summary

The Matrix provides a virtual platform for intelligent agents through managing Matrix-agent relationships in an agent society. The Matrix maintains a superior transmission performance in its virtual platform through combining centralised and decentralised topologies. The Matrix also presents seemly mobility as the peer-to-peer connections between agents only exist temporarily.

The Matrix employs a DSC usage centre to assure that every agent is upgradeable. Different from agent mediators and facilitators in other middle-agent-based systems, a Matrix in the AOCD framework not only coordinates its internal agents but also provides its internal agents with various information resources, such as the DSC items, data resources, etc. An agent can perform self-upgrade through obtaining new DSC items from the Matrix and replacing dated items. This Matrix-agent connection design simplifies the self-upgrade processes of an agent, which are rather complicated in many agent-based systems.

To solve the bottleneck problem of the centralised topology in the Matrix-agent design, we introduce a unified Matrices structure, in which the AOCD framework is decomposed into a number of Matrices. Each Matrix in this structure is responsible for dealing with a limited number of agents within a relatively small scale of area. The unified Matrices structure adopts the Supernode model, which has been proven to be efficient in various applications and literature studies [48, 84, 133]. Communication and cooperation within the AOCD framework are based on Matrix cooperation (through the keyword intersection methods, such as most familiar partner method or supplemental partner method) and agent matching (through capability and relevance match, such as the agent-rank algorithm). The unified Matrices structure consists of a number of Matrices groups; and one of the Matrices in each group is selected as a super-node to manage a group. These selected super-node Matrices normally possess superior computing performances; and each super-node Matrix manages the group through using a Matrix network maintainer that contains the Matrix distribution information in the group.

CHAPTER 5
AGENT DESIGN METHODOLOGIES

5.1. Introduction

Intelligent agent has been defined in various works of literature but there is no universally accepted definition. Nevertheless, this situation does not obstruct the development of agent technology. More and more definitions of agent tend to focus on describing agent characteristics. Based on various agent definitions and classifications [70, 125, 162, 163, 174], the major agent characteristics can be summarised as autonomous, adaptive, computational, cooperative, customising, flexible, mobile, perceptive, proactive, and reusable.

An efficient agent design ought to concentrate on these agent characteristics in order to maximize the inherent advantages of agents in an application. Agent design methodologies involve various issues such as agent communication language, agent functionality design, intelligent reasoning techniques, business process adaptation, etc. Apparently, there is no standard principle for agent design at this stage. However, every agent design methodology is related to one of the agent characteristics. Thus, agent design methodologies must exert the advantages of agent characteristics and concentrate on certain areas that can maximise an agent's advantages for a specific application; and this guideline is likely to become a standard principle for agent design. For instance, the design of the AOCD framework is to provide an open structure for Decision Support System (DSS) through using agents [167]. Therefore, adaptability, computational, cooperativity, flexibility, and reusability received more attention than other agent characteristics because an open DSS is dynamic and requires constant updates. Moreover, a complex problem is often solved through cooperation in an open DSS.

An important issue raised in this chapter is the agent capability description. This issue becomes important when it comes to a specific agent system design. It not only takes agent characteristics into account but also incorporates specific system requirements. The cooperation and coordination among heterogenous agents are based on agent capability descriptions, which indicate the responsibilities and functionalities of an agent. Role is a popular term employed to indicate an agent's characters in a

multi-agent system [33, 164]. However, an agent's role in a multi-agent system is based on its practical capabilities. In other words, an agent's role can be regarded as the abstract level of the agent's capability descriptions. Thus, existing role-based agent modelling methodologies can be merged to the agent capability description domain.

This chapter mainly focuses on the issues including agent architectural design, agent communication, agent capability description, and agent development life cycle. These issues reflect the major agent characteristics, which are adaptability (e.g. section 5.1.2.), cooperativity (e.g. section 5.2.), flexibility (e.g. section 5.1.1. & 2) perceptivity (e.g. 5.1.3.), mobility (e.g. section 5.1.4.), and reusability (e.g. section 5.1.1. & 2.). Moreover, the autonomy and computation characteristics are the basics in the overall AOCD-based agent design and they are taken into account in each specific design section.

5.2. Agent Architectural Design

Agent architectural design has become a primary issue in the intelligent agent area as more and more concrete developments has emerged in the intelligent agent area such as JACK (agent development environment) [44], KAoS [20], etc. Existing agent architectural design methodologies brought forth several basic questions including: How would agent architectural design impact on the efficiency of an agent-based application? What are the major factors affecting the agent architectural design methodologies? What are the major methodologies used in current agent architectural design? The following reviews aim to pursue the tentative answers to these questions.

For the first two questions, their answers are based on the analysis of the agent characteristics. An efficient agent architectural design ought to provide concrete functionalities and capabilities actualising the major agent characteristics. Any agent architectural designs, which violate this principle could lead these designs to some difficult situations, such as system low efficiency, pointless development of agent capabilities, lack of development standardisation, and so on; an agent-based system could even lose its inherent advantages without carefully considering the agent characteristics. Therefore, an appropriate agent architectural design virtually guarantees the success of an agent-based system in fundamental terms. Moreover, an

inappropriate agent architectural design can increase the probability of project failure or being over-budget.

For the second question, readers may deduce that one of the major factors that affect the agent architectural design is still the agent characteristic. An agent architectural design must utilise the advantages of agent characteristics in design process thus an agent-based system can benefit from the agent's inherent advantages. It is not possible to illustrate specifically all the factors that affect the agent architectural design because every agent design is case-based, which is subject to the change of a real case. However, in general, the major factors that affect agent architectural design can be summarised as follows: agent-reasoning methods, agent capability description, and agent cooperation and communication methods. Various reasoning methods can be chosen from existing AI (Artificial Intelligence) technology. The adoption of different reasoning methods could impact on the agent design structure. For instance, a Belief-Desire-Intention (BDI) agent can solve a problem through operating within three logic bases, namely, 'beliefs', 'desires', and 'intentions', whereas a Layered agent [162] solve a problem through decomposing the problem and passing to different layers. The way of describing an agent's capabilities affect the task allocation design, agent-matching design, etc.

For the last question, the extensive studies have been conducted in this research to discover the major methodologies of agent architectural design. The methodologies illustrated below represent the typical thoughts in agent architectural design. Firstly, Wooldridge [162] suggests four classes of agent architectures based on different realisation means of decision making in agent design, which include, logic based architecture, reactive architecture, BDI architecture, and layered architectures. Wooldridge's agent architectures basically take the reasoning and decision-making processes into account, which mainly reflect the autonomous and perceptive characteristics. These architectures are constructed differently because their decision-making processes are varied. Secondly, Brazier et al. [23] suggest a generic agent model. In this design model, agents are classified into four different categories and each category represents an agent architecture that contains a number of components. This model mainly focuses on the customising and flexible characteristics. Thirdly, the agent pattern method has been used in agent-oriented software engineering processes [7, 58]. Agent pattern design focuses on the agent design process from the software engineering point of view. This method adopts the object-oriented software

engineering techniques, which provide several standardised design patterns for general agent design. Aridor and Lange [7] describe a set of agent design patterns that mainly focus on flexibility, reusability, and adaptability.

In the AOCD framework, the agent architectural design methodologies mainly place emphasis on the following aspects: autonomous, adaptive, cooperative, customising and flexible. The agent architectural design in the AOCD framework is the combination of several existing design methodologies such as the agent pattern concept, BDI model, ticket pattern for agent travel, etc.

5.2.1. Agent Pattern Design in AOCD

Patterns are efficient solutions to recurring design problems [159], especially for task-orientated applications such as agents, which require frequent cooperation and constant update. Agent pattern design can enhance the flexibility and reusability of an agent-based application. Current agent pattern design classifies agents into different categories and provides different patterns to the agents according to their categories. For instance, Aridor and Lange [7] classify agent patterns into travelling, task, and interaction patterns. Deugo and Weiss [34] classify agent patterns into architectural, communication, travelling, and coordination patterns. However, the variety of agent design patterns causes the problems of system standardisation and integration. These problems hold back the popularisation of agent technology in various applications, and also increase the complexity of system development because the standardisation and integration processes need to be concerned.

In the AOCD framework, the architectural design of every agent follows a general agent pattern. The use of the general agent pattern in AOCD simplifies standardisation and integration processes. Nevertheless, the general agent pattern offers a component, called Domain Specific Component (DSC), to identify each agent's specialised capability. This design principle tallies with the structure of human society, in which various normal persons have general appearances and capabilities and yet each individual person specialises in different working domains, for example some become scientists, some become bankers, etc.

Figure 5.1 shows a general pattern design for an AOCD agent. This general agent pattern combines and modifies existing agent design architectures, such as the Generic agent model [23], SKELETONAGENT [26], the Travelling Patterns [7], Agent Cloning [131], and BDI agent model [162]. The DSC design in AOCD agent is

motived by the *agent specific task* component in the Generic agent model and the *skills* component in SKELETONAGENT. The Travel Control Centre in AOCD agent is based on the Travelling Patterns with more detailed expansions. The Agent Capability Registry in AOCD agent has similarity with the *yellow-pages* component in SKELETONAGENT.

As shown in Figure 5.1, an AOCD agent exchanges information with the environment through two types of interfaces. This design is based on the author's original work, which incorporates several existing mechanisms such as the BDI model. For communication and cooperation with other agents, the agent uses the agent-agent connection interface. For communication and cooperation with a Matrix, the agent uses the Matrix-agent connection interface. If the agent receives any information from its interface connections, e.g. requests, it delivers the information to the Communication and Language Centre to extract recognisable information for the BDI processing component. Once the BDI component receives the recognisable information, it starts to process the information and generate results. The generated results can be further processed by the DSC or the Communication and Language Centre and sent to the environment. A small knowledge base is deployed to help the reasoning process for the BDI component. This small knowledge base only contains the key information for the agent reasoning process and it can only be updated through connecting itself to a main AOCD database.

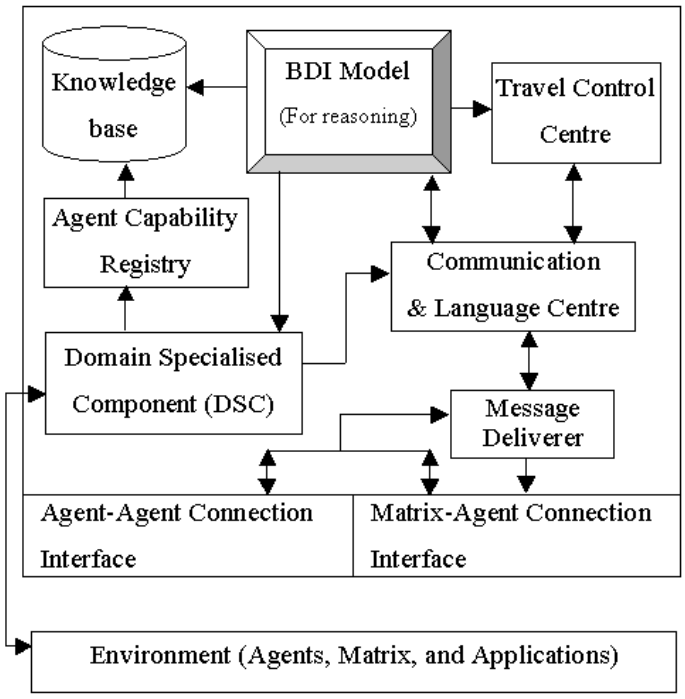


Figure 5.1. General Agent Pattern Design based on AOCD.

5.2.2. The Domain Specific Component Design for Agents

As mentioned in Chapter 4, the Matrix contains a DSC usage centre for updating agents' capabilities and each agent carries a DSC. Each DSC in an agent is a slot container, which offers numerous slots for containing the specialising domain components. Each specialising domain component possesses some special capabilities. Each domain component can plug into a DSC slot to perform specific tasks as an item, for example a domain-component can connect the agent knowledge base to a stock market database to acquire useful knowledge for the agent.

Domain Components	
Connecting to X database	S1 ← Slot number
Printing data to printer Y	S2
Drawing curves bases on Z data set	S3
Idle	S4
Idle	S5

Figure 5.2. DSC Slot Design.

The DSC normally communicates with the environment through the agent-agent or Matrix-agent interfaces; it also can establish communication with the environment directly. A problem arises when some plugged components are rarely used or never used. To solve this problem, a component usage evaluation mechanism is used to examine the usage efficiency of a plugged component. There are two major factors affecting a DSC's usage efficiency, which include the usage frequency factor and functionality similarity factor. The usage frequency indicates the total number of visits to/from the BDI model and the environment. The functionality similarity indicates the similarity of a plugged item's functionality with the other plugged items' functionalities.

$$E = \left(\sum_{i=1}^{m} V_i \Big/ T \right) \times F$$

where E denotes the DSC usage efficiency; V_i denotes a visit to a plugged item; m denotes the last visit to the plugged item; T denotes the time period, which starts from the first visit to a plugged item and ends by the last visit to the plugged item, basic timing unit is day; F is functionality-similarity factor. F can be further expanded through decomposing the task description of a plugged item into several key terms and comparing these key terms with other plugged items' key terms, see Figure 5.3.

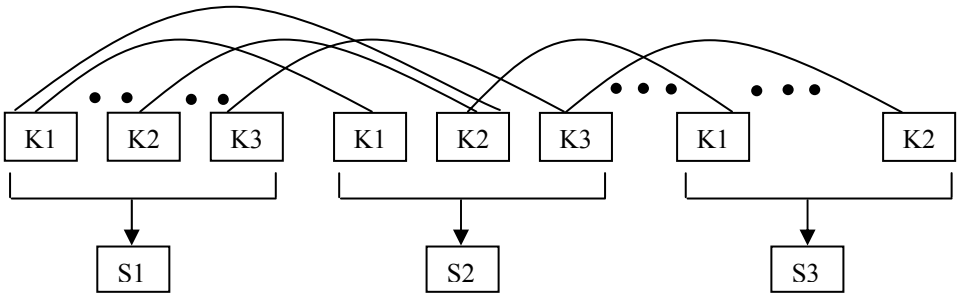


Figure 5.3. Functionality Similarity Calculation.

As shown in Figure 5.3, the functionality-similarity value of S1 and S2, namely F1, is calculated by exhausting all combinations of choosing any one key term from S1 and any one key term from S2. F2 represents the functionality-similarity value of S1 and S3. Therefore, the overall functionality-similarity value of S1 is the composition of F1 and F2.

$$F = \sum_{\substack{m,n \in C1 \\ m \neq n}} \sum_{i,j \in C2} S(L_m(K_i), L_n(K_j)) \quad , \qquad \text{(Eq.5.1)}$$

where *m* and *n* denote the DSC slot number (e.g. S1, S2, S3 in Figure 5.3), *m* must not equal to n because the key terms are only calculated with similarity to other slots but not the same slot; *i* and *j* denote the key term number (e.g. K1, K2, K3 in Figure 5.3); *S(x, y)* is a function to calculate functionality similarity between *x* and y; $L_x(K_y)$ denotes key term K_y in slot L_x; C_1 denotes the total slot number of the DSC; C_2 denotes the set of decomposed key terms. Therefore, the DSC usage efficiency can be further expanded as:

$$\begin{aligned} E &= \left(\sum_{i=1}^{m} V_i \Big/ T \right) \times F \\ &= \left(\sum_{i=1}^{m} V_i \Big/ T \right) \times \sum_{\substack{m,n \in C1 \\ m \neq n}} \sum_{i,j \in C2} S(L_m(K_i), L_n(K_j)) \end{aligned} \quad . \qquad \text{(Eq. 5.2)}$$

If a plugged DSC item has low *E* value compared with other items, and there is no slot for a new item then the low E item will be unplugged from the DSC and sent to the Matrix for reuse. Therefore, the design of the DSC can enhance the reusability and adaptability of an AOCD-based agent.

The Message Deliverer component in an AOCD agent is responsible for storing and delivering incoming/outgoing messages, which include external requests/tasks, internal requests, processing results, etc. The Message Deliverer component deploys a waiting queue to store all the messages and deals with these messages according to the First-In-First-Out (FIFO) principle.

The BDI model is a major component in an AOCD agent, which can be regarded as a central processing unit in agent design. Section 5.1.2 introduces the details of the BDI model design in AOCD.

The Travel Control Centre (TCC) enables an AOCD agent to travel to a remote site. This component is not essential for every AOCD agent as some agents do not travel or barely travel. Therefore, the TCC also deploys the plug-and-play design. Whenever an agent requests travel, a travel resource item will be plugged into the TCC. If an agent is barely travelling, the travel resource item will be removed from the TCC to reduce the memory consumption. Section 5.1.3 introduces the detailed design of TCC.

5.2.3. BDI Model in AOCD

The BDI (Belief-Desire-Intention) model is employed as the design architecture for decision-making and reasoning process in the AOCD framework. The origin of the BDI model was inspired by the theory of human practical reasoning developed by Bratman [21, 22]. Currently, this model is used widely in many organisational and industrial systems and it has been found efficient in some highly successful agent architectures to date [162]. The advantages of using the BDI model in agent designs can be summarised in several reasons. First, the BDI model provides a natural way of describing the agent reasoning process by using three mental states: beliefs, desires, and intentions. This three-states reasoning method proves to be useful in capturing human knowledge and reasoning for specific tasks [111], which is described as intuitive advantage [162]. Second, the BDI model simplifies the agent development process by giving a clear functional decomposition [162], which increases the flexibility and efficiency of updating agent functionalities. Finally, the BDI model exhibits extraordinary adaptability to the dynamic environment because the three bases (B, D, I) have a very low degree of dependency or are totally independent in many cases. A change to a base will not interfere with the other bases and a change of an item (e.g. rule, belief) within the base also will not interfere with the other items in the same base. The benefits of the BDI model are proven by its widespread implementations in agent softwares, such as dMARS, 3APL, JACK, JADEX, etc.

Certainly, the BDI model is not the only option for agent design; layered architecture also has been deployed in many agent-based applications, such as InteRRaP [107] & TouringMachines [46]. Layered architecture design decomposes an agent's behaviours and creates separate subsystems to deal with these different types

of behaviour. The Touring Machines architecture described by Ferguson [46] is a typical layered architecture design. The main problem with layered architecture is it lacks the conceptual and semantic clarity of un-layered approaches. This disadvantage makes the transformation from logic-based semantics to a layered architecture to be a very difficult process because many logic-based semantics cannot be separated completely. This weakness discourages the use of layered architecture in AOCD because the AOCD framework is a DSS based architecture that involves many logic-based semantics. The other two agent architectures suggested by Wooldridge seem not favourable in the AOCD framework because of their inherent disadvantages. The information source of the reactive agent architecture is limited to local information [162]. This situation becomes the major problem that prevents the adoption of reactive agent architecture in the AOCD framework, which is a multi-agent-based system relying heavily on information sharing from various sources. The logic-based agent architecture requires highly logical semantics [162], however, it is difficult in the AOCD framework, which often deals with many unstructured or semi-structured requests.

The BDI model is not without limitations. In some cases, the BDI model suffers from its incompetency in dealing with some descriptions that are un-decomposable and some constraints are vague. Moreover, the BDI model seems to have less efficiency in dealing with uncertainties in reasoning processes because it has less focus on probability calculations that might result in using a large amount of beliefs to express a simple probability calculation. Nevertheless, the BDI model is still the best option for agent-based applications as it has been used in some very successful agent architectures to date [60, 162]. This is the motivation for the implementing BDI model in the AOCD framework. Most importantly, the AOCD framework aims to solve problems in DSS, and the BDI model has excellent capabilities to clarify a decision-making process in a folk psychology way [111]. Therefore, the BDI model turned out to be the most efficient reasoning method for the AOCD framework.

In the AOCD framework, the BDI model is deployed as a central unit. Figure 5.4 shows the BDI model design in the AOCD framework. An AOCD agent connects to the Matrix or another agent through the Matrix-Agent connection/Agent-Agent interfaces. Once a connection is established, the agent is able to send messages to or receive messages from the environment through the agent-agent or agent-Matrix interfaces.

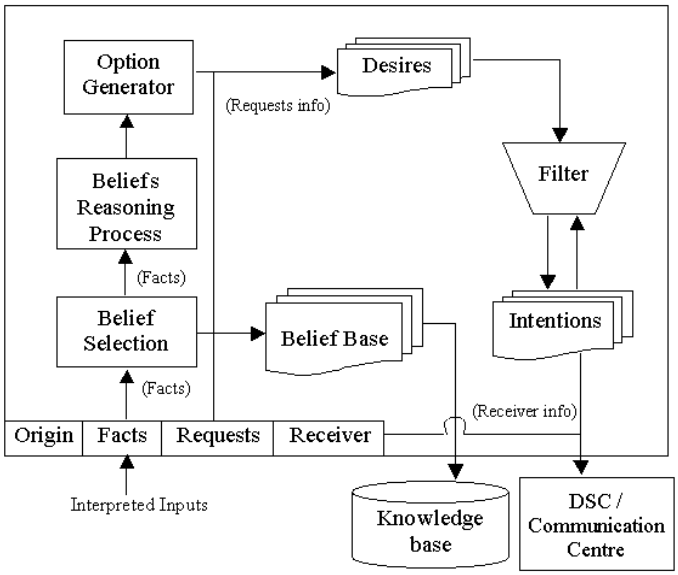


Figure 5.4. BDI Model in AOCD Agent Design.

The received messages are decomposed into several categories and forwarded to a Communication and Language Centre. The Communication and Language Centre translates the received information into some recognisable formats and sends the BDI for reasoning. The BDI component provides the capabilities of domain problem solving.

Inside the BDI component, the Beliefs Reasoning Process generates results by taking input facts into account and delivers the results to an Option Generator, which maps a set of beliefs and a set of intentions to a set of desires. The desires are sent to a Filter for an agent's deliberation process (deciding what to do). After the filtering process, an agent is able to generate one or more intentions, which will be sent back to the Communication and Language Centre or DSC.

If a final result is sent to the DSC then the agent can take action directly through triggering a plugged DSC item. The DSC is designed with this ability because some messages from the environment do not require a feedback. For example, a manager sends a message to a schedule agent demanding a printout of today's timetable. This message does not contain any complicated decision making process and is one-way. In this case, it is not necessary to send any feedback to the manager. Thus the DSC item that has printing capability would simply make a printout of the timetable instead of performing any further activities. If the final result is sent to the Communication and Language Centre, then the result will be translated to a recognisable format according to its receiver's standard. To further understand the process in the BDI model, the following scenario is provided. According to Wooldridge [162], the Option

Generator maps a set of beliefs and a set of intentions to a set of desires, which can be expressed logically as follows. We adopt the following notations: if A is an arbitrary set, then $\wp(A)$ is the powerset of A, *Bel* represents the rules in Knowledge Base, *Int* represents the Intention sets, and *Des* represents the desires sets. The options can be represented as a mapping as: *Options*: $\wp(Bel) \times \wp(Int) \rightarrow \wp(Des)$.

The Filter function performs a deliberation process that can be expressed as follows.

Filter: $\wp(Bel) \times \wp(Des) \times \wp(Int) \rightarrow \wp(Int)$.

The filter should satisfy the following constraint [162]:

$\forall B \in \wp(Bel), \forall D \in \wp(Des), \forall I \in \wp(Int),\ filter(B, D, I) \subseteq I \bigcup D.$

The following ambulance allocation case illustrates the BDI process in an AOCD agent. Once a request for an ambulance is received, then an AOCD agent starts to assess the request information. If the agent believes ambulance A is the closest ambulance to the accident site and A is available, then the agent sends A to the site. When A is on the way to the site, if the agent finds ambulance B (previously unavailable) is closer than A, and is available; then it sends B to the accident site and recalls the assignment from A. The agent has desire to send an ambulance *X* to the accident site *Y* as soon as possible if the following applies:

- The agent has belief that there is a valid request for an ambulance.
- The agent has belief that there are some ambulances available.
- The agent has belief that *X* is the closest ambulance to site *Y*.
- The agent has belief that there is a previous assignment to an ambulance.

The desire to send an ambulance *X* and recall a previous assignment occurs if the following applies:

- *X* is available and closest to an accident site.
- *X* is closest to an accident site and just returning to being available.
- Previous assignment is recalled.

The intention of the ambulance process is to find the closest ambulance to accident site Y. Therefore, the BDI process can be generated for the AOCD agent as follows:

BRF: Belief ($\exists$ *Request*, *Request* $\in$ T) $\wedge$ Belief ($\exists$ *Ambulance*, *Ambulance* $\subset$ A) $\longrightarrow$ Belief ($\exists$ *Request*, $\exists$ *Ambulance*)

Options: Belief ($\exists$ *Request*, $\exists$ *Ambulance*) $\wedge$ Belief ($\exists X \in Ambulance \subset$ A, D (*X*, *Y*) = S) $\wedge$ Intention (*Find_Closest_Ambulance*, *Site_Y)* $\longrightarrow$ Desire ($X \rightarrow Y$) $\wedge$ Desire (*Recall_assignment*)

Filter: Belief ($\exists$ *Request*, $\exists$ *Ambulance*) $\wedge$ Belief ($\exists$ K $\rightarrow$ *Y*, K $\in$ *Ambulance*) $\wedge$ Desire (*X* $\rightarrow$ *Y*) $\wedge$ Desire (*Recall_assignment*) $\wedge$ Intention (*Find_Closest_ Ambulance*, *Site_Y)* $\longrightarrow$ Intention (*Assign*, X $\rightarrow$ Y) $\wedge$ Intention (*Recall*, K $\rightarrow$ Y)

Execute: Intention (*Assign*, *X* $\rightarrow$ *Y*) $\wedge$ Intention (*Recall*, K $\rightarrow$ Y)

Action: $X \xrightarrow{assign} Y$; $K \xleftarrow{recall} Y$.

Where T denotes existing requests for ambulances; A denotes ambulances are available; D (*X*, *Y*) denotes the distance between *X* and *Y*; *S* denotes the shortest distance.

This case demonstrates the working process in an AOCD agent's BDI component. This illustration uses descriptive languages to provide readers with an easily understandable environment; the actual process in the BDI model is formulised and standardised. Based on this model, the beliefs, intentions, and filter constraints can be further expanded to construct a complete BDI-based AOCD agent.

5.2.4. Agent Mobility and Travel Control Centre (TCC)

Generally, an agent that possesses travel capabilities is called mobile agent, which can travel from the current environment to another. A mobile agent is often described as "an executing program that can migrate from machine to machine in a heterogeneous network" [66, 165]. In a heterogeneous agent network, mobile agents are classified as a major category of intelligent agents. The agent mobility is the capability of an agent that can migrate to other machines. A theoretical analysis of agent mobility based on different agent network topologies is discussed in Section 7.2. Each AOCD agent has capability to travel once an agent possesses a travel resource item in the AOCD framework. Hence, each AOCD agent can be a mobile agent. This is one of the major differences between the AOCD-based system and other agent-based systems. Agent mobility is important for the AOCD-based system because the cooperation between Matrices and agents performs frequently. The AOCD systems often require agents to travel to other Matrices to their participations in problem-solving and decision-making processes.

The core component in an AOCD agent for travelling is called the *Travel Control Centre* (TCC). A travel resource item is employed to guide an AOCD agent's travel in the TCC that consists of several plug-in slots and it can support more than one travel itinerary. The number of plug-in slots in TCC is extendable and it depends on the agent's requirements (See Figure 5.8), and each slot contains a travel itinerary.

Currently, several agent transfer technologies have been applied to mobile agent design, which include mobile code [150], remote objects [16], and agent cloning [131]. The design of TCC integrates existing mobile agent technologies with the Ticket pattern suggested by Aridor and Lange [7]. Agent cloning is still preliminary in the multi-agent area. However, it has been studied in several research works, e.g. [131].

Figure 5.5 shows the process of an AOCD agent travelling to a remote machine. The mobile agent travel process is actually the combination of the agent cloning process [131] and the agent code transferring process. In other words, the destination Matrix allocates a new agent structure for a travelling agent and receives the travel agent's contents including the agent capability registry, knowledge base, Domain Specific Components, and Travel Control Centre from the travelling agent. Then the Matrix distributes the received contents to the new agent structure to complete the travelling process. A new agent structure contains no specific DSC items.

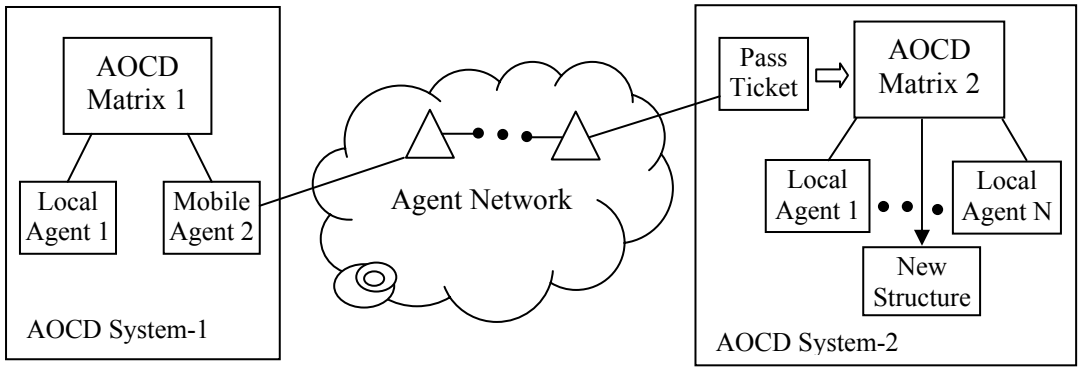


Figure 5.5. Mobile Agent Travelling in An Agent Network.

In Figure 5.5, mobile agent 2 in AOCD Matrix 1 has request to travel to AOCD Matrix 2 system through a connected agent network. There is no direct connection between Matrix 1 and Matrix 2, thus mobile agent 2 needs to travel through other nodes to reach its destination – Matrix 2. In the AOCD framework, a travelling agent often passes by other Matrices to reach its destination Matrix when the origin and destination are not connected. Therefore, finding the shortest path is a major issue for TCC (see Section 5.3.2). The transit Matrices do not clone agent bodies for the travelling agent, they only receive the agent contents and deliver to the next node.

Once the travel itinerary is resolved, mobile agent 2 can travel to Matrix 2. However, mobile agent 2 needs a pass ticket to enter Matrix 2. If mobile agent 2 is permitted to enter, a request will be sent to Matrix 2 to allocate an empty agent body for mobile agent 2. Meanwhile, Matrix 2 starts receiving mobile agent 2's contents

and distributes the contents to the empty agent body. The final stage of the travelling process is to verify the accuracy of the destination agent with the origin agent. Table 5.1 illustrates the detailed process.

The following algorithm illustrates the process of settling a travelling agent.

Algorithm: Settlement Process for Travelling Agent in Matrix

```
1.  if Ticket (AgentX) = False then
2.        {WaitingQueue (AgentX, Pass);}  // AgentX waits for a pass ticket.
3.  else {  // //the start of settling process.
4.        if CloneAgent (AgentX, AgentC, MatrixY) = False then
5.            {WaitingQueue (AgentX, Body);}  // AgentX waits for an empty body.
6.             // AgentX is original travelling agent; AgentC is cloned agent.
7.        else { i = 0;   // distributing agent contents.
8.              do while AgentX_Capabilities (i) <> Null
9.                 {AgentC_Capabilities (i) = AgentX_Capabilities (i);
10.                  i = i + 1;} // transferring agent capabilities.
11.             loop
12.             i = 0;
13.             do while AgentX_Knowledge_item (i) <> Null
14.                 {AgentC_Knowledge_item (i) = AgentX_Knowledge_item (i);
15.                  i = i + 1;}  // transferring agent knowledge base.
16.             loop
17.             i = 0;
18.             do while AgentX_DSC_item (i) <> Null
19.                  {AgentC_DSC_item (i) = AgentX_DSC_item (i);
20.                   i = i + 1;} // transferring agent's domain specialised components.
21.             loop
22.             i = 0;
23.             do while AgentX_TCC_item (i) <> Null
24.                 {AgentC_TCC_item (i) = AgentX_TCC_item (i);
25.                  i = i + 1;}  // transferring agent's travel control centre.
26.             loop
27.            }
28.            if Verifying (AgentC, AgentX) = False then
29.               { WaitingQueue (AgentX, Content);}
30.             // new agent's contents do not match original agent then re-do the transferring.
31.             else {
32.             // Updating Matrix's parameters.
33.             AgentNum = AgentNum + 1;  // the total number of plugged agent increased.
34.             MatrixAgent_InterfaceNum = MatrixAgent_InterfaceNum + 1;
35.             i = 0;
36.             do while AgentC_Capabilities (i) <> Null
37.                { if NewCapability (AgentC_Capabilities (i)) = True then
38.                     {add (AgentC_Capabilities (i), MatrixCapabilityRegistry);}
39.                  i = i + 1;}  // Updating Matrix's capability list.
40.    }   //the end of the settling process.
```

Take the organisation merge scenario described in Chapter 3 as the example to explain the agent travel process. Assume that the 'Accounting management' agent from Institute B plan to travel to Institute A. The accounting agent will calculate the travel path to the Institute A's Matrix first and store the path info to the TCC. Then the accounting agent needs to apply for a pass entry to the destination Matrix. If the pass is granted, then the accounting agent starts to transfer its capability information to the destination Matrix. The destination Matrix will allocate an agent body without content to store the information transferred from the accounting agent when it grant pass permission.

Table 5.1. Detailed Process of Transferring An Agent in AOCD.

TCC Steps	Actions
1. Generating request (Agent)	Travelling agents send request for travel.
2. Identifying travel route (Agent)	Travelling agents resolve the travel route.
3. Delivering contents (Agent)	Delivering agent contents to next node based on route.
4. Requesting pass ticket (Agent)	Travelling agents send requests for a pass ticket to enter.
5. Generating empty body (Matrix)	Matrix clones an empty agent body.
6. Receiving agent contents (Matrix)	Matrix receives agent contents.
7. Distributing agent contents (Matrix)	Matrix distributes received contents to the empty body.
8. Verifying agent contents (Matrix)	Matrix verifying the cloned agent with origin agent.

5.2.5. Agent Interface Design in AOCD

There are three different agent interfaces in the AOCD framework, which include agent-agent interfaces, agent-Matrix interfaces, and agent-user interfaces. Agent-agent and agent-Matrix interfaces are invisible to system users. In other words, these interfaces do not directly interact with human users. On the contrary, agent-user interfaces focus on interactions between users and agents.

The agent-agent and agent-Matrix interface designs in the AOCD framework mainly focus on the efficient way of establishing connections between agents or between an agent and a Matrix. The difference between the agent-agent connection and the agent-Matrix connection is that the agent-agent connection is established temporarily whereas the agent-Matrix connection is established for an agent to reside in the Matrix. Thus, the agent-agent interfaces in the AOCD framework need to have a certain degree of flexibility. In the AOCD framework, we adopt a request-based connection for the agent-agent interface. The agent-agent interface deploys a request receiver, which stores the requests from other agents or the Matrix as shown in Figure

5.6. Once a request is received, the host agent starts to process the request from the requesting agent and the agent interface will indicate the information of the current requesting agent.

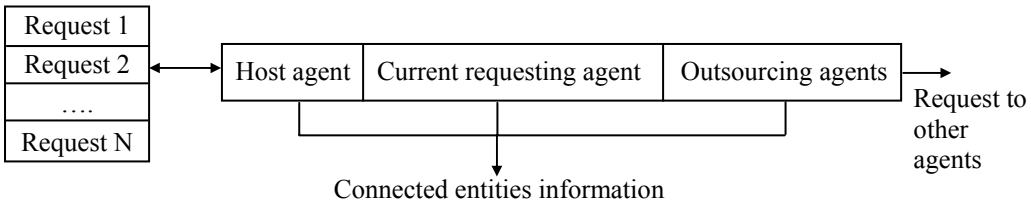


Figure 5.6. Agent-agent Interface Design.

This agent-agent interface makes use of a queue to allow multiple requests for connections, however only one connection will be established in one request-solving process. After the completion of a request-solving process, the first request in the queue will be processed, and the rest of the requests follow this procedure.

The agent-Matrix interface involves the agent capability registration, the dynamic detection for interface allocation, and the participants' information (including all the participants in solving a request). The detailed design can be found in Chapter 4.

Unlike the agent-agent and agent-Matrix interfaces that are based on a system component design point of view, the agent-user interface is more from a computer-human interaction perspective. The agent interface design of the AOCD-based agents follows two major guidelines.

First, the agent-user interface design ought to dynamically adjust to the situation and present a user-friendly interface to facilitate the efficient collaborations between the human and agents. Related work can be found in the Assistant-Design of Interaction Environments for Users (ADIEU) system [121]. Second, the interface design in the AOCD framework needs to implement the appropriate group awareness mechanisms. Most of the current multi-agent systems are distributed and reply on agent cooperation to solve a complex problem. In many cases, a multi-agent system requires human participation when a problem cannot be solved by agents. Consequently, the needs for applying group awareness and self-awareness mechanisms in software agent interface design are rising. In the AOCD framework, the agent-user interface provides the information on the portion of a task completed. This information has been proved important in collaborative authoring [123] and it certainly has the same impact on multi-agent systems as both of them are based on collaborative work. The agent-user interface in AOCD also provides other

information, such as agents' expertise (agent general capability) and location, cooperation history between requesting agents and service-provider agents, and availability. The more information about the agent design in groupware can be found in [39], in which a number of groupware applications for agents are discussed, and also indicates the future directions for agents in groupware.

5.3. Agent Communication Methodologies

As current organisational and enterprise systems are becoming larger and more complex, it is not realistic to build a single agent to deal with all the information processes of an organisation. An agent-based system normally consists of numerous individual agents, which are responsible for their specific domains. Therefore, a complex problem can be solved through agent cooperation and coordination. For instance, in a multi-agent system, each agent possesses a certain number of skills (capabilities) that can solve some problems in specific domains. In many cases, an individual agent is insufficient to solve a request that overlaps several different domains, thus several agents are required to solve the request through negotiation and cooperation.

Existing agent cooperation methodologies have focused on several areas including agent matching and capability description [15, 137, 138], agent planning and negotiation [49, 50, 82, 128, 141], agent task decomposition [37, 81, 90], and agent communication language [52, 93, 103]. Among various agent cooperation and communication methodologies, agent communication language is essential for agent cooperation. All the communications between agents are based on agent communication languages. The next section introduces the design of agent communication language in the AOCD framework.

5.3.1. AOCD Agent Communication Language

Various agent communication languages have been suggested in previous works, such as Knowledge Query Manipulation Language (KQML) and Foundation for Intelligent Physical Agent - Agent Communication Language (FIPA ACL). The AOCD Agent Communication Language (AOCD ACL) basically adopts the FIPA ACL message structure [52] since the FIPA ACL is more powerful with composing new primitives. Figure 5.7 gives the message structure of FIPA ACL.

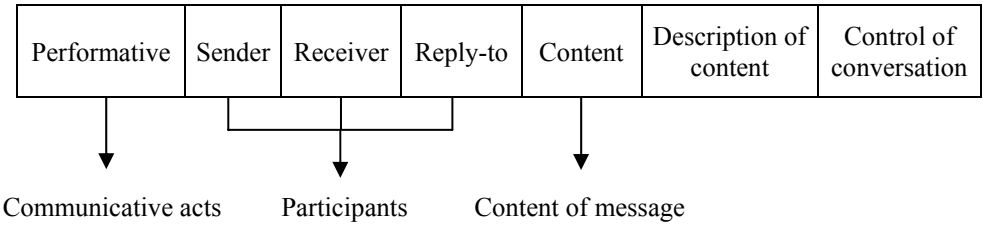


Figure 5.7. FIPA ACL Structure [52].

The Primitive parameter indicates those atomic actions that are not composed by more than one communicative act and it is the only mandatory parameter in FIPA ACL though most of the FIPA ACL messages contain Participants and Content parameters.

The content of a message contains the information of whatever the communicative act applies to. The meaning of the content of any ACL message is interpreted by the receiver of the message. All of the FIPA message parameters share the frame and ontology. The domain of discourse of the ontology in FIPA ACL includes their parameters described in the FIPA ACL structure. Table 5.2 illustrates the parameters in FIPA ACL and their descriptions.

Table 5.2. FIPA ACL Message Parameters [52].

Parameter	**Category of Parameters**
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

The major difference between AOCD ACL and FIPA ACL is that: the AOCD ACL strengthens the parameters of participants by indicating the general service domain of the participants. This design improves the efficiency and accuracy for the Communication and Language Center of an AOCD agent to convert the AOCD ACL

to knowledge-based descriptions through narrowing down the service domain in the matching and decomposition processes.

A typical description of the participants in AOCD ACL is shown below:

Parameter	Description
Sender / Receiver	Denotes the identity and specialty of the sender of the message, that is, the name and the context of the entity of the communicative act and its general service domain.

For instance, a sender description can be ": sender (entity-identifier :name [Matrix1-agent1] :domain [sales management])". The description in brackets behind ':domain' indicates the agent's general service domain. This description can be further broken down into details by adding more specific descriptions after a dash, like ": sender (entity-identifier :name [Matrix1-agent1] :domain [sales management – sales statistics & sales policies])". The domain section is allowed to be null, which indicates the sender of the message has general capabilities and this often happens when the sender of a message is a Matrix.

Another difference between AOCD ACL and FIPA ACL also can be observed from the above example: that is, AOCD ACL indicates the Matrix-agent relationship in the participant performative. In the AOCD framework, agent communication language is not only used between agents, it has been extended to the communications between agents and Matrices. In FIPA ACL, a message only states the participant's name, whereas an AOCD ACL message needs to describe the Matrix-agent relationship through using a dash line to connect the both parts. The description of two parts indicates the agent identity and the Matrix identity that the agent resides in, such as Matrix1-agent1 means the sender is agent1, which is residing in Matrix1. If the sender of a message is a Matrix then the sender description can be ": sender (Matrix-identifier :name [1] :domain [management])" . If the sender of a message is an agent and it does not reside in any Matrices then the sender description can be ": sender (agent-identifier :name [null-2] :domain [logistic management]". The 'entity-identifier' indicates whether a message is from a Matrix or from an agent. If the 'entity-identifier' section is an agent-identifier then the Matrix name needs to be identified before identifying the agent name by using a dash line to connect them, such as ": sender (agent-identifier :name [1-2] :domain [logistic management]". This parameter indicates that the message sender is agent 2, which resides in Matrix 1 and agent 2 specialises in logistic management.

An example AOCD ACL is illustrated as below:

```
( request
    : sender (Matrix-identifier :name [1] :domain [])
    : receiver (set (agent-identifier :name [2-11] :domain [sales statistics] ) )
    : content
       " ( ( action (agent-identifier :name [2-11] :domain [sales statistics] )
         ( calculate book-sales-2006 ) ) ) "
    : protocol AOCD-request
    : language AOCD-acl
    : reply-with  transaction169 )

( inform
     : sender (agent-identifier :name [2-11] :domain [sales statistics])
     : receiver (Matrix-identifier :name [1] :domain [])
     : content
        " book-sales (2006, 36590112) "
     : language AOCD-acl )
```

In this example, the Matrix 1 sends a request to agent 11 (resides in Matrix 2 specialise in sales statistics) to obtain the book sales amount in 2006. The results also will be recorded in transaction number 169. Agent 11 produces the results and sends them back to Matrix 1, which indicates the book sales in 2006 are 36590112.

5.3.2. Topological Description Language for Agent Networks

Each agent supports the *Topological Description Language for Agent networks* (*TDLA*). The detailed information of the TDLA is described in Chapter 7. In this section, we discuss how this function is implemented in the AOCD agent design.

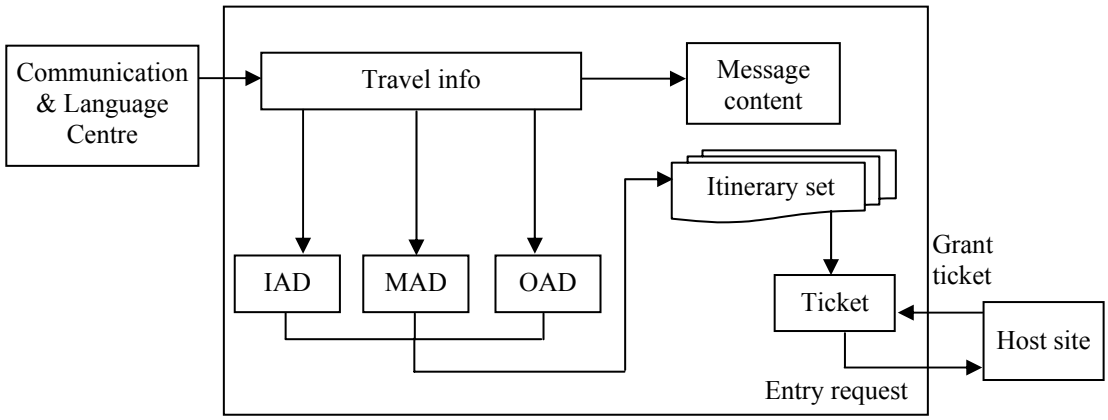


Figure 5.8. A Schematic View of the TCC Structure.

In the AOCD framework, each agent carries a set of identifiable TDLA descriptions. The TDLA descriptions contain information including individual agent description, (IAD), main agent-groups description (MAD), and overall agent-network description (OAD) [171]. These descriptions are stored in the Travel Control Centre (TCC) component. Each travelling agent needs to carry these descriptions before

travelling. In a TCC, an Itinerary set component contains the information about a travelling agent's routes. A travelling agent may transit at several temporary sites before reaching to its destination; the Itinerary set contains the information on all the routes and each route is stored in an itinerary. The choice of the travelling route is based on various factors, which include the network traffic factor, the network topology factor, and the connectivity between origin and destination. The IAD, MAD, and OAD determine the network topology factor. The Communication and Language Centre (CLC) is the source of providing travel information. The BDI model also participates in TCC; it often determines whether an agent needs to travel based on the BDI reasoning results. In some cases, agents can determine whether they need to travel based on the information from the CLC. Figure 5.8 shows the TCC structure.

The first step in the travelling process is obtaining and decomposing the travel information. The original travel information consists of a destination site name (if it is unique) or a specific address, the travel mode (one-way, round trip), message content, and communication language.

The second step of the travelling process is applying the IAD, MAD, and OAD to the itinerary selection process and generating the travel routes for the itinerary set.

The final step of the travelling process is: an agent travels according to the itinerary set, which is based on the First-In-First-Out (FIFO) principle. In other words, if there are several routes in the itinerary set, the agent will travel based on the first route in the set. When the agent enters the destination site then the first route in the set will be removed and the second first itinerary becomes the first route, as shown in Figure 5.9. To enter into the destination site the agent needs to have a ticket, so the agent sends an entry request to the host site and it can enter the host site once it gets an entry ticket, then the first itinerary is complete. If the itinerary set is empty then the agent has reached the final destination.

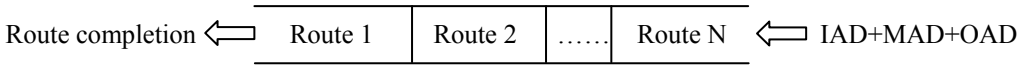


Figure 5.9. FIFO Principle in Itinerary Set in TCC.

The following example illustrates the travelling process of an AOCD agent, namely, Agent M. Agent M received travel information from the CLC, which contains the following information. We assume Agent M resides in Matrix 6 and they share same IP address that is: 138.77.37.10.

Destination address: IP address: 138.77.203.15. Host name: Matrix 11.

Travel mode: Round trip.

Message content: ((action (agent-identifier :name [2-11] :domain [sales statistics])
(calculate book-sales-2006)))

Language: AOCD ACL.

- The first step is: decomposing the received information into two major sections. The first section contains the destination address and travel mode. If the travel mode is single trip then the travel tag = 1. If the travel mode is a round trip then the travel tag = 2 and there are two destination addresses and once an agent reaches its destination then the travel tag will be deducted by 1 (until it equals to 0). The first destination address is the destination address in the received information and the second destination address is the current address. In this example, the destination addresses are:

Travel Tag = 2	D1 = 138.77.203.15, Name = Matrix 11	D2 = 138.77.37.10, Name = Matrix 6

The second section contains the message content and communication language. In this example, the information in this section is:

((action (agent-identifier :name [2-11] :domain [sales statistics]) (calculate book-sales-2006)))	AOCD ACL

- The next step is: incorporating destination addresses with IAD, MAD, and OAD to generate the itinerary set. The detailed descriptions of IAD, MAD, and OAD are provided in Chapter 7. First, the TCC uses IAD to identify Agent M's current position and connectivity. The attachment probability is identified by the destination addresses. We assume the IAD, MAD, and OAD are provided as follows:

IAD – Agent M:
Degree of links: 3 (138.77.37.1; 138.77.37.15; 138.77.38.20)
Local address: 138.77.37.10
Extensibility: *MC – DL* = 7 (*MC*: Maximum Connections, *DL*: Degree of Links)

MAD – Group 1:
Total number of nodes: 8 (138.77.37.1; 138.77.37.10; 138.77.37.12; 138.77.37.15; 138.77.37.31; 138.77.38.11; 138.77.38.20; 138.77.38.27)
Total number of links in the group: 7
Main group selection criteria: geography-based in Melbourne/Australia
Root nomination: 138.77.37.10
Context: major connection with the group that has nominated root 138.77.201.1

MAD – Group 2:
Total number of nodes: 3 (138.77.201.1; 138.77.201.18; 138.77.201.22;)
Total number of links in the group: 2
Main group selection criteria: geography-based in Brisbane/Australia
Root nomination: 138.77.201.1
Context: major connection with the group that has nominated root 138.77.201.1

MAD – Group 3:
Total number of nodes: 4 (138.77.203.1; 138.77.203.7; 138.77.203.12; 138.77.203.15;)
Total number of links in the group: 7
Main group selection criteria: geography-based in Sydney/Australia
Root nomination: Null
Context: major connection with the group that has nominated root 138.77.201.1

OAD:
Diameter of the network: 9
Total number of nodes: 15
Main agent group info: Total group number: 3
Group 1: (Root nomination: 138.77.37.10)
(Group selection criteria: based in Melbourne)
Group 2: (Root nomination: 138.77.201.1)
(Group selection criteria: geography-based in Brisbane)
Group 3: (Root nomination: no root)
(Group selection criteria: geography-based in Sydney)

Therefore, the detailed information on the origin and destination can be derived from IAD, MAD, and OAD. A Set Intersection mechanism is deployed to determine whether there is connectivity between an origin site and a destination site. The following proposition describes the Set Intersection mechanism.

$$(\forall N_i \in G_1) \wedge (\forall N_j \in G_2) \wedge \exists L(N_i, N_j) \Rightarrow C(G_1, G_2) \qquad \text{(Eq.5.3)}$$

where $L(N_i, N_j)$ denotes a link between nodes N_i and N_j. G_1 denotes agent Group 1 and G_2 denotes agent Group 2. $C(G_1, G_2)$ denotes that sets G_1 and G_2 are connective. Eq.3 can be further expanded when there are more agent groups involved.

$$\exists(C(G_a, G_b)) \wedge \exists(C(G_b, G_c)) \Rightarrow C(G_a, G_c) \qquad \text{(Eq.5.4)}$$

In this way, a travelling agent can determine whether it is possible to travel between the origin site and destination site. A travelling agent can only travel if its origin site and its destination site belong to two connective groups. In this example, agent M's origin site belongs to group 1 and the destination site belongs to group 3.

Figure 5.10 illustrates the network structure instead of providing every IAD of each agent in the network. The underlined nodes are the nominated roots in their groups.

Let the origin node (IP: 138.77.37.10) as N_1; the destination node (IP: 138.77.203.15) as N_3. According to Eq. 3 and 4, we have $(N_1 \in G_1) \wedge (N_3 \in G_3)$ but there is no link between G_1 and G_3. However, according to OAD information there is another group G_2 that can be used. When G_2 is taken into account, then we can have:

$(138.77.38.20 \in G_1) \wedge (138.77.201.18 \in G_2) \wedge \exists L\,(138.77.38.20,\ 138.77.201.18) \Rightarrow C(G_1,G_2)$

$(138.77.201.22 \in G_2) \wedge (138.77.203.1 \in G_3) \wedge \exists L\,(138.77.201.22,\ 138.77.203.1) \Rightarrow C(G_2, G_3)$

$$\exists(C(G_1,G_2)) \wedge \exists(C(G_2,G_3)) \Rightarrow C(G_1,G_3)$$

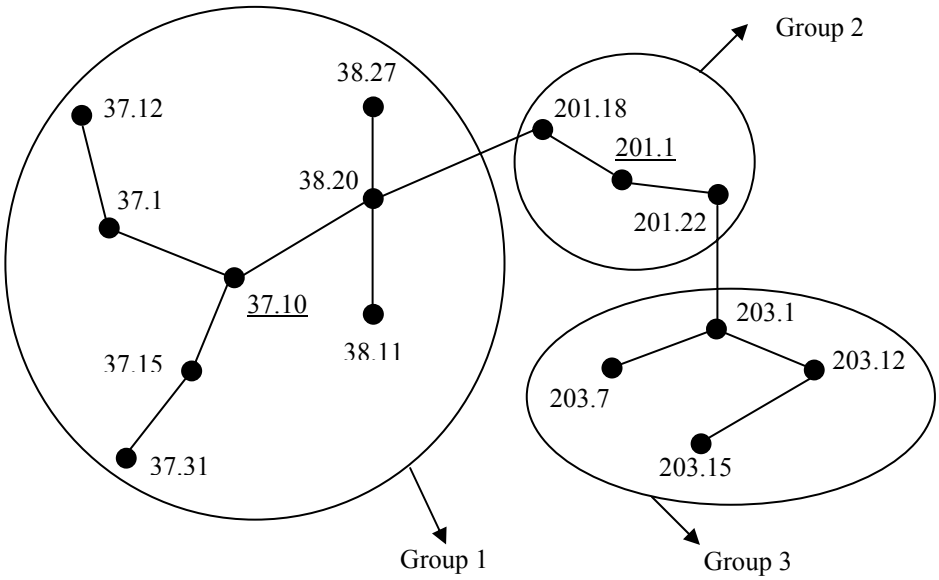


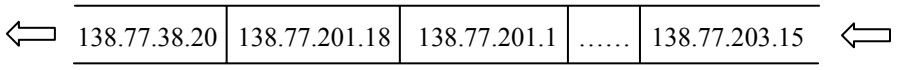
Figure 5.10. Agent Network Diagram based on IAD, MAD, and OAD.

Based on the above calculation, Group 1 and Group 3 are connective. Therefore, the travel path for agent M from N_1 to N_3 can be established through Group 2. The selection of the itinerary for agent M from N_1 to N_3 is based on all the possible itineraries from N_1 to N_3, which can be generated by exhausting all combinations of the nodes from N_1 to N_3. The selected itinerary should have the least path number. In this example, there is only one itinerary that can be generated that is:

138.77.37.10 (origin) $\rightarrow$ 138.77.38.20 $\rightarrow$ 138.77.201.18 $\rightarrow$ 138.77.201.1 $\rightarrow$ 138.77.201.22 $\rightarrow$ 138.77.203.1 $\rightarrow$ 138.77.203.12 $\rightarrow$ 138.77.203.15 (destination).

In many cases, there is more than one itinerary that can be generated for a travel agent. The selection of an appropriate itinerary is based on three major factors. First, a selected itinerary must contain the least paths among the generated itineraries. Second, a selected itinerary should be efficient in terms of network traffic and this can be determined by the network transaction speed. The last one is the site availability, which indicates the availabilities of all sites included in the selected itinerary. The other factors also might affect agent travel, such as location, extensibility, etc.

- The final step is: storing the selected itinerary in the itinerary set and agent M travels according to the itinerary set.



Once the first site address (route) in the itinerary set is complete, then this route will be removed from the itinerary set. If the itinerary set is empty then the agent completes one round of travel. Meanwhile, the travel tag will be deducted by one and the first destination address will be removed from the destination section message as shown in Figure 5.11. If the travel tag is zero then the travel is complete, otherwise it continues according to the destination information in the destination section.

Travel Tag = 1	D2 = 138.77.37.10, Name = Matrix 6

Figure 5.11. Destination Section Message after Completing One Round Travel.

The above example illustrates the process of agent M travelling from N_1 to N_3 through using IAD, MAD, and OAD. The OAD information is used to determine the number of involved agent groups specially when the origin agent group and destination agent group are not directly connected. The MAD information indicates all the agents in a group. The MAD information is useful to determine which group that a specific node belongs to. The IAD information describes the specific links between nodes, which is helpful to determine whether two groups have a connection or (transitive connection).

5.4. Agent Capability Descriptions for Cooperation

Agent Capability Descriptions (ACD) are essential for agent cooperation and task allocation since they describe what services an agent can perform. The methodologies for ACD in the AOCD framework are based on existing methodologies, such as *EXPECT* [61], *LARKS* [137], Interface Communication Languages (ICL) [101] etc. In

[112], the major characteristics of a capability description language are summarised as follows:

- The ability to declare what action a service performs.
- The ability to allow a capability to have different sets of inputs.
- The ability to declare preconditions and effects in some named rule definition language.
- The ability to refer to ontological descriptions of the terms used in the description and thus place the use of the terms in context.
- The ability to make the description explicit in the domain or context that the service operates.
- The ability to classify capabilities based on aspects of the description enabling exact or partial matches between required and provided capability descriptions.

LARKS is a language to advertise agent capabilities for agent matching. A LARKS message contains the structured description of the context, types, input, output, and constraints of an agent [137]. The structure of agent capability description in the AOCD framework is based on LARKS. The next section introduces the agent capability description language in the AOCD framework.

5.4.1. The Structure of ACD Language in AOCD

The AOCD agent capability description language incorporates the LARKS and *ontology fragments* [110] methods. The LARKS offers a simple and efficient solution for semantic matching. It is a desirable method for the AOCD framework because AOCD-based systems require quick and less complex semantic matching process as they adopt the agent-rank algorithm to narrow down the matching objectives (see Chapter 6). The ontology fragments method can efficiently support the AOCD framework as it offers three levels of descriptions: the general capability description can be applied to the Matrix, the middle level description can be applied to agents, and the bottom level descriptions can be applied to the DSC items.

We suggest a top-down folding tree to classify agent capabilities in the AOCD (see section 5.3.2). An agent's capabilities are classified into different levels, and the bottom level descriptions (the most specific descriptions) in the agent capability description tree can be further expanded by a set of ontology fragments. Figure 5.12 shows the structure of the AOCD agent capability descriptions based on capability classification.

Table 5.3. Agent Capability Description Language in AOCD.

```
Service Name: Book Sales
Capability list: {Y University book sales statistics, Y University book price index}
Ontology fragments <Y University book sales statistics>:
  {Fragment 1: Book sales by subjects
   Inputs: subject-name | String; (or) subject-code | Integer; (or) ISBN No | Integer
   Outputs: Sale-amount | Double
   InConstraints: Digits (subject-code) = 6; Digits (ISBN No) = 10;
   OutConstraints: Currency (99999999999999.99)
   Fragment 2: Book sales distributor information
   Inputs: distributor-name | String; (or) distributor-code | Integer; (and) location | String
   Outputs: Distributor-info | String
   InConstraints: 2 < Length (distributor-name) <= 15; Digits (distributor-code) = 6
                  2 < Length (location) <=15
    ……
   Fragment N: Book sales by year
   Inputs: Year | Integer;
   Outputs: Sale-amount | Double
   InConstraints: Digits (Year) = 4
   OutConstraints: Currency (9999999999999.99)}
 Ontology fragment <Y University book price index>
  {Fragment 1: Price index based on book
   Inputs: book-name | String; (or) ISBN No | Integer
   Outputs: price | Double
   InConstraints: 2 < Length (book-name) <= 20; Digits (ISBN No) = 10;
   OutConstraints: Currency (999999999.99)
   …….
   Fragment M: Forthcoming books preview
   Inputs: preview-category | String
   Outputs: book-name | String; price | Double; Author | String; Publisher | String
   InConstraints: Length (preview-category) = 6
   OutConstraints: book-name;}
```

The overall AOCD agent capability description language contains service name, capability list, ontology fragments, inputs, outputs, InConstraints, and OutConstraints. An example of the AOCD agent capability description language is described in Table 5.3. The service name indicates the general capability of an agent; the capability list indicates the agent capability in specialised domains; ontology fragments represent a specific task that an agent performs and it adopts plug-and-play concept; inputs

specify the input content of an ontology fragment for computation or reasoning; outputs specify the output content of an ontology fragment generated according to the inputs; InConstraints and OutConstraints indicate the constraints for inputting and outputting data.

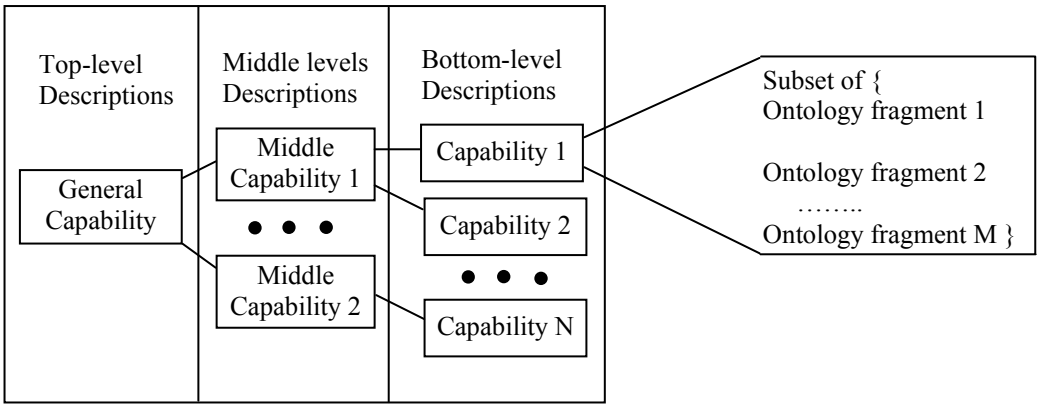


Figure 5.12. Structure of the Multi-level Agent Capability Descriptions.

5.4.2. A Top-down Tree for Classifying Agent Capabilities in AOCD

In the AOCD framework, a top-down folding tree is suggested to describe agent capabilities, especially to classify capabilities based on aspects of the description for matches between required and provided capability descriptions [112]. The classified agent capability descriptions can be converted to the AOCD agent capability description language (see section 5.3.1) for advertisement and matching within a multi-agent system.

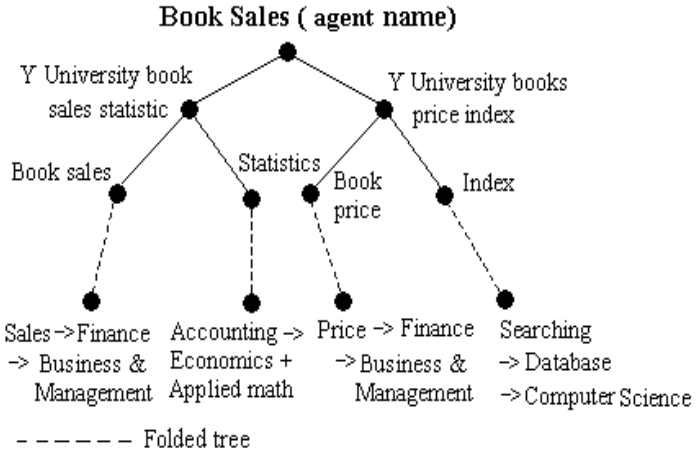


Figure 5.13. An Example of A Top-down Folding Tree.

In a top-down folding tree, the root of the tree describes an agent's capability set and each successive level parent node represents the capability of an agent, where parent nodes represent the sub-domains to which the child nodes belong. The leaf

nodes define the general categories to which the agent's capability belongs. The middle level nodes and leaf nodes in a description tree can be folded, which enhances the efficiency in describing an agent's capabilities by omitting irrelevant or unimportant information. The category classification of the fields of knowledge of science, social science and arts, and humanities adopts various standards such as [62]. Figure 5.13 gives an example of a top-down folding tree.

The folding process may occur when a parent node only has one child node. These parent and child nodes can be folded as one node and their descriptions are joined together by using a separator '→'. If there are more than two levels of these nodes, they all can be folded as one node and their descriptions joined together. Moreover, the bottom two levels of nodes can always be folded. If there is more than one child node attached to a parent node in the last two levels of the tree, then fold them and join the nodes' descriptions by using '+' to separate the child leaves (see Figure 5.13, under 'Statistics').

5.4.3. Agent Roles in Agent Capability Descriptions

The term of 'role' has been utilized in many agent-based systems for modelling communication, coordination, and organisational structure [33, 164]. The role description in a multi-agent system is mainly to clarify an agent's responsibility and capabilities [33]. In the AOCD framework, an agent's role can be described by an abstract level of agent capability descriptions. Generally, the description of the root node in an AOCD-based agent capability description tree can be regarded as the role of the agent.

5.5. AOCD Agent Development Life Cycle

The rapid development of intelligent agent technology has impacted existing methodologies of Software Development Life Cycle (SDLC). Traditional information systems mainly involve seven development phases include the requirement phase, the specification phase, the design phase, the implementation phase, the integration phase, the maintenance phase, and the retirement phase. Multi-agent-based systems completely shift current system development methodologies. It offers a Plug-and-Play method, which allows external components (agents) to be integrated to an existing system without disturbing the system structure and working process. We believe

multi-agent-based systems will eventually replace current system development model. Objects or components in current applications will be replaced by intelligent agents, which is more powerful, intelligent, flexible, and proactive.

The agent development life cycle in the AOCD framework consists six major stages as shown in Figure 5.14.

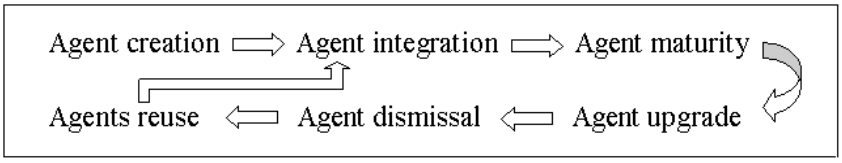


Figure 5.14. Agent Life Cycle in Agent Society.

At the *agent creation* stage, agents can be either designed by the system developers or reused by the agent society. At the *agent integration* stage, agents establish connections (relationships) with the Matrix. At the *agent maturity* stage, agents acquire knowledge from the Matrix and other agents, and become mature. At the *agent upgrade* stage, when agents' new acquired knowledge reaches a certain level, they need to upgrade their knowledge base and their functionalities accordingly. At the *agent dismissal* stage, agents are dismissed from the system through removing their connections with the Matrix and erasing their registered information from the Matrix register. At the *agent reuse* stage, the dismissed agent (not permanently deleted from the system) can be reused through reconnecting to the Matrix. These six stages are not compulsory to every agent society except agent birth and agent integration stages, which must exist in any AOCD-based agent society.

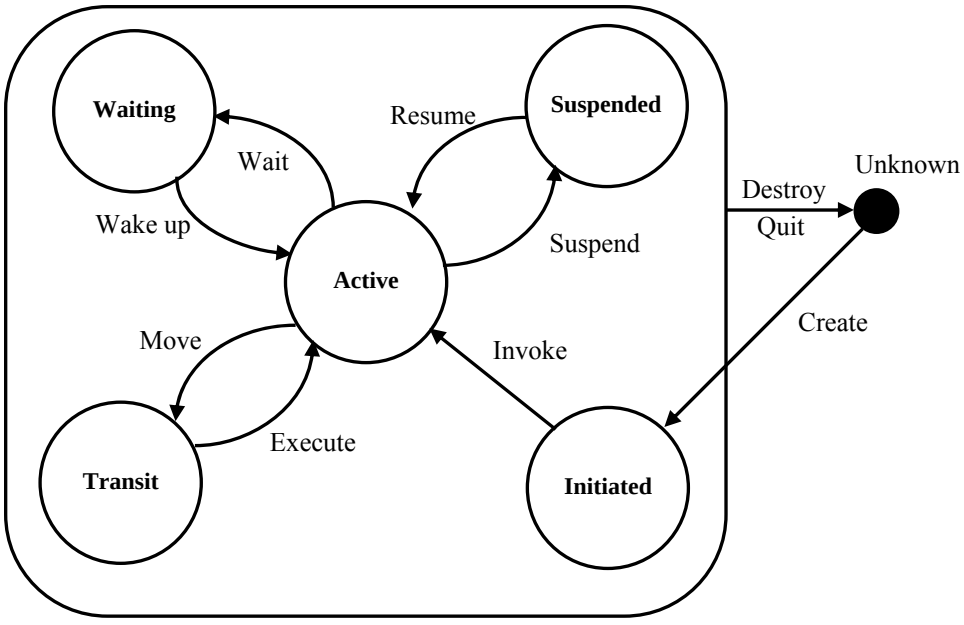


Figure 5.15. Agent Life-cycle Defined by FIPA [12].

Different from the agent life cycle defined by FIPA [12] (see Figure 5.15), the AOCD agent life-cycle integrates the agent-upgrade and agent-reuse stages. This variation makes an AOCD agent's life cycle more consistent and reusable than other agent-based systems.

Based on the AOCD agent life-cycle, the AOCD agent development processes have some unique features and they are very different from the classical SDLC. The classical SDLC process has been successfully applied to many software design processes. The most common models used in traditional SDLC include the waterfall model and incremental model. One of the major strengths of these common SDLC models is the sequential and highly structured design process. However, this strength has become a weakness in developing a DSS [56]. The reason for the incompetency of applying traditional SDLC in DSS development that has been indicated in [56, 100] is that: the development of a DSS involves various semi-structured problems and these problems cause the difficulty to identify the structures or nature of the problems prior to the initiation of the design phase, which a SDLC requires.

To tackle the above problem, the AOCD agent development adopts a three-stage development process, which has similarity with the DSS-based development life cycle and also overlaps the agent-based development life cycle. Notably, the development of an agent in an AOCD system is more efficient than developing a subsystem in a traditional DSS. The reason for this is the development of an AOCD agent is based on an individual agent's functionalities. In other words, the agents in the AOCD framework are developed independently. Therefore the process of analysing compatibility and adoptability for the overall system can be minimized in the AOCD system. On the contrary, the development of a DSS subsystem needs to analyse the overall system design structure, which increases the complexity of system design.

Existing methodologies for the agent-based development process have been various, such as agent-oriented analysis and design [25], agent modelling technique for systems of BDI agents [87], Multi-Agent Scenario-Based Method (MASB) [106], and Prometheus Methodology [116]. Among the various methodologies, Prometheus methodology is likely to be the best solution for the AOCD framework. This methodology emphasises building agents using BDI platforms, which is compatible with the AOCD's agent design. The Prometheus methodology is not the only solution to AOCD; MASB is another useful method for multi-agent system in the field of cooperative work. In the AOCD framework, we adopt the agent design phases in the

Prometheus methodology as the major guideline in building the AOCD agents and we exploit MASB method for constructing the cooperative framework in an AOCD system including the Matrix-agent connection development.

Table 5.4 AOCD Agent Development Life Cycle.

Agent Specification Phase	Architectural Design Phase	Detailed Design Phase
Use cases	Functionality description	Kernel Function Coding
Action and precepts	Knowledge base analysis	Developing Rule-base
Interaction Diagrams	Data flow analysis	Developing Database
	Event analysis	Developing Interpreter
	Interface analysis	Constructing AOCD ACL
		Developing Filter, Plan, Action

The AOCD agent development life cycle consists of three phases: the agent specification phase, the architectural design phase, and the detailed design phase. Table 5.4 shows the design phases and the design details in each phase in the AOCD agent development life cycle, which is evolved from Padgham and Winikoff's methodology [116].

The agent specification phase identifies the basic actions and the role in the system, along with inputs and outputs and any important shared data sources. The interaction diagrams show the relationships between the agents and the Matrix. The architectural design phase uses the output from the agent specification phase to analyse *(i)* knowledge base, *(ii)* events in environment, *(iii)* data flow, and *(iv)* relationship between agents. The detailed design phase is based on the architecture design phase, which emphasises coding and implementation issues.

5.6. Summary

This chapter illustrates the agent design methodologies in the AOCD framework, which include agent architectural design, agent communication methodologies, agent capability descriptions, and agent development life cycle. The design methodologies used in each section comply with the major agent characteristics.

The AOCD-based agents deploy a general agent pattern for structural design, which provides a standard design structure for every AOCD-based agent. This general agent pattern eliminates the problems of inconsistency and incompatibility in a heterogeneous agent system. The use of the DSC methodology in the general agent

pattern enhances the adaptability, flexibility, and reusability of the AOCD framework. It offers numerous slots for containing the domain components, which has special capabilities. The BDI model is the core component for the decision-making and reasoning process in the AOCD framework. The BDI model exhibits superior adaptability to the dynamic environment because the three bases (B, D, I) have a very low degree of dependency or are totally independent in many cases. The TCC component provides the AOCD agents with travelling capabilities over a multi-agent network based on TDLA. Each agent carries a TCC component, which consists of numerous slots for storing travel itineraries.

The Agent communication methodologies section introduces the AOCD agent communication language and the topological description language for agent networks. The AOCD ACL adopts the structure of FIPA ACL and strengthens the parameters of participants by indicating the general service domain of the participants. The AOCD ACL also indicates the Matrix-agent relationship in the participant performative.

The AOCD agent development life cycle consists of three major stages, which include the agent specification phase, the architectural design phase, and the detailed design phase. The stages in the AOCD agent development are related and have loose dependency that allows a flexible and efficient design process for DSS.

Overall, the AOCD framework employs several novel design methodologies. However, these new methodologies for AOCD agent development are not without limitations, such as the decomposition problems for unstructured descriptions in BDI, the bottleneck issues in TDLA when a destination site is overloaded, the difficulties in standardisation of agent capability descriptions for different applications, etc. Nevertheless, these problems do not hinder the development of the AOCD framework because the agent-based DSS is becoming a future direction in the DSS area.

CHAPTER 6
AGENT MATCHING MECHANISMS

An agent-matching process enables service matchmaking among heterogeneous agents in a multi-agent framework. Agent cooperation can be achieved through matching requesting agents with service-provider agents, and through such cooperation, multi-agents can solve a variety of complex problems. Improving the efficiency of the agent matching process is an important issue in the AOCD framework since multi-agent systems mainly rely on agent cooperation. Thus, the frequency of demanding for agent matching is rather high in most agent-based systems including the AOCD-based systems. The adoption of an appropriate agent-matching mechanism will enhance agent cooperation and communication efficiency within a multi-agent system, especially in an AOCD-based system.

This chapter describes our recently developed new agent-matching algorithm, the *Agent-Rank* algorithm, which ranks service-provider agents according to their contributions to a nominated requesting agent based on the AOCD framework. The Agent-Rank algorithm overcomes the problems of agent matching in a large agent network through combining the general ranking scores with the request-based ranking scores. In our experimental evaluation, we have found that the Agent-Rank algorithm can significantly improve efficiency in the agent-matching and re-matching processes.

This chapter is organised as follows. The next section reviews the existing research works on agent matching methodologies and the research problems. Section 6.2 introduces a new Agent-Rank algorithm based on the AOCD framework. This is followed by the detailed descriptions of the two parts of the Agent-Rank algorithm, which are described in two sections respectively. Finally, we provide a brief summary on our research findings in this chapter.

6.1. Background Review of Agent-Matching Methodologies

Multi-agent systems can solve many complex problems by incorporating various task-oriented agents. For instance, a complex financial investment plan can be decomposed into numerous subtasks or procedures such as an investment policy, financial forecasts, organisational resources, credit evaluation and so on. A task-

oriented agent can represent a problem-solver in each subtask domain. In this way, a complex problem can be tackled with a high degree of efficiency and flexibility. Thus applying an effective agent cooperation and communication process is the key to a multi-agent system since no agent possesses a global view of the entire system [18]. A multi-agent system must provide some competent agent matching and searching mechanisms in order to improve the cooperation and communication efficiency among agents.

Existing agent-matching mechanisms are helpful for improving the efficiency of interaction and cooperation between agents [95, 137, 138]. Nevertheless, current knowledge-based reasoning mechanisms are struggling to provide an efficient semantic match based on different information resources [68, 155]. This affects the quality and accuracy of an agent matching process. In other words, existing agent matching mechanisms are limited by current semantic reasoning technology. We often experience this limitation in our day-to-day life. For instance, Web search engines give us some irrelevant information; language translation software misunderstands our words, etc. These situations prove current semantic matching mechanisms are still very preliminary. Therefore, the agent-matching process has to be highly tolerant, even requiring some human decision-making. The expectation of exact matches in every matching process is not realistic. However, narrowing down the matching objects in a complex matching process can significantly improve matching efficiency.

Most of the existing agent-matching methods deploy description language-based matching, such as LARKS (Language for Advertisement and Request for Knowledge Sharing) [137, 138], SDL (Service Description Language) [136], etc. Most of these methods rely on matching agent-service-descriptions, also called service advertisements. These descriptions indicate an agent's capabilities and services. An R-agent can search for a P-agent through matching the descriptions provided by the other agents. In LARKS, there are three matching mechanisms [137, 138]:

- Exact Match: This is the most accurate matching, which requires that both service descriptions being literally equal.
- Plug-In Match: A less accurate matching method, in which two matching descriptions are generally similar. Exact match is a special case of plug-in match.
- Relax Match: The least accurate matching method. It determines the similarity of two descriptions by returning a numerical distance value instead of semantically matching the two descriptions. However, the calculation of numerical distance

value is still based on service descriptions. In other words, semantic matching is still essential to the process.

All of these matching mechanisms are essentially based on semantic matching, which involves searching for an appropriate record (agent) from their information providers (service register). These mechanisms often neglect the probabilistic factors in the matching process. For instance, a plug-in match process might match an appropriate agent from a large multi-agent system. In fact, thousands of agents can fulfil the search requirement and the first matched agent might not be the most appropriate P-agent. Unfortunately, current agent-matching mechanisms do not consider this circumstance and this increases the failures in current agent matching processes. Moreover, current agent-matching mechanisms often neglect some important factors, such as response quality, visiting history, etc. This flaw affects the effectiveness of an agent-matching process. For instance, an unpopular agent might be constantly selected as a P-agent just because it meets the semantic description but actually it may not be the appropriate agent or the knowledge of the selected agent may even be out of date. These problems also appear in the other existing agent-matching methodologies such as SDL [136] and Capability Description Language [161].

Therefore, we introduce a new agent-matching algorithm, namely the *Agent-Rank* (AR, hereafter) algorithm in this chapter. The use of the AR algorithm could eliminate these problems by ranking all the potential P-agent candidates and considering various factors that affect agent matching efficiency. The AR algorithm is a method to improve the efficiency of existing agent-matching mechanisms. In particular, the re-matching process is highly efficient in AR-based systems. For instance, when the selected P-agent(s) is (are) inappropriate, the re-match process is much easier than in current mechanisms. The re-match process in the AR algorithm does not require searching the agent information server again. It simply continues its operation based on the ranking list generated previously.

A comparative ranking function has been suggested in [89]. However, the comparative ranking function mainly deals with goal-oriented tasks and neglects some important factors in agent cooperation such as response quality, visit frequency, agent importance, etc. Moreover, this early study [89] does not distinguish between the P-agent and R-agent, which complicates the ranking process because the ranking objects involve all agents in a system instead of a group of related P-agent candidates. This

results in the comparative ranking process becoming extremely complicated when a large number of agents are involved.

6.2. Agent-Matching Methodologies in AOCD

In an agent matching process, we often expect regular re-match because a selected service-provider agent (P-agent) cannot fulfil its requesting agent's (R-agent) requirement. Thus, it is more practical to have some mechanisms that could provide the information on all P-agents and rank these P-agents according to their previous service quality, service frequency, etc. Hence, an R-agent can avoid massive searching among various P-agents by selecting a P-agent with a top ranking score. The use of the AR algorithm can enhance the efficiency of the existing agent-matching mechanisms. The AR algorithm is inspired by the *PageRank* algorithm [117], which has been successfully applied in Web search applications.

The AR algorithm is based on the AOCD framework [168]. However, the algorithm can be used in other agent-based systems through adopting more complex Agent Association Graphs (see Figure 6.1). The primary concept of the AR algorithm is to rank all the P-agents in a multi-agent network. According to the ranking information, an agent-matching process is able to select appropriate P-agents more efficiently and accurately. Normally, an agent-matching process selects the highest rank P-agent.

The AR algorithm supports both human-based and agent-based decision-making processes. The AR algorithm allows a decision-maker (human or agent) to choose some of the most relevant P-agents according to their ranks. Existing agent-matching mechanisms can improve their matching efficiency and quality through adopting the AR algorithm in their matching processes. Thus, the current matching processes will search for P-agents according to the provided ranks instead of massive searching.

The AR algorithm adopts the underlying concept of the PageRank algorithm [117], but it is very different from the PageRank algorithm. First, PageRank is a Web page-based mechanism, whereas the AR algorithm deals with the agent-matching process within multi-agent-based systems. Second, PageRank does not consider the response quality issue in its matching process. On the contrary, the response quality factor is one of the major factors that affect an agent's ranking score in the AR algorithm. Third, PageRank forms a probability distribution over Web pages. Unlike PageRank,

the AR algorithm indicates the ranks of the P-agent candidates associated with an R-agent. In other words, the frame of reference in PageRank is an overall network, whereas it is a specific R-agent in the AR algorithm.

In the AOCD framework, the agent-ranking process consists of two major stages. The first stage is called the *General ranking* stage, which ranks the importance of all the agents that have relationships with the R-agent. The second stage is *Request-based ranking*, which further focuses on specific searching and matching based on a request and incorporates other major factors such as visiting history, quality of service and the results from the general ranking stage.

6.3. Part I: General-Ranking Algorithm in AR

The general ranking process ranks the importance of all the agents related to the R-agents. The related agents refer to those agents that send or receive requests from or to the request agent. The general ranking process adopts the basic idea of PageRank that calculates agents' incoming and outgoing links to form an importance distribution over an agent network. Unlike PageRank, the visit frequency of an R-agent to a P-agent is calculated in the general ranking process because the frequency of usage of a P-agent is an important criterion for evaluating an agent's importance. We use an *Agent Association Graph* (*AAG*), shown in Figure 6.1, to help analyse the ranking process. The AAG represents the cooperation history of an R-agent with its associated agents.

Drawing an AAG is always the first step in analysing the general ranking process. Figure 6.1 illustrates an example AAG to explain the general ranking process. Figure 6.1 is a Small world topology graph [157] formed by an R-agent and its associated agents (P-agent candidates).

Most of the network graphs in the general ranking process can be transformed to a Small world-based graph. In such a graph, the R-agent (shown as "Request Agent" in Figure 6.1) is always the centre of the graph. The links in these graphs represent previous cooperation between the R-agent and the other agents. The general ranking process ranks all the P-agent candidates in the R-agent-based graph. The ranking process is based on the calculation of the total inbound links of the P-agents.

An example of the general-ranking process is provided in Chapter 8 (Section 8.2.1) to explicitly demonstrate the calculation procedures in this process.

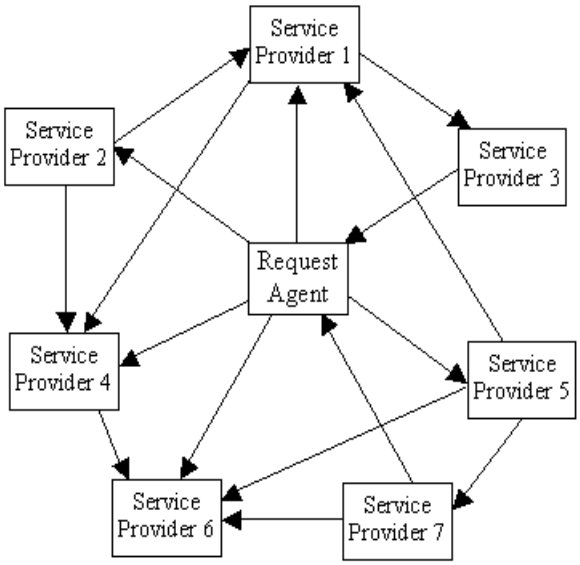


Figure 6.1. An Agent Association Graph.

6.3.1. Factors in the General-Ranking Process

There are several crucial details that need to be considered including the total number of links in a graph, the damping factor, the visit frequency by the R-agent, the total visit numbers in a period, and the response quality. The total number of links, which can be systematically evaluated node-by-node:

$$E = \frac{\sum_{i=1}^{m}(G_i + C_i)}{2} \quad . \qquad \text{(Eq. 6.1)}$$

where E denotes the total number of links in an AAG; G_i denotes the outbound links of an agent; C_i denotes the inbound links of an agent in an AAG; and m denotes the total agent number, including the R-agent, in an AAG. The reason for Eq.6.1 is that: the inbound links of each node (agent) are added to the outbound links to generate the total number of links $\sum_{i=1}^{m}(G_i + C_i)$. When the total link number of all the agents is added, the number is duplicated because one link has two nodes and thus each link is counted twice. Therefore, it needs to be divided by 2, which is the total number of links.

The damping factor in the general ranking process is different from its description in the PageRank algorithm [117]. The damping factor in the general ranking process is based on the ratio of the total number of an agent's inbound links to the total

number of links of all agents associated with the R-agent (including the R-agent). This is because the AR algorithm is an R-agent-based ranking process, so the importance of the R-agent in the network formed by its associated agents affects the ranking scores. The damping factor is calculated as follows:

$$D = C_r / E$$

$$D = \frac{2C_r}{\sum_{i=1}^{m}(G_i + C_i)} \quad . \qquad \text{(Eq. 6.2)}$$

where D $(0 < D < 1)$ denotes the damping factor, C_r denotes the number of an R-agent's inbound links, and E denotes the total number of links in an AAG.

The damping factor is affected by an R-agent's total number of inbound links. When an R-agent's total number of inbound links increases, the importance of the other agents decreases.

The coefficient of a P-agent candidate's visit frequency (*VF*) to an R-agent is considered in calculating the general ranking scores. The VF provides information about how often an R-agent has cooperated with its P-agent candidates previously. We consider it a crucial and request-independent factor. In other words, the VF factor is not affected by the requests sent by an R-agent and the VF information helps to determine whether an agent's capability is out of date.

There are two principles we employ in this preliminary stage including: (*i*) only the P-agent candidates' inbound links from the R-agent are considered, (*ii*) only recent visits are considered (generally 60-180 days subject to specific organisational cases). The coefficient of a P-agent candidate's VF is calculated as follows:

$$f(V_p) = 1 + (V_p / \sum_{i=1}^{n} V_i) \quad . \qquad \text{(Eq. 6.3)}$$

where $f(V_p)$ denotes the coefficient of recent visits from an R-agent to a nominated P-agent candidate; V_p denotes the total number of recent visits from an R-agent to its nominated P-agent candidate p; n denotes the total number of P-agent candidates that have inbound links from their R-agent; V_i denotes the total number of recent visits from an R-agent to all its P-agent candidates on an AAG (i from 1 to n).

The reason for Eq.6.3 is: the coefficient of the VF is important to the general ranking score; however, its impact on the score should not be unlimited. Therefore, the adoption of the ratio of V_p to V_i limits the impact of the VF factor and the ratio is

always between 0 and 1. In addition, the VF factor is positive such that if there are more visits from the R-agent to its P-agent candidate, then this P-agent's general ranking score should be higher. The coefficient is always greater than 1. For a P-agent candidate that only has outbound links to its R-agent, then $f(V_p) = 1$.

The response quality factor provides information about the quality of the service provided by a P-agent to its R-agent previously. In other words, the response quality reflects the history of a P-agent candidate's service quality to its R-agent.

During the process of dealing with a request from an R-agent, a selected P-agent does not always provide a response to the R-agent. The record of the response quality of a P-agent candidate affects the agent's reliability, which affects the agent's ranking score. In other words, if a P-agent candidate's response quality to an R-agent is fairly low then this agent's ranking score based on the R-agent decreases. The response quality issue in the agent matching area has been raised in previous work such as [95]. The response quality is calculated as follows. Let:

$$T_{avg} = \left(\sum_{i=1}^{n} T_i \right) \Big/ n \ ,$$

denote the average response time of all the P-agent candidates communicating with the R-agent previously. The response quality is given by,

$$Q_p = \left(1 - \frac{(T_p - T_{avg})}{T_p} \right) \times \left(\left(1 + \frac{1}{L_p} \right) \Big/ 2 \right)$$

$$= \left(T_{avg} \ / T_p \right) \times \left(0.5 + 1 / 2 L_p \right) \ . \qquad \text{(Eq. 6.4)}$$

where Q_p denotes P-agent candidate X's average response quality to an R-agent (it represents the P-agent's response quality to its R-agent); T_i denotes each individual P-agent candidate's average response time to its R-agent (these P-agent candidates have inbound links from their R-agents) and i is a variable; T_p denotes X's average response time to its R-agent (T_p is the average response time of all the requests from R-agent dealt with by X); L_p denotes the number of links that X established with other agents to solve a request from its R-agent.

The reason for the inclusion of L_p is that: when X cannot resolve a request from an R-agent independently and demands other agents' involvement, then the reliability of the results decrease because additional agents are involved. For a P-agent candidate

that only has outbound links with its R-agent, then $L_p = \left(\sum_{i=1}^{n} L_i \right) \Big/ n$, which is the average value of the other agents' L_p in the same AAG. The reason for using (1 + 1/ L_p) / 2 is to restrict the value of Q_p.

6.3.2. Composition of General Ranking

Based on the previous equations, the general ranking equation can be deduced as follows:

$$GR(A) = \underbrace{\left((1-D) + D \times \left(\sum_{j} \frac{GR_j}{G_j} + GR_{RA} \right) \right)}_{\text{Part 1}} \times \underbrace{Q_A \times f(V_A)}_{\text{Part 2}}$$

where *GR(A)* denotes the general ranking score of a P-agent candidate A (A hereafter); GR_j denotes the general ranking score of P-agent candidate *j* that points its outbound link towards A (votes A); G_j denotes the outbound links of P-agent candidate *j*; *j* is a variable that denotes the P-agent candidates vote A; GR_{RA} denotes the ranking score of the R-agent that has outbound links to the P-agent candidates; Q_A is the response quality of P-agent A to its R-agent; $f(V_A)$ is the coefficient of A's visit frequency. If the selected P-agent has no inbound links from its R-agent, then $GR_{RA} = 0$. Otherwise,

$$GR_{RA} = \sum_{j} \frac{GR_j}{n}.$$

where *n* denotes the total number of P-agent candidates; GR_{RA} denotes the general ranking score of an R-agent. The R-agent's general ranking score is the average value of all its P-agent candidates' general ranking score. The reason to count an R-agent's rank into its general ranking score is that: if a P-agent candidate receives inbound links from its R-agent then the general ranking score of this agent increases. Thus, an R-agent has an impact on an agent's general ranking score. Note that the determination of *GR(A)* is an iterative process, since GR_j in the equation will in turn depend on *GR(A)*.

The general ranking equation can be viewed as two parts. Part 1 calculates A's importance (ranking score) among the other P-agent candidates and part 2 calculates A's ranking score based on its previous relationships with the R-agent. The above general ranking equation can be further expanded as follows:

$$GR(A)=\left(\left(1-\frac{2C_r}{\sum_{i=1}^{m}(G_i+C_i)}\right)+\frac{2C_r}{\sum_{i=1}^{m}(G_i+C_i)}\times\left(\sum_j\frac{GR_j}{G_j}+GR_{RA}\right)\right)\times\frac{T_{avg}\times(1+\frac{1}{L_p})}{2T_p}\times\left(1+\frac{V_p}{\sum_{i=1}^{n}V_i}\right) \quad \text{(Eq. 6.5)}$$

6.4. Part II: Request-Based-Ranking Process in AR

The general ranking scores are based on the history of P-agent candidates' services. On the contrary, the *Request-Based Ranking* (referred to as RBR hereafter) disregards previous relationships between R-agent and P-agent candidates and it only calculates the ranking scores based on a particular request.

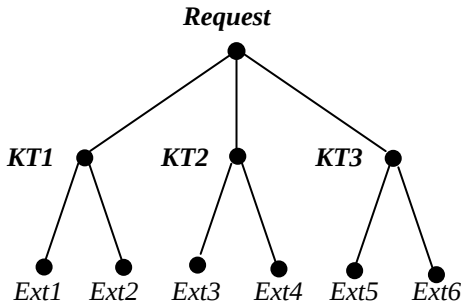


Figure 6.2. An RBR Semantic Extension Tree: *KT* denotes key term; *Ext* denotes semantic extension.

In [15], a Weighted-Tree Similarity (WTS) algorithm is introduced for agent matching in the E-business environment. Despite the limitations of the WTS algorithm, the basic idea behind the WTS seems useful in the agent matching process. The RBR calculation process adopts the similarity tree concept, in which a request is decomposed into several key terms and each key term is further extended to be a more specific and comprehensive description. Through the semantic extension process, a request from an R-agent turns into a semantic extension tree (see Figure 6.2) and the RBR scores are based on the calculation of the end nodes.

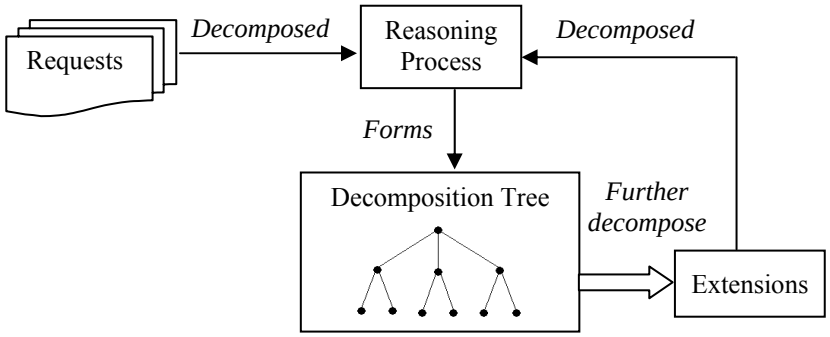


Figure 6.3. Generating A Semantic Extension Tree.

A request is decomposed through a reasoning process and forms a semantic extension tree. Some extensions can be further expanded and added to the tree. The final semantic extension tree is completed when it exhausts all the extensions of each node. Figure 6.3 shows the process of generating a semantic extension tree. The RBR process is based on the similarity calculation of semantic extension trees the agent capability description trees (see Section 6.4.1).

The process of constructing a RBR semantic extension tree or an agent capability description tree is based on existing semantic networks [134] and ontology technology. There are several reasons for adopting a tree-based similarity calculation method in agent matching in preference to using existing text mining or Web-based semantic matching technology.

Firstly, a large amount of time delay resulting from an unsuccessful agent matching process necessitates high matching accuracy. The time delay in an unsuccessful agent matching process is much greater than in an unsuccessful Web page search. This is because a false match is only detected in an agent matching process when a P-agent fails to handle the request from its R-agent. For instance, a P-agent spends a great deal of time dealing with a request but finally fails to provide some useful results. Therefore, conducting a pre-evaluation based on an agent's overall capabilities to find the most (or nearly the most) appropriate agent is necessary. The time delay for a re-match in the Web-based searching process is much lower than that of agent matching. Thus, efficient similarity calculation in agent matching is crucial.

Secondly, current agent description languages cannot distinguish agent capabilities efficiently as many agents, especially in a large multi-agent system, seem to provide very similar descriptions. Therefore, an agent matching process requires more accurate and detailed analysis compared to simple text mining.

Thirdly, the tree-like descriptions are efficient representations of an agent's capabilities. An agent's capabilities and functions can be efficiently broken down into more specific details by using a tree-like representation method. The following sections explain the main procedures of the RBR calculation. The agent capability description tree describes how agent capabilities are constructed and how requests can be reformed for comparing with agent capabilities. The RBR similarity calculation process indicates the how a request is calculated based on the agent capabilities.

6.4.1. Agent Capability Description Tree

In [137], an agent description language, LARKS, is deployed to describe an agent's capability and to provide an efficient agent service matching mechanism. In [115], agent capabilities are formalised in a Beliefs-Desire-Intentions (BDI) framework. Based on previous work, we suggest a top-down folding tree to classify agent capabilities.

In a top-down folding tree, the root of the tree describes an agent's capability set and each successive level parent node represents the capability of an agent, where parent nodes represent the sub-domains to which the child nodes belong. The leaf nodes define the general categories to which the agent's capability belongs. The middle level nodes and leaf nodes in a description tree can be folded, which enhances the efficiency in describing the agent's capabilities by omitting irrelevant or unimportant information. The category classification of the fields of knowledge of science, social science and arts, and humanities adopts various standards such as [62]. Figure 6.4 gives an example of a top-down folding tree.

The folding process may occur when a parent node only has one child node. These parent and child nodes can be folded as one node and their descriptions are joined together by using a separator '→'. If there are more than two levels of these nodes, they all can be folded as one node and their descriptions joined together. Moreover, the bottom two levels of nodes can always be folded. If there is more than one child node attached to a parent node in the last two levels of the tree, then fold them and join the nodes' descriptions by using '+' to separate the child leaves (see Figure 6.4, under 'Statistics').

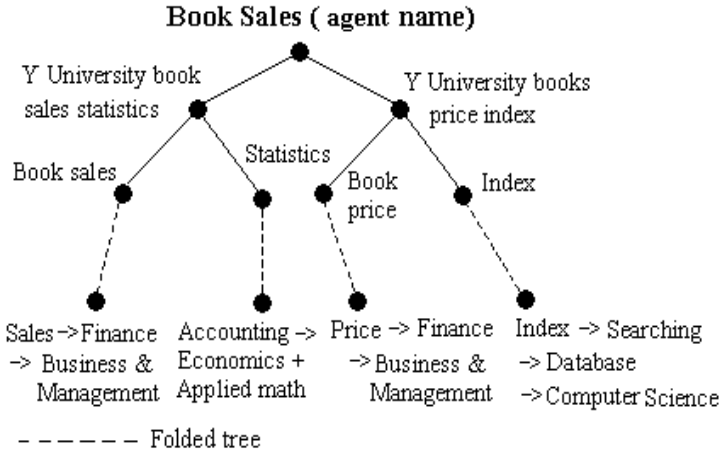


Figure 6.4. An Example of A Top-down Folding Tree.

6.4.2. RBR Similarity Calculation

An extraction process is deployed to collect two information sets from two trees while calculating the similarity between a RBR semantic extension tree and an agent capability description tree. Each set contains a number of semantic elements and coefficients representing the node information in both trees. The similarity calculation is based on semantic matching between the two sets and coefficient calculation of the matched elements. The RBR similarity calculation process makes use of some concepts from existing data mining and information retrieval technologies including keyword-based retrieval, similarity-based retrieval and document classification analysis [69].

The *Cosine-Similarity Measure (CSM)* [59, 126] has been applied extensively in the information retrieval domain to calculate the similarity between two vectors [59], which is their inner product normalised by their lengths,

$$sim_{Cos}(X,Y) = \frac{\vec{X} \cdot \vec{Y}}{\left|\vec{X}\right\|\vec{Y}|} \quad . \qquad \text{(Eq. 6.6)}$$

However, the CSM cannot function efficiently when the dimensions of these vectors are unequal. To overcome this, we adopt the *Cartesian-Product* [71] method in the RBR-based similarity calculation between two sets, which is defined as:

$$A \times B = \left\{ (a,b) \middle| a \in A \ \ and \ \ b \in B \right\} \qquad \text{(Eq. 6.7)}$$

where *A* contains the elements extracted from a RBR semantic tree, and B contains the elements extracted from an agent capability description tree. The RBR-based similarity calculation consists of three main steps. The first step generates a Cartesian product; the second step calculates the similarity value of two elements in the Cartesian product; the final step sums the level coefficient and the path coefficient of a pair of matched elements through incorporating their similarity values. For instance, consider two sets S_1 and S_2, which are extracted from a RBR semantic extension tree (S_1) and an agent capability description tree (S_2), respectively.

The first step is to generate the Cartesian product by using the agent capability description tree and a RBR tree. Let:

$$S_1 = \left\{ l_1 A_1, l_2 A_2, \ldots l_i A_i \right\},$$

$$S_2 = \left\{ \frac{1}{p_1} B_1, \frac{1}{p_2} B_2, \ldots \frac{1}{p_j} B_j \right\},$$

where A_i is an element in an agent capability description tree; B_j is an element in the RBR extension tree; l_i denotes the level coefficient of A_i; p_j denotes path coefficient of B_j and N denotes the total extension number of a nominated node's parent node. For a RBR extension tree, the similarity value decreases when the level number increases, whereas the similarity value increases when the path number increases in an agent capability description tree. We shall refer to the root node as level 0 (or path 0), its child nodes as level 1; and the rest of the nodes' level numbers are increased by 1 from their parents' level number. Let $i = 3$, $j = 4$, thus the Cartesian-product of set S_1 and S_2 is:

$$S_1 \times S_2 = \left\{ (l_1 A_1, \frac{1}{p_1} B_1), (l_1 A_1, \frac{1}{p_2} B_2), (l_1 A_1, \frac{1}{p_3} B_3), (l_1 A_1, \frac{1}{p_4} B_4), \right.$$
$$(l_2 A_2, \frac{1}{p_1} B_1), (l_2 A_2, \frac{1}{p_2} B_2), (l_2 A_2, \frac{1}{p_3} B_3), (l_2 A_2, \frac{1}{p_4} B_4),$$
$$\left. (l_3 A_3, \frac{1}{p_1} B_1), (l_3 A_3, \frac{1}{p_2} B_2), (l_3 A_3, \frac{1}{p_3} B_3), (l_3 A_3, \frac{1}{p_4} B_4) \right\}$$

The next step is to calculate the similarity value of two elements (words) in a Cartesian product. In the RBR-based similarity calculation process, every agent description tree is fully expanded and every node is calculated in the Cartesian product. The methods used in constructing descriptive trees and comparing elements between semantic elements can be diverse according to existing semantic technology. Several projects have been carried out in this area, such as *WordNet* [45]. Here, we employ a three-type semantic relationship model, which is used in WordNet, to describe the similarity relationships between two words. The three types of relationships [45, 96] are:

(*i*) Synonym: Two words are synonymous.

(*ii*) IS-a: Two words are in a superset and subset relationship.

(*iii*) Has-a: One word has ownership of another word. Also known as part-whole relationship between words.

We define that if two words are synonyms, then the similarity value is 1; if two words have an 'IS-a' relationship then the similarity value is 0.9; if two words have a 'Has-a' relationship then the similarity value is 0.85. The initialisation value of each relationship type is based on previous experience, and these values can be adjusted according to different applications.

If two words match, their upper (or lower in the case of agent capability description tree) level words are required for further matching. The comparison process continues until the entire upper or lower levels are matched. An algorithm is provided below to demonstrate the process of calculating the similarity values in other levels related to the two matched words.

Algorithm: Similarity Value Calculation

```
1.  S := 1;  /* two matched nodes for further match */
2.  while (NodeX ≠ Root) and (NodeY ≠ Leaf) do
3.        if Synonym (NodeX, NodeY) := true then
4.             {S := (S + 1) / 2;}
5.        else if ISA (NodeX, NodeY) := true then
6.             {S := (S + 0.9) / 2;}
7.        else if HAS (NodeX, NodeY) := true then
8.             {S := (S + 0.85) / 2;}
9.        else
10.            {S := (S + 0.1) / 2;}
11.       /*if other level nodes not match S = 0.1*/
12.       Node(X) := NodeSelection (NodeX+1);
13.       Node(Y) := NodeSelection (NodeY-1);
14.       MoveUpper (NodeX);
15.       MoveLower (NodeY);
16.    return (S);
end.
```

In this algorithm, S denotes the similarity value of a Cartesian product (two elements); Node X denotes a node in the RBR extension tree; Node Y denotes a node in an agent capability description tree; Synonym (*x, y*) is the function that determines whether two words are synonyms; ISA (*x, y*) is the function determining whether two words are hypernyms/hyponyms; HAS (*x, y*) is the function determining whether two words are meronyms/holonyms; NodeSelection (*x*) is the function that determines which node is the next level node to move to. MoveUpper (*x*) is the function that moves to the upper level node for comparison (for the RBR extension tree); MoveLower (*y*) is the function that moves the lower level node for comparison (for the agent capability description tree). If two words do not match then exit the process (if at the starting point then $S = 0$; else S remains the last calculated value). If there are

more than two upper or lower level nodes, then select the highest similarity value. If all the similarity values are the same value then randomly select one of them.

The final step is to extract level and path coefficients and to perform an equivalence operation between two elements in a Cartesian product to calculate the RBR-based similarity value. The RBR-based similarity score (value) is calculated as:

$$RS = \frac{\left(\sum_{i \in R, j \in C} \frac{l_i}{p_j} \times S_{ij} \right)}{M} = \frac{\left(\sum_{i \in R, j \in C} \frac{l_i}{p_j} \times (S_d + S_x) \right)}{M} . \quad \text{(Eq. 6.8)}$$

Here, *RS* denotes the final RBR-based similarity score; *C* denotes the total level number of a RBR semantic extension tree, *R* denotes the total level number of an agent capability description tree; $S_{i,j}$ denotes the composite similarity value; *M* denotes the total element number in the Cartesian product; S_d denotes the independent similarity value of a Cartesian product without considering other level nodes; S_x denotes the total similarity values of other nodes that are ancestors or descendants of the two nodes in the Cartesian product; l_i denotes the level number of a node in an agent capability tree; p_i denotes the level number of a node in a RBR semantic tree.

Table 6.1. RBR Similarity Values for Set A and B.

Cartesian Product	Independent Similarity Value	Composite Similarity Value
A_1, B_1	1	1
A_1, B_2	0	0
A_1, B_3	0	0
A_1, B_4	0.85	0.85
A_2, B_1	0	0
A_2, B_2	1	1
A_2, B_3	0.9 (composite A_1, B_4)	(0.9+ 0.85) / 2 = 0.875
A_2, B_4	0	0
A_3, B_1	0.9	0.9
A_3, B_2	0	0
A_3, B_3	1 (composite A_1, B_4)	(1 + 0.85) / 2 = 0.925
A_2, B_4	0	0

For instance, supposing Synonym (A_1, B_1) = true; HAS (A_1, B_4) = true; Synonym (A_2, B_2) = true; ISA (A_2, B_3) = true; ISA (A_3, B_1) = true; Synonym (A_3, B_3) = true. We have the results shown in Table 6.1.

In Table 6.1, *Independent Similarity Value* indicates the similarity calculation of a Cartesian product without considering the nodes in other levels. *Composite Similarity Value* indicates the final similarity value of a Cartesian product that incorporates the similarity value of the nodes in other levels.

Thus, we have the similarity values for all the Cartesian products:

$$S_1 \times S_2 = \left\{ \begin{array}{llll} (\frac{l_1}{p_1}), & 0, & 0, & 0.85(\frac{l_1}{p_4}), \\ 0, & (\frac{l_2}{p_2}), & 0.875\ (\frac{l_2}{p_3}), & 0, \\ 0.9(\frac{l_3}{p_1}), & 0, & 0.925\ (\frac{l_3}{p_4}), & 0 \end{array} \right\}$$

Therefore, the final similarity score of two trees is:

$$RS = \left\{ (\frac{l_1}{p_1}) + 0.85(\frac{l_1}{p_4}) + (\frac{l_2}{p_2}) + 0.875(\frac{l_2}{p_3}) + 0.9(\frac{l_3}{p_1}) + 0.925(\frac{l_3}{p_4}) \right\} \Big/ 12$$

Overall, the RBR-based similarity calculation process follows the two main equations (6.7) and (6.8). Various agents possess different agent capability description trees, and the RBR process ranks the similarity scores of all the agent capability description trees. The final agent ranking scores combine the general ranking scores and the RBR similarity scores.

In Chapter 8, the experimental results and examples are provided (see Section 8.2.2) to further explain the RBR similarity calculation process.

6.5. Summary

In a versatile large-scale multi-agent system, the effective matching and deployment of agents will have a critical impact on overall problem solving performance. The agent-matching problem has a different level of complexity compared with the searching and matching of web pages. Over and above the linear matching of strings to determine relevance, the discovery of functions and capabilities is required. Here, we use a methodology whereby past relationships and performance are taken into account.

The Agent-Rank algorithm provides an efficient methodology to match an R-agent's request within numerous P-agents. The AR algorithm consists of two major processes: the general ranking process and request-based ranking process. The general ranking process improves matching efficiency through ranking the related P-agents that have a service history and incorporating major ranking factors such as the total number of links, visit frequency, damping factor, and response quality. In the request-based ranking process, the R-agent's requests and the P-agent candidates' capabilities are converted to semantic-based description trees. The matching process is based on similarity computations on the semantic trees. The P-agent candidate with highest similarity score is more likely to be selected as the P-agent. The Agent-Rank algorithm avoids massive search among numerous agents, and is performance efficient in a large multi-agent system. The AR algorithm considers the relationships between agents, i.e. service-provider agents and request agents. The Matrices are the frameworks for supporting agent activities, however they do not interfere the ranking process. Therefore, the Matrices are not involved in the agent ranking process.

Although the present algorithm does carry some ranking computation overhead, such computation cost is considered acceptable in comparison with the penalty of inappropriate matching which may result in defective solutions and time wasted in carrying out re-matching.

CHAPTER 7

THEORETICAL ANALYSIS OF AGENT NETWORK TOPOLOGIES

Existing topological theories in the computing field have been mainly applied to data communications and distributed systems areas. As an emerging discipline, the research work in the multi-agent area has not, currently, attached much importance to topological theory. The topological theories based on other areas cannot fulfil the needs of designing an agent network because it has specific characteristics, which include: mobility and intelligence. In other words, an agent-based topological network design not only takes the agent distribution into consideration but also needs to consider the agent mobility and intelligence. Most of the current research work in the agent network topology area adopts topological theory from the distributed system and computing network fields without considering mobility and intelligence aspects. Therefore, the development of topological theory in the agent area has become a fundamental and crucial issue in the multi-agent field.

An *agent network topology* represents the information of agent distribution over an agent network, which incorporates agent mobility and intelligence aspects into the process of arranging and configuring an agent network. The term, "agent network topology", is derived from mathematical topological theory. This concept overlaps with topological theory in data communications and distributed systems areas. Different from existing network topological theory, such as LAN or distributed system networks, agent network topology emphasises communications and transactions between agents. As mentioned above, the mobility and intelligence issues are the major concerns in agent network topologies and these aspects have not received much attention.

Topological analysis of intelligent agent networks provides crucial information about the structure of agent distribution over a network. It enables the adoption of suitable frameworks for specific multi-agent systems such as the AOCD framework. However, many existing multi-agent systems are still struggling with constructing the appropriate and efficient networks for agent cooperation and communication. It seems

to be very difficult for most of these systems to be successful because of the lack of a guide for using appropriate topologies for constructing an agent network. To address these issues, this chapter provides a classification of agent network topologies and a novel topological description language for agent networks.

In this chapter, agent network topologies are systematically classified into two major categories, which include the simple-agent-network topology category and the complex-agent-network topology category. In each category, the agent network topologies are further broken down into several sub categories. The AOCD framework adopts a hybrid topology, which is a combination of several simple agent network topologies. This chapter provides a concreted performance analysis on three common agent-network topologies based on the AOCD framework. In Chapter 8, we also provide some experimental results to prove the efficiency of the hybrid network based on the AOCD framework compared with other two common agent network topologies.

We also introduce a new topological description language for intelligent agents in this chapter, i.e. the *Topological Description Language for Agent networks* (*TDLA*), which provides an efficient method for agents to perceive the network information and make appropriate decisions.

7.1. Background Review and Related Works

The analysis of network topologies has been carried out in many disciplines. For instance, *(i)* cost calculation methodologies for wide area network design [122] has been used for network design estimation; *(ii)* performance analysis of distributed systems has been carried out in [5, 72]; *(iii)* analysis of network topologies impact on dependability offers an evaluation method for estimating the effect of topology on dependability [83]; *(iv)* The description of the seven evaluation properties for analysing distributed systems topologies has been pointed out in [105]. Notwithstanding the previous works in network topologies, the current topological theory based on multi-agent systems is inadequate.

7.1.1. Motivations

Current multi-agent systems are becoming larger and more complex. Adopting an appropriate agent network topology is the key to constructing an efficient agent

network. Unlike traditional network topological theories, such as Local Area Network (LAN) or traditional distributed systems that do not consider intelligence issues, agent network topological theory takes the mobility factor and the intelligence factor into consideration. It also incorporates the other major factors, such as the agent matching efficiency, deadlock issues, and concurrency control issues into agent topological analysis. However, current research work in the agent network topology area is not systematic and relies on graph-based methodology. In addition, most of the traditional network topologies are graph-based and static, and are inefficient in describing dynamic networks such as multi-agent networks. This is because the graph-based topologies are unable to provide the descriptive information of a multi-agent network, which confounds the abstract understanding of the overall network. Beside, the graph-based topological analysis of a network topology often lacks precise measurements of each agent. Moreover, existing agent network topologies are incapable of providing much detailed information of each agent and its relationship with other agents on a network. This increases the difficulty of agent communication and cooperation, such as agent searching or matching, over a network. In brief, existing topological theory cannot fulfil the needs of agent networks.

In these circumstances, we systematically classify agent topologies [171] and develop an experimental program to evaluate the performances of different agent network topologies based on the AOCD framework [172, 173]. We suggest that the research direction of agent network topologies needs to follow three fundamental characteristics including intelligence, mobility, and flexibility. We summarise the impact of these factors on agent networks as follows:

- The agent mobility factor in an agent network topology has a strong impact on the topology because multi-agent networks require flexible structures, allowing agents to enter and leave a system efficiently.
- The intelligence factor is another key factor that distinguishes agent network topology from existing topologies. The individual nodes in existing networks, such as LAN or distributed systems, do not consider intelligence issues, whereas intelligence is one of the most important characteristics of agents. TDLA reflects the intelligence factor in agent network topology theory [171].
- An agent network design also needs to take the flexibility factor into consideration. This is particularly crucial for an open system such as the AOCD framework since most open systems are changing constantly.

7.1.2. Traditional Network Topological Theory

The development of the agent network topological theory is based on traditional topological theories, particularly the Distributed Artificial Intelligence (DAI) area. Extensive research has been carried out in analysing the network topologies in the DAI and data communication network areas [105, 122].

In the real world, a network for an industry organization is complex and specific to that organization. Therefore, it is not realistic to establish a general model for many different agent networks. However, a complex network can be divided into several simple topologies. For instance, Local Area Network (LAN) theory generally defines four basic topologies [104, 122], which include star topology, bus topology, ring topology, and tree topology. In practice, we often use these topologies by combining them into one network, which is called a hybrid topology. This phenomenon also has been found in distributed systems such as multi-agent systems. Based on the traditional topological theory of the LAN, three basic network topologies for distributed systems have been introduced in [105], which include: centralised, decentralised and hybrid topologies. The topological theory developed in [105] is not systematic and lacks the comprehensive analysis of topological theory. However, it provides some basic ideas about the classification of simple distributed networks and is helpful to the development of topological theory in multi-agent systems. There are three major forms of hybrid topologies introduced in [105]: (*i*) centralised + centralised, (*ii*) centralised + ring, and (*iii*) centralised + decentralised (partial connection). The hybrid topology in the AOCD design is different from the previous works; rather it adopts centralised + mesh topology (fully connected). The centralised + mesh topology is an efficient but costly solution. Nevertheless, existing technologies are able to reduce costs through using SuperNode [48] mechanisms. In recent years, the hybrid topology began to be used in distributed systems like those described in Groove [38], KaZaa [113], and Morpheus [143]. The successful performance of these systems inspires the use of the hybrid topology in distributed DSS, and as a result the AOCD framework is proposed.

As mentioned in the previous sections, traditional network topologies mostly focus on static or graph-based networks, which often neglect dynamic changes and uncertainty factors in a transaction. To fulfil the requirements of large-scale agent networks, the complex network topologies are suggested.

7.1.3. Complex Agent Network Topological Theory

Most of the topologies for distributed systems do not take an individual node's conditions, such as mobility, intelligence and relationships with other nodes, into consideration. Moreover, traditional topological theories have limitations in describing simple networks, which are relatively small in size and can be fitted into a certain pattern. The patterns to describe simple agent network topologies are classified into several categories [171]. From a future perspective, as multi-agent applications become more widespread, agent networks will expand and eventually become fairly large networks. This means that future agent network topological theory will rely on complex agent network topological theory.

Several researchers have carried out related work in the complex network topology area. The organisational structures of agent networks have been preliminarily categorised into ten paradigms [75] including Hierarchy, Holarchy, Coalition, Team, Congregation, Society, Federation, Market, Matrix, and Compound. Each agent organisational paradigm has certain advantages and disadvantages for agent cooperation and coordination. However, this classification method does not mainly focuses on agent network infrastructures; rather it considers agent cooperation strategies. This classification method also does not take the topological performances of agent networks into consideration. In addition, this classification does not explicitly clarify the network forms of each paradigm.

To specifically analyse the efficiency of a complex agent network, the topological theory should be the major method for the analysis rather than considering other factors, such as an agent cooperation strategy. Existing complex agent network topologies are various, and the Small-world topology [5] and Scale-free topology [72] are the most influential concepts among the existing complex agent network topologies.

The development of topological theory soon spreads to complex agent networks [2, 80] after the introduction of the Small-world theory and its application to complex networks [135, 156, 157]. Most of the related work in the field emphasises the application of specific topologies, such as Small-world topology or Scale-free topology, to agent networks. This work does not raise the issue of classification of agent networks, which limits the research in terms of gaining a comprehensive

understanding of agent networks. Thus, the following sections contribute to the clarifying of the agent-based network topologies.

7.2. Classification of Agent Network Topologies

As mentioned in the previous sections, the topological theory in multi-agent systems area is still preliminary. A systematic study on agent network topologies is urgently required. In this section, we summarise our previous work [171] and classify the agent network topologies into two main categories, which include (*i*) simple agent network topologies, and (*ii*) complex agent network topologies. Notwithstanding that complex agent network topologies can be eventually divided into a number of simple agent network topologies, we argue that it is very necessary to develop the topological theory for complex agent networks. This is because complex network theory provides a conceptual and abstract view of an overall network especially for a large-scale agent network.

7.2.1. Simple Agent Network Topologies

We classify the simple agent network into six basic topologies:

(1) Centralised agent network topology;
(2) Peer-to-peer agent network topology;
(3) Broadcasting agent network topology;
(4) Closed-loop agent network topology;
(5) Linear agent network topology;
(6) Hierarchical agent network topology.

Mobility analysis of the agent network topology is the key issue as it directly impacts on the overall network performance. Therefore, in this section we emphasise the topological classification and mobility analysis of each simple agent network topology.

Centralised agent network topology

We define the centralised agent network topology as: the topology has a central agent and only this central agent is connected with other agents over the network. There is no direct connection between any two agents except with the central agent, as

shown in Figure 7.1. A star-like topology is one of the common examples of centralized topology.

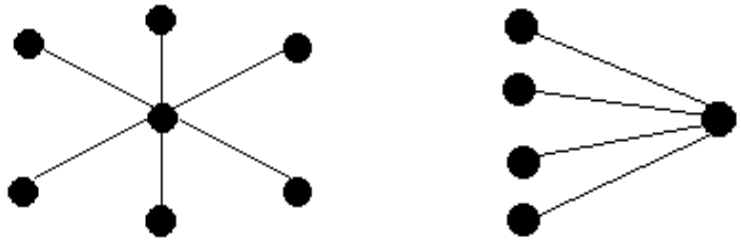


Figure 7.1. Centralized Topology.

In the centralised agent network topology, the central agent is vital to the network. However, the central agent's mobility is very inefficient. The total connection of a topology is one of the key facts that affect agent network mobility. Based on our previous agent mobility analysis [171], we develop the following equation to define the connection-based coefficient of agent network mobility:

$$m_{co} = \frac{1}{\sum_{i} (Totalconnection / import) \times m_i} \qquad \text{(Eq. 7.1)}$$

where m_{co} is the connection-degree coefficient of mobility. m_i is the mobility of an individual agent *i*, which indicates the degree of flexibility that an agent detaches from an agent network. *Import* means the importance of an agent that is measured by the number of connections the agent has. These variables also will be used in the remaining topologies in this section. Based on Eq.7.1, the connection-based coefficient of centralised agent network mobility is:

$$m_{co} = \frac{1}{\sum_{i=1}^{v} ((v-1)/a_i) \times m_i} \qquad \text{(Eq. 7.2)}$$

where a_i is the number of connections for agent *i*, *v* denotes the total number of agents over a network including the central agent. The variables *v* and a_i will also be used in the following topologies in this section.

The centralised agent network topology has a relatively stable mobility coefficient in terms of increasing agent numbers. The terminal agents of the centralised topology have high mobility as they only have one connection to the network centre. However,

the whole network relies heavily on the central agent that makes the mobility of the central agent very inefficient.

Peer-to-peer agent network topology

In the peer-to-peer agent network topology, each agent has direct connection(s) with other node(s) over a network. There is no restriction on the connections between agents. There are two categories in the peer-to-peer agent network topology: one is the fully connected peer-to-peer topology and another is the partially connected peer-to-peer topology as shown in Figure 7.2.

(a) Fully Connected ***(b) Partially Connected***

Figure 7.2. Peer-to-peer Agent Network Topology.

In the fully connected peer-to-peer topology, each agent has connections with all the other agents over a network. In the partially connected peer-to-peer topology, each agent has at least one connection with another agent over a network and the maximum connections are smaller than *c*. In the fully connected peer-to-peer topology, if *v* denotes the number of agents over a network, then *c* is given by exhausting all combinations from choosing any two agents:

$$c = \binom{v}{2} = \frac{v(v-1)}{2} \ , \ 1 \le v < \infty \qquad \text{(Eq. 7.3)}$$

Thus, the connection-based coefficient of the fully connected peer-to-peer agent network mobility is:

$$m_{co} = \frac{1}{\sum_{i=1}^{v} (c / a_i) \times m_i} \qquad \text{(Eq. 7.4)}$$

In the fully connected peer-to-peer agent network, the agent network mobility decreases rapidly when the total agent number increases.

Broadcasting agent network topology

We define the broadcasting agent network topology as follows: all the agents are connected through a common media and there is no direct connection between any of the two agents as shown in Figure 7.3. A bus topology is a typical broadcasting agent network topology.

In the broadcasting agent network topology, every agent has the same role in the network (unlike in the centralised agent network topology, where the central agent has a more important role than other agents) and there is no direct connection between any two agents. The connection-based coefficient of the broadcasting agent network mobility is:

$$m_{co} = \frac{1}{\sum_{i=1}^{v} (v / a_i) \times m_i} \qquad \text{(Eq. 7.5)}$$

The network mobility of the broadcasting agent network topology is stable and efficient. The importance of agents over the network is the same. Each agent has only one connection to the media, which results in the mobility of agents over the network being even.

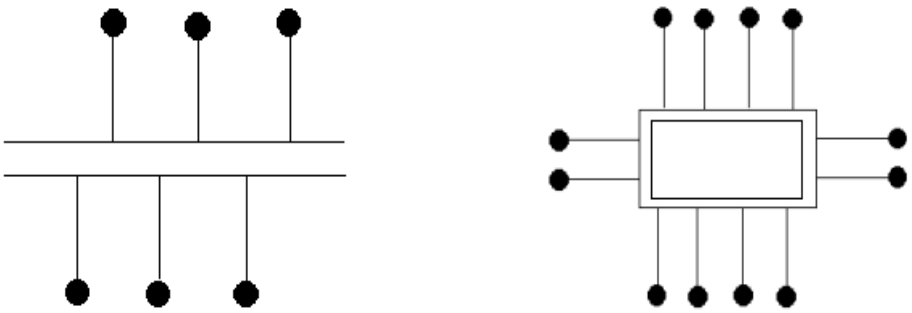
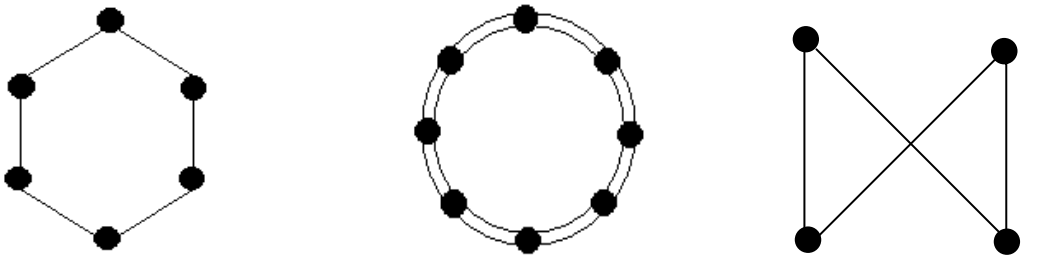


Figure 7.3. Broadcasting Topology.

Closed-loop agent network topology

In the closed-loop agent network topology, a network forms a loop and each agent connects exactly to two other agents. In the case of removing connections between any two agents, the topology will turn into a linear topology as shown in Figure 7.4.



(a) Simple Ring *(b) Dual Ring* *(c) Anomalous closed-loop*

Figure 7.4. Closed-Loop Topology.

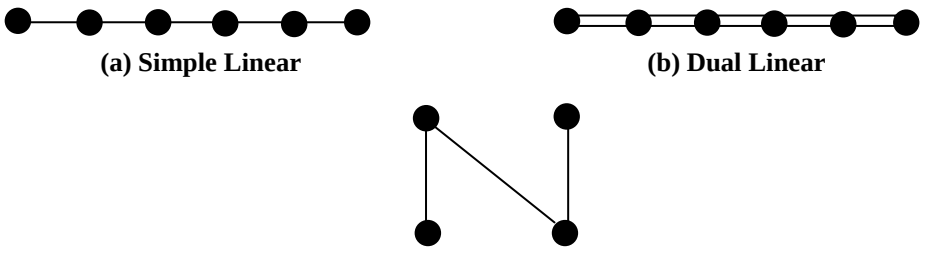
Simple ring and dual ring topologies are the typical closed-loop topology. The connection-based coefficient of closed-loop agent network mobility is:

$$m_{co} = \frac{1}{\sum_{i=1}^{v} (v / a_i) \times m_i} \qquad \text{(Eq. 7.6)}$$

The coefficient calculation of network mobility is based on the total connections over the network. Therefore, the closed-loop agent network topology has the same coefficient as the broadcasting topology as they have the same connections when they have the same number of agents over a network. However, for an individual agent, the closed-loop agent network topology is less efficient than the broadcasting agent network topology because there are two connections for each agent in the closed-loop topology.

Linear agent network topology

In the linear agent network topology, all agents are distributed in linear form in sequential order and there is no loop in the network. Each agent is connected to two neighbour agents except two end agents that are connected to only one neighbour agent. In many cases, this topology is regarded as inefficient because the communication between two end agents is extremely inefficient. However, in some cases it appears to be efficient such as in a pipelining process.



(a) Simple Linear **(b) Dual Linear**

(c) Anomalous Linear

Figure 7.5. Linear Topology.

The connection-based coefficient of linear agent network mobility is:

$$m_{co} = \frac{1}{\sum_{i=1}^{v}((v-1)/a_i)\times m_i} \quad \text{(Eq. 7.7)}$$

Basically, the linear agent network topology can be considered as a ramification of closed-loop topology. The coefficient calculation of network mobility based on the total connections is slightly smaller than for the closed-loop topology. In practice, we regard the coefficients of the network mobility of these two agent network topologies as the same, especially when the number of agents is large. Figure 7.5 gives three typical linear agent-network topologies.

Hierarchical agent network topology

In the hierarchical agent network topology, an agent is a basic unit and a number of agents form a group, which is connected to an upper level agent as shown in Figure 7.6. In the hierarchical agent network topology, an agent is not connected to other agents except with its upper level agent.

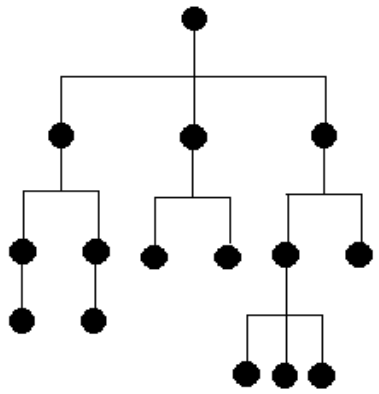


Figure 7.6. Hierarchical Topology.

A recursive method can be used to determine if a topology is a hierarchical topology by starting from the end points, which have no lower-level agent connected. We also define a very important characteristic of hierarchical topology: there is no loop in a hierarchical topology. The connection-based coefficient of the hierarchical agent network mobility is:

$$m_{co} = \frac{1}{\sum_{i=1}^{v}((v-1)/a_i)\times m_i} \quad \text{(Eq. 7.8)}$$

The total numbers of network connections for hierarchical and linear topologies are the same when their agent numbers are the same. However, the mobility of an

individual agent in a hierarchical topology is more complex because the importance of each individual agent in a hierarchical network is different. This fact results in the removal of the important agents (those that have more connections), which are inefficient and it consequently decreases the overall network mobility.

7.2.2. Complex Agent Network Topologies

Traditional topological theory is insufficient to describe a complex agent network such as a multi-agent system. This is because the traditional topological theory is unable to define the relationships between nodes and describe an overall view of the network efficiently. In these circumstances, the topological theory for a complex network is required to provide more abstract descriptions. Current topological theory classifies networks into four major categories:

(1) Regular Network Topology;
(2) Random Network Topology;
(3) Small-World Network Topology;
(4) Scale-free Network Topology.

In a regular network, agents are distributed in an orderly way and the connections between nodes are based on certain constraints: for example, the wiring process is based on finding neighbour agents the shortest distance from each other [157].

In a random network, the connections between two agents are generated randomly [43]. A random network is more realistic than a regular network in describing real-world complex networks such as multi-agent systems. However, the limitation of the random network topology is the difficulty of predicting, monitoring and controlling the network.

In the real world, a network topology is assumed to be either completely regular or completely random. The actual topology is somewhere between these two extreme cases and it is defined by Watts and Strogatz as the *Small-world* topology [157]. Small-world network theory enables possible control or monitoring over the network especially in some critical areas of a network through observing and adjusting the probability p. p denotes the probability of randomness when an agent is connecting to other agents. When p is 0, a network is wired completely in order. When p is 1, a network is wired completely randomly. For $0< p <1$, a network is in small-world topology scope.

The three topologies we discussed in the previous sections are basically static and homogeneous, with a peak at an average value and decay. Such networks are called *exponential networks* [154]. However, recent research in the field of complex networks indicates that a number of large-scale complex networks, including the Internet, World Wide Web, and metabolic networks, are scale-free [10] and the vertices over such a network are not homogeneous. The scale-free network topology is considered the most suitable topology for multi-agent based systems taking the high-degree mobility of an agent network into account.

Mobility calculations for complex agent network topologies are based on that agent network's specific facts. We argue that there is no general and precise equation (or a set of equations) for a complex agent network's mobility calculation because every complex agent network has a very different structure to other complex agent networks. Unlike the simple agent network topology, the classification of complex agent network topologies is conceptual and abstract. Therefore, the mobility calculation for a complex agent network has to be specific to that network. We recommend using a macro-statistics methodology to calculate the mobility of a complex agent network instead of dividing the complex network into numerous simple agent networks and calculating their values individually. We predict that the calculated result of a complex network topology should be close to the combination of numerous simple agent networks' calculated results.

The following sections further introduce these complex agent network topologies in detail.

Regular network topology

In a regular network, nodes (agents) are distributed in order and the connections between nodes are based on certain constraints. For example, the wiring process is based on finding neighbour agents within the shortest distance. Figure 7.7 shows an overview of a regular network, a small-world network and a random network [157]. We will explain the transformation process shown in this Figure in the small-world section.

A regular network topology can describe simple networks but it is incapable of describing complex networks efficiently. Generally, regular network topology is limited to describing static networks.

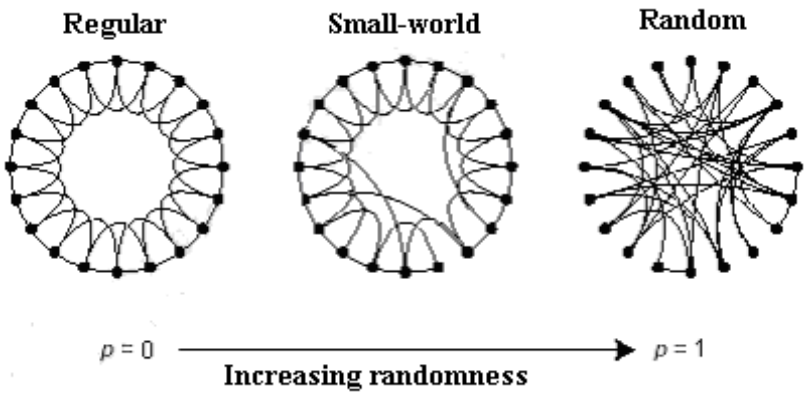


Figure 7.7. Overview of Three Complex Network Topologies [157].

Random network topology

Random network topology is based on random graph theory, which is described by Erdós and Rényi in 1959 [43]. In such a network, the connections between two nodes (agents) are generated randomly.

A graph with vertices contained in a set X can be considered as a binary relation $R \subset X \times X$ by defining R as: $(a,b) \in R$ if there is an edge between a and b. On the contrary, each symmetric relation R on $X \times X$ gives rise to a graph on X. A random graph is a graph R on an infinite set X satisfying the following properties [19]: *i*) R is irreflexive. *ii*) R is symmetric. *iii*) Given any $n+m$ elements $a_1,...,a_n,b_1,...,b_m \in X$ there is $c \in X$ such that $c \in X$ is related to $a_1,...,a_n$, and c is not related to $b_1,...,b_m$.

Based on random graph theory, a random network topology can describe a large-scale complex network. It is more realistic than a regular network in describing the real-world complex networks such as multi-agent systems. However, the limitation of random network topology theory is the difficulty of predicting, monitoring and controlling a network. For most agent-based systems this is unacceptable because most of the implemented agent-based systems require a high degree of monitoring and controlling. Therefore, the Small-world network topology is suggested.

Small-world network topology

As we introduced in the previous sections, a regular network topology is easier to monitor and control but it is inefficient in describing a real-world complex network. Conversely, the random network topology has a high degree of disorder, which

increases the difficulties of operations over the network such as agent search, agent matching, etc.

In the real world, the connection topology is treated as either completely regular or completely random. It is somewhere between these two extreme cases and it is defined by Watts and Strogatz as the Small-world topology [157]. As shown in Figure 7.7, *p* denotes the probability of randomness when an agent is connecting with other agents. When *p* is 0, a network is wired completely in order. When *p* is 1, a network is wired completely randomly. For $0 < p < 1$, a network is in Small-world topology scope.

Small-world network theory enables the possible control or monitoring over the network, especially in some critical areas of a network, through observing and adjusting the probability *p*.

The Small-world concept is becoming important in multi-agent systems, in which agents are often considered as nodes. It is difficult to use simple or regular agent network topology to describe an overall view of a large multi-agent system. Small-world topology could efficiently describe the conceptual view of a complex agent network. However, Small-world topology still lacks the ability to adapt to a dynamic environment. In other words, Small-world topology is not an ideal solution for the networks that are constantly changing. Therefore, Scale-free topology is suggested.

Scale-free network topology

The three topologies we discussed in the previous sections are basically static and homogeneous, and peak at an average value and decay. Such networks are called *exponential networks* [154]. However, recent research in the field of complex networks indicates that a number of large-scale complex networks including the Internet, WWW, and metabolic networks, are scale-free and the vertices over such a network are not homogeneous.

Barabási and Albert [10] suggest a scale-free network topology, in which a network is allowed to change network connections dynamically and the nodes (agents) on the network are inhomogeneous. The generation scheme of a scale-free network can be summarized into two major steps [154]:

a) **Growth:** Start with a small number (m_0) of nodes; at every time step, a new node is introduced and is connected to $m \leq m_0$ existing nodes.

b) **Preferential attachment:** The probability p_i that a new node will be connected to node i (one of the m existing nodes) depends on the degree k_i of node i. As a result, $p_i = k_i \Big/ \sum_j k_j$, where degree k_i is the number of edges of a node.

Scale-free network topology is considered as the most suitable topology for multi-agent based systems taking the high-degree mobility of an agent network into account. As we know, one of the most important characteristics of an agent network is that it requires high-degree mobility. To support such a high-degree mobility, a network topology needs the ability of adapting to the dynamic environment and that is the advantage of scale-free topology compared with the other three static topologies.

7.2.3. Hybrid Agent Network Topologies

A hybrid agent network topology often combines two or more simple agent network topologies. Since the hybrid agent network topology is relatively more complex than the other six basic simple agent network topologies, it is often regarded as a complex network. Technically speaking, the hybrid agent network topology is still a simple agent network topology because it operates on a small scale compared to the complex network topologies from a large-scale point of view. However, hybrid agent network topology inherits some of the characteristics from both simple and complex agent network topologies. Every hybrid network topology can be analysed through dividing them into several simple agent network topologies and incorporating them with complex agent network topologies. Thus, it is not necessary to establish a new category for hybrid agent network topologies. In other words, the analysis of hybrid network topologies are based on the theory of simple agent network topologies and complex agent network topologies.

A hybrid agent network cannot describe an overall complex network in a specific and efficient way. However, it can explain a complex network in a simple way and it is efficient to a limited scale. It can solve some complex problems that the simple agent network topology cannot solve appropriately. For instance, two different agent networks can be joined together to form a hybrid agent network without changing their current structures. A hybrid agent network topology can be a combination of any two or more simple agent network topologies such as a closed-loop topology and a centralised topology. The hybrid network topology eliminates the difficulty of

concurrent control, which mainly plagues the centralised topology. Our study shows that the hybrid topology offers superior performance in agent-based systems [170].

Nevertheless, simple agent network topological theory (including hybrid topology) can only describe a limited scale of agent networks in a simple way. Hence, current topological theory in the multi-agent field adopts more complex topological theory such as small-world topology and scale-free topology to describe a large-scale complex network.

Similar to that of the complex agent network, the mobility of a hybrid agent network topology is the combination of a number of simple agent networks. Unlike the complex agent network topology, in a hybrid agent network we can divide a network into a number of simple agent networks because we define a hybrid agent network as being much smaller than a complex agent network, which is formed by a couple of simple agent networks. For an AOCD-based network, the coefficient of the network mobility is the same as that of the centralised agent network topology, which is:

$$m_{co} = \frac{1}{\sum_{i=1}^{v}((v-1)/a_i) \times m_i} \qquad \text{(Eq. 7.9)}$$

The reason for Eq. 7.9 is that: in an AOCD-based system, the connections between the Matrix and agents exist permanently, which are (v – 1) connections (v is the total number of nodes on the network). But the connections between agents exist temporarily. Therefore, we will not take the temporary connections between agents into account although they might slightly decrease the network's mobility. Figure 7.8 shows two typical hybrid agent-network topologies.

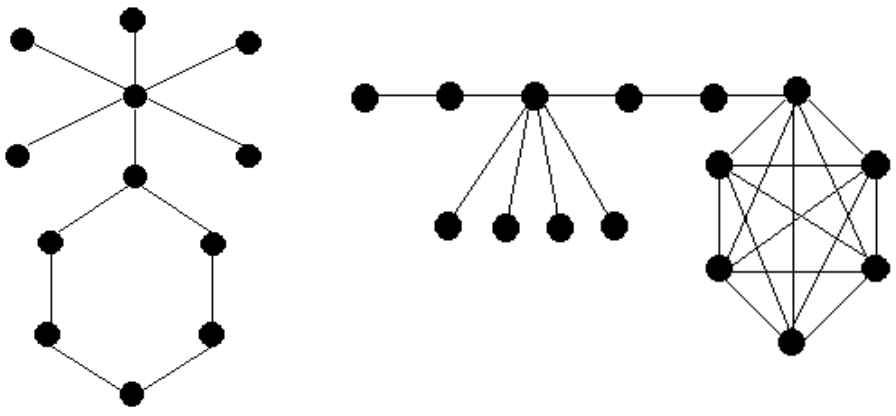


Figure 7.8. Two Typical Hybrid Agent Network Topologies.

7.2.4. Topological Description Language for Agent Networks

Topological Description Language for Agent networks consists of three major sections. These three sections are: *i*) *Individual Agent Description* (*IAD*). *ii*) *Main Agent-groups Description* (*MAD*). *iii*) *Overall Agent-network Description* (*OAD*).

In the IAD section, the description emphasises the information about each agent on a network. The content of IAD includes (*i*) degree of links (the number of links connected to an agent), (*ii*) extensibility, which indicates whether an agent allows new connections to be attached and how many connections can be attached, (*iii*) local address, which is essential for grouping agents by location or generating actual geographic map, (*iv*) attachment probability and (*v*) routing table, which stores the information of connected agents to the described agent.

As shown in Figure 7.9, an example of an IAD expression for the circled agent is as follows. The attachment probability indicates the probability of the selected agent to be attached by other agents. The assumptions are: (*a*) the nominated node is allowed to attach a maximum of 10 agents, and (*b*) there are 50 vacant attachments available among the agents that have connections with the nominated agent (including the nominated agent).

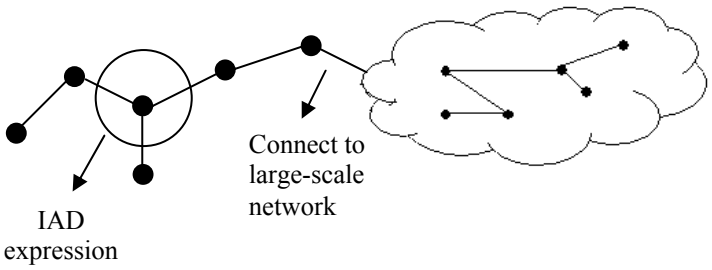


Figure 7.9. IAD for Individual Agent.

(1) Degree of links: 3;

(2) Local address: 138.77.201.20;

(3) Extensibility: $MC - DL = 7$ (*MC*: Maximum Connections, *DL*: Degree of Links);

(4) Attachment probability: $\text{Individual Extensibility} \Big/ \sum_{i=1}^{n} (\text{MC} - \text{DL})\,i$;

where *n* is the total number of agents that have connections with the nominated agent (including the nominated agent) and the individual extensibility is the current agent's extensibility. The result in this case is: 7/50 = 0.14.

(5) Routing table: See Table 7.1.

Table 7.1. IAD Routing Table.

Connected Agent Address
138.77.201.38
138.77.201.37
138.77.202.1

In the MAD section, the description provides network information based on a group of local agents. A large-scale agent network is usually divided into a number of sub networks (or groups). The information provided by MAD describes the information on main sub networks. MAD information includes (*i*) total agent number in a group, (*ii*) total number of links in a group, (*iii*) main group selection criteria, (*iv*) possible root(s) nomination, (*v*) loop detection in a group and (*vi*) context within an overall network. The criteria for grouping a number of agents are various, and are based on real cases. The geographic area indication is one of the common criteria for selecting a number of agents as the main agent group. If an agent has much more maximum connection capacity than other agents, this agent normally is nominated as the root of the group. In some cases, there is no agent nominated as the root. It occurs when the maximum-connection capacity of each agent over a network is equal. To further explain the MAD, we use the previous example and suppose the circled part is a main group of agents as shown in Figure 7.10.

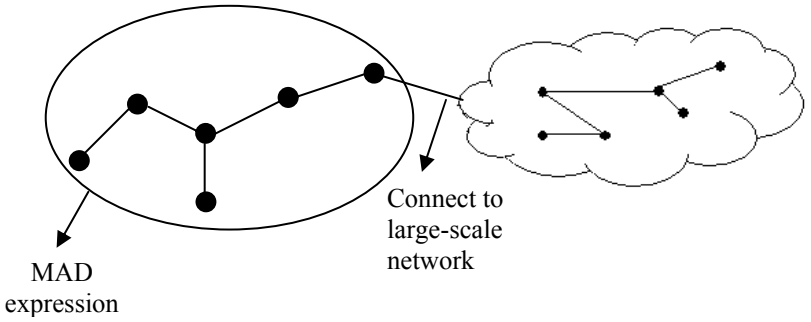


Figure 7.10. MAD for A Main Agent Group.

The MAD is expressed as follows:

(1) Total number of agents: 6;

(2) Total number of links in the group: 6;

(3) Main group selection criteria: geography-based in Melbourne/Australia;

(4) Root nomination: 138.77.201.20;

(5) Loop detection: No loop in the group;

(6) Context: major connection with the group that has nominated root 138.77.201.1.

In the OAD section, the general information about the network is provided, which includes (*i*) the diameter of the network, (*ii*) total agent number and (*iii*) info of main agent groups. The diameter of the network is $D = max_{(i,j)}\, d(i,j)$, where *i* and *j* represent two agents. In other words, the longest path between two agents is the diameter of the network. The previous example in Figure 7.11 can be used to explain the OAD expressions:

(1) Diameter of the network: 9;

(2) Total agent number: 12;

(3) Main agent group info: Total group number: 2

Group 1: (Root nomination: 138.77.201.20)

(Group selection criteria: based in Melbourne)

Group 2: (Root nomination: no root)

(Group selection criteria: geography-based in Brisbane).

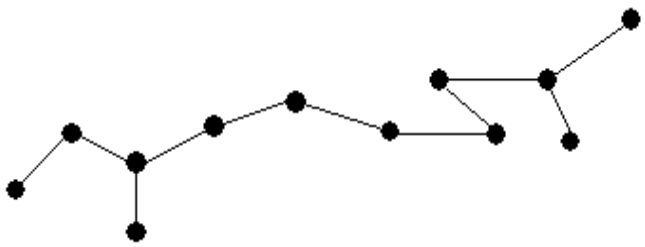


Figure 7.11. OAD for Overall Agent Network.

Given the descriptions provided by IAD, MAD and OAD, a network generator is able to automatically determine the capability of the network, the preferable area to attach new agents, the topology category (or the combination of topologies) that the network most likely belongs to and the mobility of each agent. In principle, the agent that has a lower degree of connections is likely to have more mobility. TDLA offers the intelligent capability of generating a description of an agent network through using statistical results provided by its three sections.

7.3. Theoretical Analysis Based On Three Agent Network Topologies

The agent framework in AOCD architecture is a hybrid topology combining centralised and decentralised topologies. Three common network topologies are suggested for the agent framework [55]: centralised topology, decentralised topology and hybrid topology. The hybrid topology is likely to be the most efficient framework

for agent communication in the AOCD architecture. In the AOCD architecture, the Matrix plays as a centralised coordinator that mainly dealing with agent relationship management and the communications between agents are decentralised. The following analysis quantifies the performance of implementing each topology in AOCD. Our analysis adopts the following notations:

- i : the position of a request set in a queue,
- J: total number of agents in an AOCD system,
- M: the number of requests that the Matrix can handle at any one time,
- N: the total number of n requests that are sent by a number of agents in a short period of time,
- R: the average size of a record in the agent information list,
- T: the average transmission time between two nodes,
- TR: the total transmission time for N requests,
- W: the total waiting time for all the agents.

The precondition of this analysis is that all the sampling requests are sent synchronously. In addition, we assume that a waiting queue is deployed to store the agents that have not been processed and will be processed in the future. It excludes the waiting time while two agents are connecting and serving. We present the following calculations for the three different agent network topologies.

7.3.1. Analysis of Centralised Agent Network Topology

In the centralised topology, as Figure 7.12 shows, an agent sends requests to the Matrix. The Matrix delivers the requests to the corresponding agents and returns the results to the requesting agent.

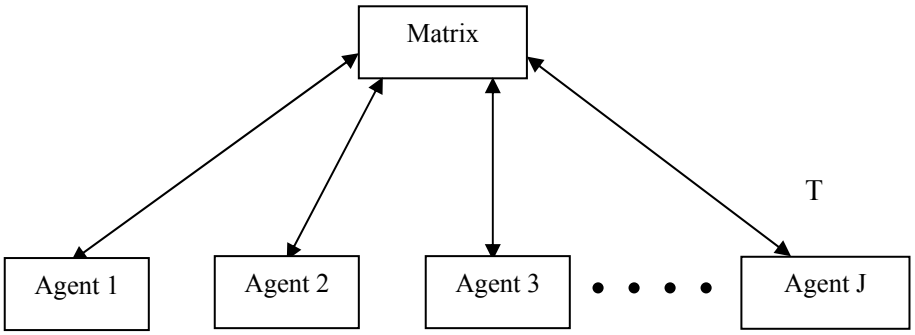


Figure 7.12. Centralised Transmission Framework.

The transmission time for a set of requests in a short period of time

All the communications between agents are transmitted through the Matrix. The total transmission time for *N* requests is bounded:

$$\lceil 2N/M \rceil \times 2T \le TR \le \lceil N/M \rceil \times 2T + N \times 2T \qquad \text{(Eq.7.10)}$$

The reason for $TR \ge \lceil 2N/M \rceil \times 2T$ is as follows. In the best case, the Matrix receives *N* requests from requesting agents then delivers them to the corresponding agents; and ideally the corresponding agents send back the results to the Matrix synchronously. In other words, there are 2*N* requests sent to the Matrix, which include *N* requests from the requesting agents and *N* requests from the corresponding agents. All the results are sent to the Matrix synchronously. Therefore, 2*N* requests are broken into $\lceil 2N/M \rceil$ sets. One set of requests contains 1 to *M* requests and the Matrix can handle one set each time. It takes 2*T* to process one set of requests, which includes the requests from the requesting agent costing *T* and the results from corresponding agents costing *T*. The transmission in the centralised topology is a two-way transmission including: sending (or receiving) requests (or results) to (or from) the Matrix and receiving (or sending) results (or requests) from (or to) the Matrix.

The reason for $TR \le \lceil N/M \rceil \times 2T + N \times 2T$ is as follows. The transmission time for delivering the requests from the requesting agents to the Matrix is $\lceil N/M \rceil \times 2T$, which happens synchronously. In the worst case, the results return to the Matrix one-by-one and each request costs 2*T* of transmission time. The transmission time for delivering the results is $N \times 2T$. Therefore, the total transmission time for the requests is $\lceil N/M \rceil \times 2T$ plus $N \times 2T$.

Total waiting time for completing a set of requests

In many cases, the agents are sending requests synchronously. Therefore, a waiting queue is deployed for storing the later-coming requests (see Figure 7.13).

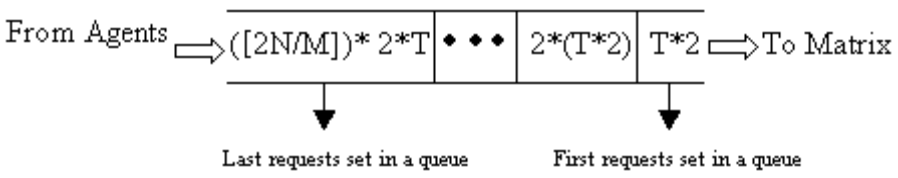


Figure 7.13. Waiting Time in A Queue.

For each request set, the waiting time in a queue is: $(\lceil i/M \rceil - 1) \times 2T$. As mentioned before, each request set contains 1 to *M* requests. Our calculation sums up each individual request set as these request sets may occur in different places; therefore we need to evaluate the total time for each of the request sets.

The total waiting time for all request sets in the queue is:

$$T\left((1 + \lceil 2N / M \rceil) \times \lceil 2N / M \rceil\right), \qquad \text{(Eq.7.11)}$$

where $N > M > 0$. The reason for $N > M$ is that if the queue is not empty then this implies that there must have been more than *M* requests sent to the Matrix. Otherwise, the waiting queue will be empty which means that there is no waiting time during transmission processes.

In Eq.7.11, we find that the average transmission time for a set of successful requests in the same period is increased when the total number of requests is increased. The unsuccessful transmission time will not be included in this model because the Matrix eliminates most of the conflict requests between agents.

Memory consumption for storing agent information

In the AOCD framework, the memory allocated to store agent information by using the centralised topology is: $R \times J$. The reason for this outcome is that in the centralised topology all of the agent information is stored in a central component. The other nodes (agents) will not keep the agent information. Therefore, the memory allocated to store the agent information is the size of each information record, which is *R,* multiplied by the agent number, which is *J*.

7.3.2. Analysis of Decentralised Agent Network Topology

Unlike other decentralised topologies, such as the decentralised topology described in [105] that are partially connected topologies, the decentralised topology discussed in this paper is a fully connected topology. As Figure 7.14 shows, the decentralised topology provides direct communication between agents.

This method eliminates the transmission time between agents and the Matrix. However, the coordination and cooperation approaches are more complicated in the decentralised topology than other topologies as agents are communicating directly and individually.

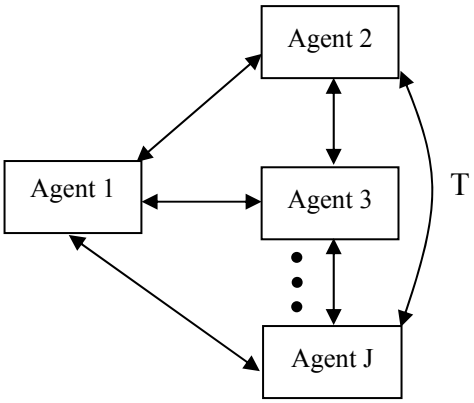


Figure. 7.14. Decentralised Transmission Framework.

Transmission time for a set of requests in short period

In a decentralised framework, no matter how many requests are occurring concurrently, the total transmission time for *N* requests is bounded by:

$$2T \le TR \le T + N \times T \,, \qquad \text{(Eq. 7.12)}$$

where $2T$ is the best case when the communication between agents happens synchronously. In other words, the requesting agents send the requests, which cost *T* and ideally send back the results to the requesting agent, which also costs *T*.

$T + N \times T$ is the worst case in which the requesting agents send requests in time *T* but the results from the corresponding agents are received asynchronously. Only one result is transmitted over the network each time. In the decentralised framework, the probabilities for successful requests are decreased dramatically when synchronous requests are increased in number.

Total waiting time for completing a set of requests

Compared with the centralised topology, the communications over the decentralised network are more likely to be disorderly. A central waiting queue for the decentralised topology is not feasible because there is no central control. Therefore, the requests in the decentralised topology will be calculated individually. Concurrent control methodologies are required, and all the requests will send detection signals repeatedly until the corresponding agent is available. In spite of deadlock and other concurrent control failures, the waiting time for all requests is bounded by:

$$0 \le W \le N \times (N-1) \quad , \qquad \text{(Eq. 7.13)}$$

where, if an agent has made *X* requests, then this agent will be counted as *X* number of agents. The best-case situation is when all the requests are accepted and processed without delay. In other words, all the requesting agents find their corresponding agents are all available. Therefore, the waiting time in the best case is 0.

The worst-case situation is when every time there are only two agents communicating, while all the other agents are waiting. In other words, there is only one request processed each time. In the calculation, there are 2*N* agents, which means that each request takes two agents for communication. Table 7.2 shows a demonstration of the worst case when four requests need to be processed. As we can calculate from Eq.7.13, four requests require a waiting time of 12 *T*.

Table 7.2. Total Waiting Time for Processing 4 Requests (worst case).

Agent	Agent 1	Agent 2	Agent 3	Agent 4	Agent 5	Agent 6	Agent 7	Agent 8
Request 1	Wait (T)	Wait (T)	Process	Wait (T)	Process	Wait (T)	Wait (T)	Wait (T)
Request 2	Wait (T)	Process	X	Process	X	Wait (T)	Wait (T)	Wait (T)
Request 3	Process	X	X	X	X	Wait (T)	Wait (T)	Process
Request 4	X	X	X	X	X	Process	Process	X
Completion	X	X	X	X	X	X	X	X

Memory consumption for storing agent information

Information sharing technology has been used in many current distributed systems, such as Gnutella and BearShare [83, 113]. Each node in such a decentralised framework carries a subset of the overall information of the system for searching and other purposes. Information sharing technology reduces redundancy in a decentralised system. Unfortunately, redundancy cannot be eliminated completely in decentralised systems because decentralised systems, particularly p2p networks, apply a high degree of redundancy to secure availability and fault tolerance [119].

In the case of implementing a decentralised topology in the AOCD framework, the memory allocated to store agent information is: $\wp(J) \times R \times J$, where, $\wp(J)$ is the average number of agent information records carried by each agent. The reason for the above calculation is that in the decentralised topology, each agent carries a certain number of records of agent information, which is $\wp(J)$. The records of agent information carried by each agent are a subset of the overall records in the system. In other words, the total number of records of agent information in the overall system

without redundancy is J. These records are distributed to each agent redundantly and each agent is allocated $\wp(J)$ records on average. Therefore, the total memory allocated to store the agent information is the product of the above factors.

7.3.3. Analysis of Hybrid Agent Network Topology

In this chapter, we introduce a novel hybrid framework that is different from the hybrid topologies introduced in [105]. The proposed hybrid framework provides a structure combining centralised and decentralised topologies. As shown in Figure 7.15, all the agents are connected to a central Matrix and each of them keeps connections with all the other agents. This hybrid topology is a centralised + mesh topology, in which agents are fully connected with each other. This topology reduces the workload of the Matrix and enhances manageability by using the Matrix as a central control component.

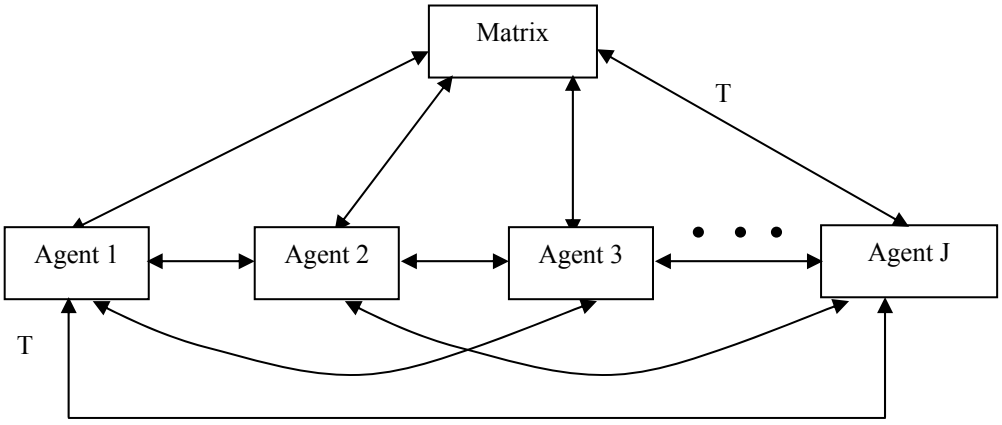


Figure 7.15. Hybrid Transmission Framework.

Transmission time for a set of requests in short period

In the hybrid framework, communication involves the agents and the Matrix. The total transmission time for N requests is:

$$\lceil N/M \rceil \times 2T + T \le TR \le \lceil N/M \rceil \times 2T + N \times T \qquad \text{(Eq. 7.14)}$$

The reason for the bound $\lceil N/M \rceil \times 2T + T$ is that in the best case, there are $\lceil N/M \rceil$ sets of requests. Each request set requires a transmission time of $2T$. T is incurred by the requesting agents sending requests to Matrix and another T is consumed by the Matrix delivering the requests to the corresponding agents. In the best case, all the requesting agents receive the results from the corresponding agents at the same time, which incur a time of T. Therefore, the total time in the best case includes the time for sending

requests that is $\lceil N/M \rceil \times 2T$ and the time for receiving results costing *T*. The reason for $\lceil N/M \rceil \times 2T + N \times T$ is that the time for sending requests is $\lceil N/M \rceil \times 2T$ because all the requests are sent synchronously as mentioned in the assumptions. In the worst case, there is only one result transmitted over the network each time and there are *N* results to transmit for completing the whole transmission process. Therefore, the total time for transmission in the worst-case situation is the sum of the time for sending the requests, which is $\lceil N/M \rceil \times 2T$, and the time for receiving the results, which is $N \times T$.

Total waiting time for completing a set of requests

Similar to the other two topologies, the waiting time in the hybrid topology includes *(i)* the waiting time for sending requests, and *(ii)* the waiting time for receiving results. Here, we have the total waiting time for completing a set of requests in the hybrid topology.

$$2T \sum_{i=1}^{\lceil N/M \rceil} i \le W \le 2T \sum_{i=1}^{\lceil N/M \rceil} i + T \sum_{i=1}^{N} (N - i) \qquad \text{(Eq. 7.15)}$$

The reason for $W \ge 2T \sum_{i=1}^{\lceil N/M \rceil} i$ is that *N* requests are delivered to the Matrix, which incur $2T \sum_{i=1}^{\lceil N/M \rceil} i$ time. In the best case, the corresponding agents send back the results to the requesting agent without delay, which means that the waiting time is 0. Therefore, the total time in the best case is: $2T \sum_{i=1}^{\lceil N/M \rceil} i + 0$.

The reason for the second inequality is that the time required for delivering requests to the corresponding agents is still the same as the best case. However, in the worst case, each time there is only one result delivered to the requesting agent and each request incurs a waiting time of *T*.

Memory consumption for storing agent information

In the AOCD architecture, the memory allocated to store the agent information in the hybrid topology is same as in the centralised topology, namely, $R \times J$.

The reason for this result is that in the hybrid topology all of the agent information is stored in a central component. The other nodes (agents) will not keep the agent

information. Therefore, the memory allocated to store the agent information is the size of each information record, which is *R*, multiplied by the number of total agents, which is *J*.

7.3.4. Overall Performance Analysis

We take M = 3 and the number of requests N is from 1 to 46 (step = 3). These variables are adjustable. Here, we adopt a small number of requests (N = 46) that could provide a relatively clear explanation within a figure. Figure 7.16 shows the total transmission time for the three topologies in the AOCD framework. TRC indicates the area in which the centralised topology might incur the transmission time. TRD indicates the area in which the decentralised topology might incur the transmission time. TRH indicates the area in which the hybrid topology might incur the transmission time.

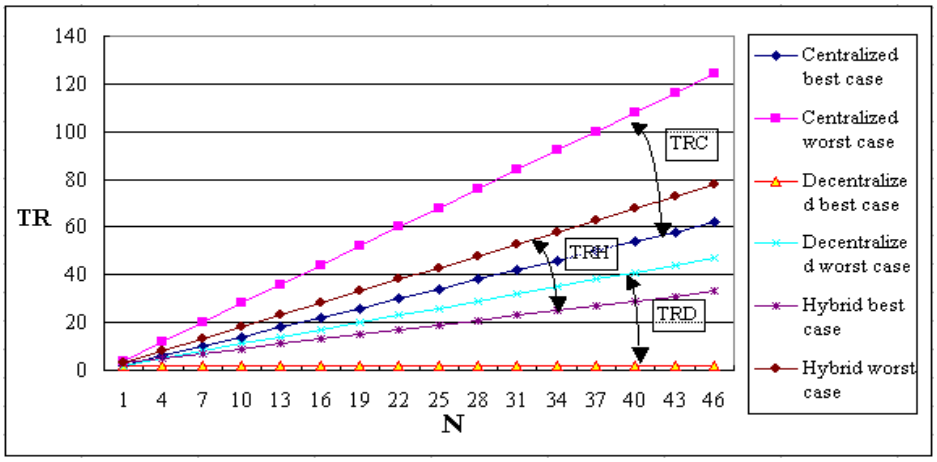


Figure 7.16. Total Transmission Time for The Three Topologies.

As shown in Figure 7.16, the angle formed between the "best case" line and the "worst case" line in the centralised topology is ∂. According to Eq.7.10, ∂ = Arctan $((\lceil 2N/M \rceil \times 2T)/N)$ − Arctan $((\lceil N/M \rceil \times 2T + N \times 2T)/N)$. The angle formed between the "best case" line and the "worst case" line in the hybrid topology is θ. According to Eq.7.14, θ = Arctan $((\lceil N/M \rceil \times 2T + N \times T)/N)$ – Arctan $((\lceil N/M \rceil \times 2T + T)/N)$. The angle formed between the "best case" line and the "worst case" line in the decentralised topology is α. According to Eq.7.12, α = Arctan $((T + N \times T)/N)$ − Arctan $(2T/N)$. When N = 10, the three angles are: *(i)* ∂ = 15.88 °, *(ii)* θ = 18.96 °, and *(iii)* α = 36.4 °. $\partial < \theta < \alpha$ indicates that the centralised topology is the most stable topology. However, the gradient of the "worst case" line of the centralised topology is

far higher than the other lines, which reflects that the increase in the number of requests would cause much higher transmission time compared to the other two topologies. The best-case total transmission time for the decentralised topology is the most efficient among the six cases. However, the angle formed by the "worst case" line and the "best case" line is the greatest among the three topologies, which reflects that this topology lacks stability.

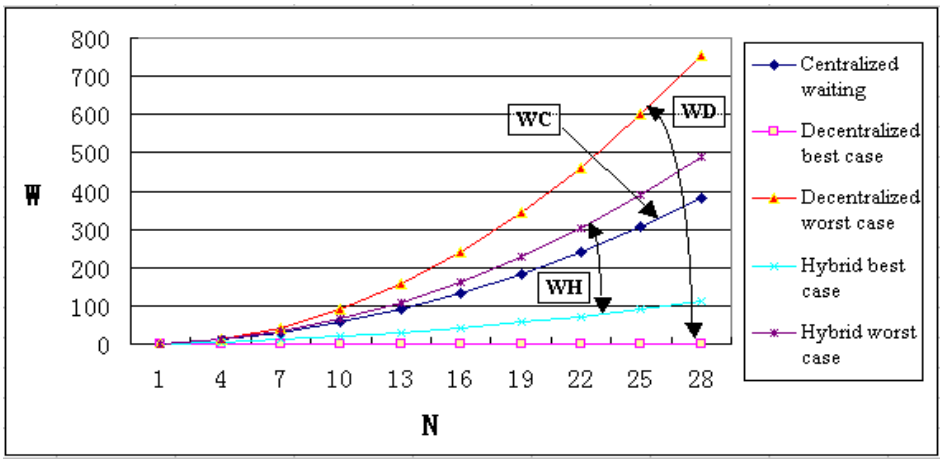


Figure 7.17. Total Waiting Time for The Three Topologies.

Among the three topologies, the hybrid topology exhibits best performance. It has good stability and low total transmission time. The hybrid topology balances the weaknesses and advantages of the other two topologies. Figure 7.17 shows the total waiting time for the three topologies in AOCD framework. The parameters are $M = 3$, N is from 1 to 28. We decrease N (from previous case 46 to 28) since the total waiting time for the decentralised topology has increased dramatically, and it does not affect the results analysis.

WC indicates the area in which the centralised topology might incur the waiting time; WD indicates the area in which the decentralised topology might incur the waiting time; and WH indicates the area in which the hybrid topology might incur the waiting time. As shown in Figure 7.17, the decentralised topology exhibits its unstable performance because the WD area, which formed by the "worst case" line and the "best case" line of the decentralised topology, is the biggest among the three topologies. In other words, the waiting time in the decentralised topology can either be the highest or the lowest among all the cases, which reflects its instability. The centralised topology is the most stable one as it only contains a single case of total waiting time. The hybrid topology is more stable than the decentralised topology but

less than the centralised topology. Nevertheless, the "best case" line of the hybrid topology is far below the "waiting time" line of the centralised topology and the "worst case" line of the hybrid topology is close to the "waiting time" line. This indicates that usually the consumption of the waiting time for a set of requests in the hybrid topology is lower than in the centralised topology.

The memory consumption for agent information in the decentralised topology is highest among the three topologies in the AOCD design. The hybrid topology and centralised topology have the same memory consumption volume, which is the number of records in the central control multiplied by the average record size. Therefore, the memory consumption volume for the decentralised topology is $\wp(J)$ times of the other two topologies, where, $\wp(J)$ is the average size of the subset of the overall agent information in the system.

7.4. Summary

An agent network topology provides crucial information about the agent distributions over the network. Conducting theoretical analysis of an agent network is the precondition for constructing an efficient multi-agent framework. However, current research work on topological theory in the multi-agent area is still preliminary.

The significance of this chapter is that it provides a systematic treatment for clarifying and organising the current topological structure in the intelligent agent field. Agent network topologies can be generally classified into two main categories, which include the simple agent network topologies and the complex agent network topologies. Each category can be further classified into several sub-categories through incorporating the agent mobility factor and existing graph theory. This classification provides agent network designers an efficient analysis mechanism for applying an appropriate network infrastructure to a multi-agent system.

In an agent-based network, intelligence is a key issue as it is one of the most important characteristics of an intelligent agent. TDLA reflects the intelligence factor in agent network topology theory. TDLA, which is based on the agent network topology theory, enables an individual agent to perceive other agents' information and the information on the network that the agent resides in. Therefore, an agent can be proactive before performing any actions, such as moving or matching, in an agent

network. In other words, TDLA endows agents with intelligence capabilities to perceive the network environment.

In this chapter, we conducted a performance analysis based on three common agent network topologies. Based on the analysis, we find that the hybrid topology presents a superior performance in AOCD frameworks compared to the other two topologies. In general, the hybrid topology provides *(i)* stability in requests transmission, *(ii)* low memory consumption, and *(iii)* relatively low waiting time. Centralised topology is also shown to be superior in this analysis. However, numerous research studies [48, 105] show the disadvantages of centralised topology, such as lack of fault-tolerance and inefficient extensibility, limit its efficiency in distributed systems. In Chapter 8, we further evaluate the performance of the agent network topologies through conducting a series of agent topological experiments.

CHAPTER 8

EXPERIMENTAL EVALUATION

In the previous chapters, we conducted the theoretical analysis on the agent topological performance and introduced the theory of the agent-ranking algorithm. However, research work in the field should not just be limited to theoretical analysis. Conducting experiments on simple agent network topologies is very necessary for multi-agent systems as research works in this area are insufficient. The experimental simulations of the agent-ranking algorithm are also vital to observe the computational results.

In this chapter, we illustrate the experimental designs and results of the three common agent network topologies as well as the agent-ranking computations. This chapter basically consists of two main parts: the first part mainly focuses on the experimental performance evaluations on the agent network topologies based on the AOCD framework [173]; the second part emphasises the experimental results for the computations of the agent-ranking algorithm.

8.1. AOCD-based Agent Network Topological Experiments

To validate our theoretical analysis, we developed a program, called the *AOCD Topological Experimental Program*, based on the AOCD framework. This is a simulation program based on the previous theoretical analysis [170], and it is designed to collect transmission data for the centralised and hybrid agent network topologies as introduced in our previous work [172, 173]. Figure 8.1 shows the main interface of the AOCD topological experiment program.

A user can enter an initial agent group number to start an AOCD topological experiment. The minimum number of user inputs is 2 and the maximum number of user inputs is 34. The reason we limit the agent group number in this experiment is because of the limitation of the screen space; if there are more than 34 agent groups then the program will not be able to provide an overall demonstration on one screen. We enable users to adjust the agent number in an agent group so that the initial number of agents can be theoretically unlimited. Therefore, the agent group number is

limited (for a total of 34) but it does not limit the total agent number. In most of our experimental sessions we adopted 3 agents in one agent group.

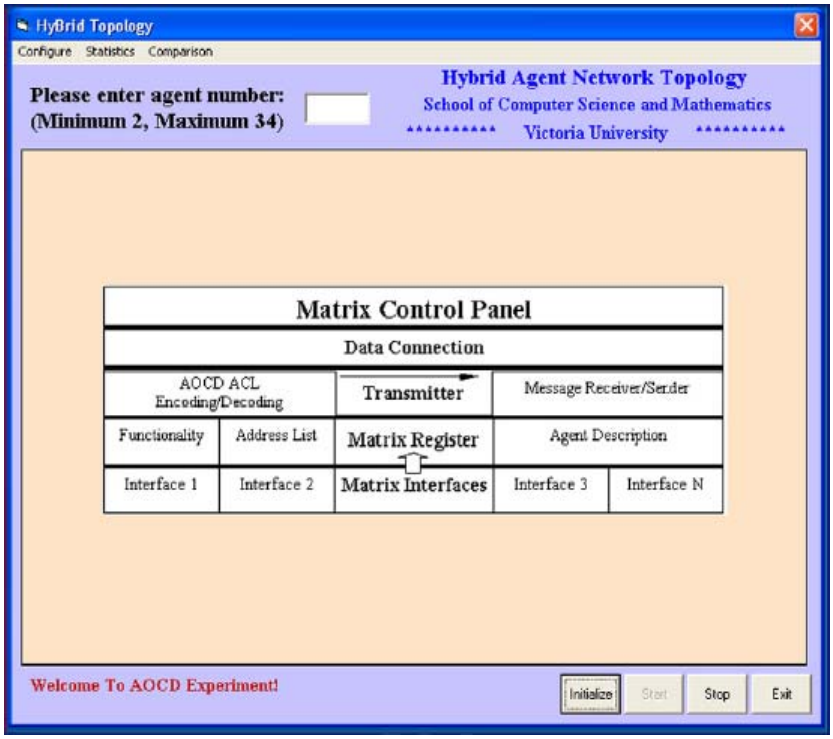


Figure 8.1. The Main Interface of the AOCD Topological Experiments.

8.1.1. Operational Procedures of the AOCD Topological Experiment

The AOCD topological experiment is basically followed by five main procedures (for both the hybrid and centralised designs). These five main procedures are listed as follows.

- **Initialising agent group:** Each agent group contains a number of agents. A user starts this experiment program by entering the initial number of agent group. The number of agents in each group is same and can be configured through the "Configuration Control" function.
- **Generating agents and requests:** After entering the number of agent groups, users can click the "Initialise" button. The system will automatically and randomly generate agents and requests. The generated agent information and requests will be stored into a database.
- **Delivering request to Matrix:** Once the generation of agents and requests is completed, the "Start" button will be activated. When a user clicks the "Start" button, the matching process will start. As shown in Figure 8.2, each agent group contains the same number of agents that connected to the Matrix. For an agent

group that is sending requests to the Matrix, a red connection line will be used to mark the link between this agent and the Matrix. Meanwhile, the requesting agent's colour will also change to red background.

- **Agent matching process:** the agent matching processes for the centralised and hybrid agent network topologies are different. The centralised process involves four transaction processes, which are: (*i*) requesting agents to Matrix, (*ii*) Matrix to corresponding agents, (*iii*) corresponding agents to Matrix, and (*iv*) Matrix to requesting agents. The hybrid process involves 3 transaction processes, which are: (*i*) requesting agents to Matrix, (*ii*) Matrix to corresponding agents, and (*iii*) corresponding agents to requesting agents. The colour of the lines between the agent groups and the Matrix will turn to black when the requests of an agent group have been processed. The matching process of the centralised system can be briefly described as follows: a requesting agent generates a request → send it to the Matrix that searches a corresponding agent based on the Matrix register → delivering the request to the corresponding agent (if available) → the corresponding agent produces the results → send the results back to the Matrix → the Matrix finally delivers the results to the requesting agent. The hybrid system adopts the similar but more efficient process. The corresponding agent sends the results directly back to the requesting agent instead of sending results back to the Matrix.

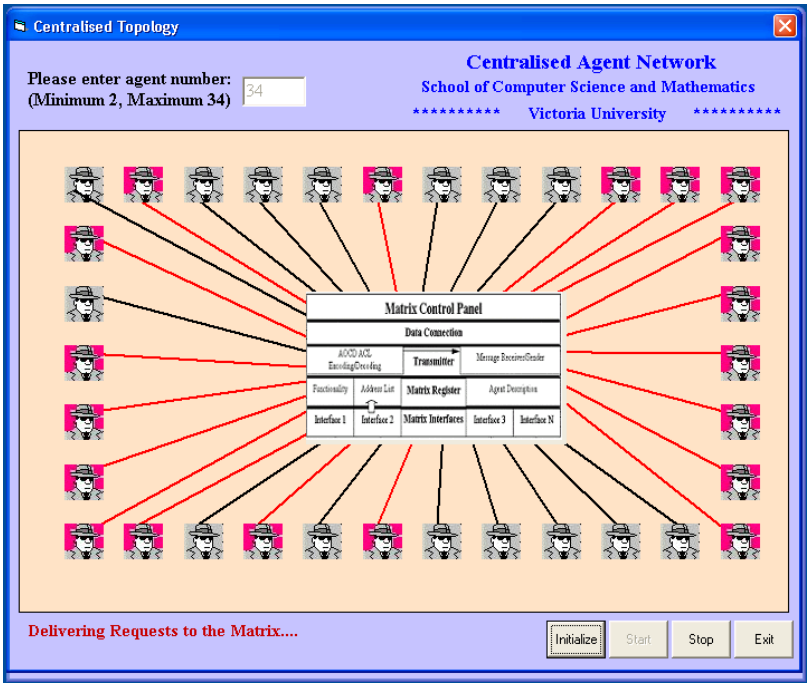


Figure 8.2. Delivering Agent Requests to Matrix (for 34 Agent Groups).

- **Completion of matching process:** The program will send a message once a matching process is completed. Figure 8.3 shows the screen of a completion of an agent matching process for 21 groups. All the connecting lines between agents and Matrix will turn into black when a matching process is completed. However the background colours of the requesting agents still remain the same in order to distinguish the requesting agents from other agents.

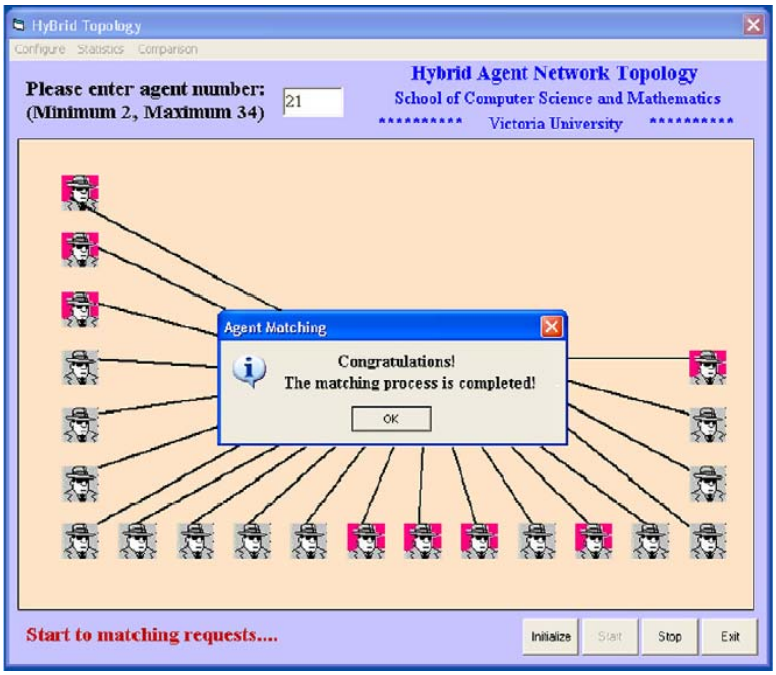


Figure 8.3. Completion of Agent Matching Process (for 21 Agent Groups).

8.1.2. Functionalities and Configurations of the Program

The AOCD topological experimental program provides the functionalities that allow a user to configure the experimental settings.

- **Control commander:**

Control Commander allows a user to configure the experimental parameters including: (*i*) *Matrix Capacity*, (*ii*) Agent number in a group, (*iii*) *Deadlock recovery* parameter, and (*iv*) Transmission interval. Figure 8.4 shows a screen of "Control Commander".

A Matrix can handle a number of requests at any one time. *Matrix Capacity* indicates the number of requests that Matrix can handle at one time. This program also allows users to adjust the agent number in an agent group and it enables more agents can be deployed in the experiment. The transmission interval parameter setting allows users to adjust the transmission time of one transaction. In this experiment, we

assume that all transactions have the same transmission time. The default transmission interval is 1 second.

The *Deadlock recovery* parameter is a unique design in this experiment. We observed that deadlocks occur when two agents request the same corresponding agent at same time or the requests queue forms a loop when requesting agents are waiting for other requesting agents. In this experiment, we use a tag to monitor the requesting queue. If an agent matching process runs through one round and the total requests in the queue do not decrease then the tag will be increased by 1. Once the tag value reaches to the value of the *Deadlock recovery* parameter then we force the release of the first request in the queue and set the tag value to default. The deadlock recovery mechanism affects the success rate of the AOCD topological experiments. The success rate is calculated through the following equation.

$$S = PR/TR \qquad \text{(Eq. 8.1)}$$

where S denotes the success rate, PR denotes the processed requests, and TR denotes the total requests that include the requests which are not processed because of deadlock.

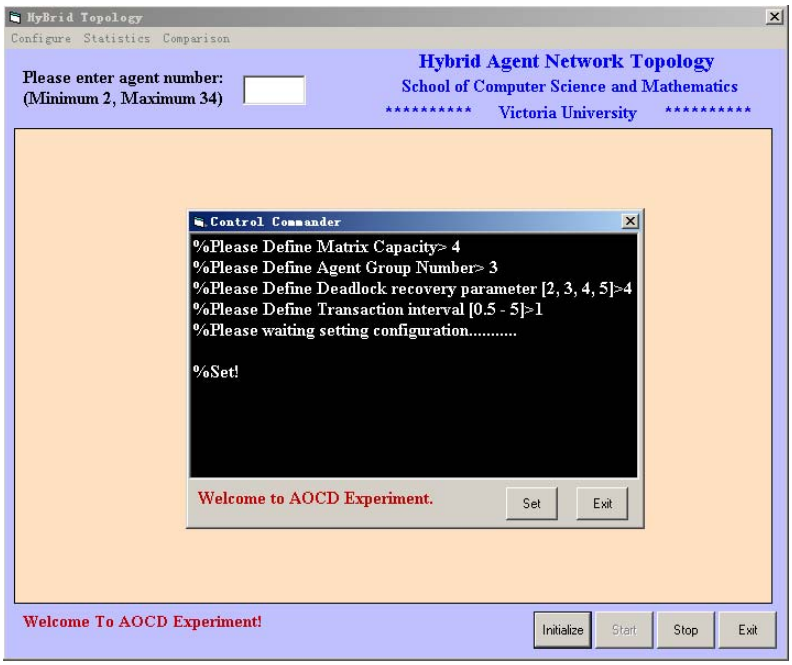


Figure 8.4. Control Commander.

- **Stop button and agent information:**

The "Stop" button allows users to terminate the program and unload all agents from the Matrix. Figure 8.5 shows the termination of a matching process. A user

also can obtain agent information by double clicking on agents. The information includes the agent number and requesting content (if the agent is requesting agent).

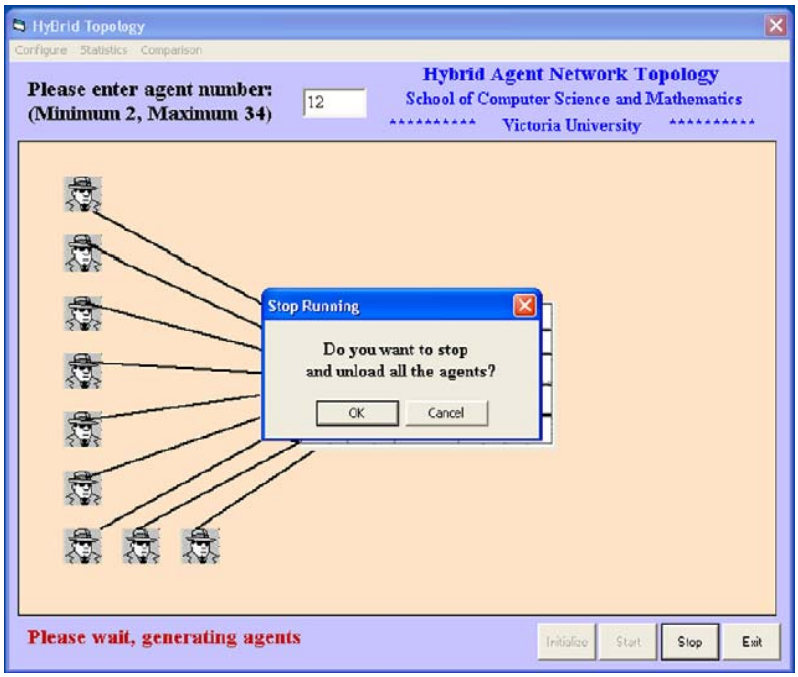


Figure 8.5. Terminating Matching Process.

- **Progress indicator:**

We use a process indicator, as shown in Figure 8.6, to inform the user about the current status of the experiment.

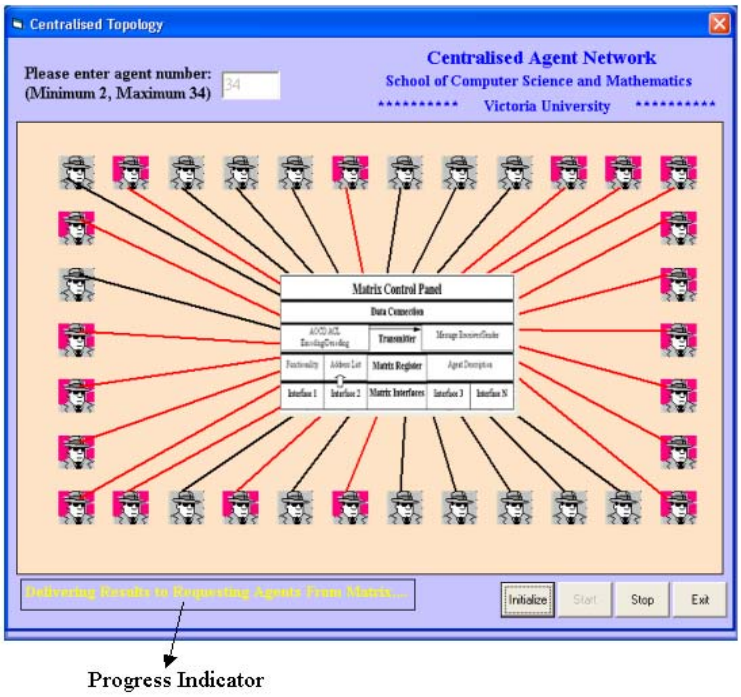


Figure 8.6. Progress Indicator.

8.1.3. Design Methodologies of the Experimental Program

There are 9 major modules both in the hybrid and centralised agent matching process, which include:

- User Input Module;
- Request Generation Module;
- Randomisation Module;
- Connection Module;
- Matrix Control Module;
- Agent Matching Module;
- Requests Delivery Module (Both to/from Matrix);
- Waiting Queue Operation Module;
- Results Delivery Module.

As shown in Figure 8.7, the structure of the AOCD topological experiment program consists of 9 modules. Both the centralised and hybrid designs have the same structure.

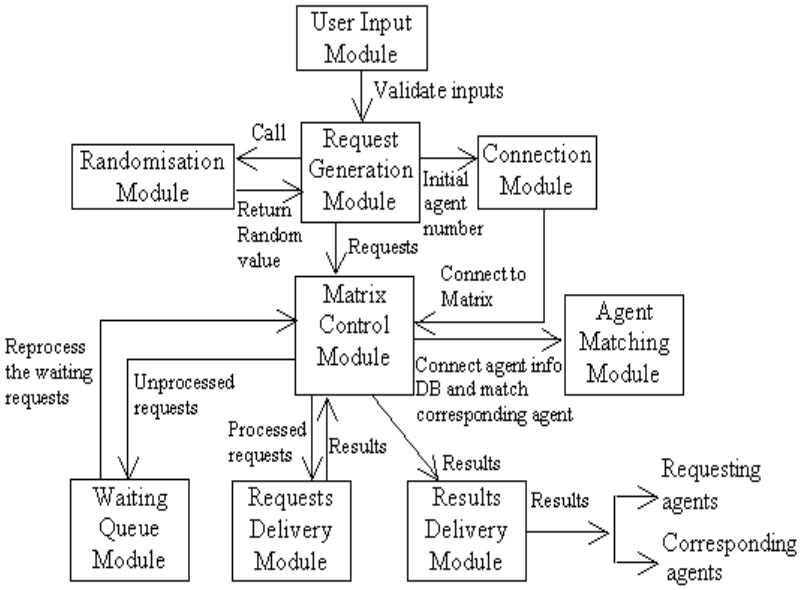


Figure 8.7. Program Structure of the AOCD Topological Experiments.

The *User Input* module mainly deals with receiving users' inputs, validating user inputs, and passing user inputs to the *Request Generation* module. Users can input an agent group number between 2 to 34. The maximum agent group number is limited to 34 only because of the limitation of screen space. If an input agent group number is a very large number, this will cause insufficient screen space for demonstrating all the agents' activities. Although the number of agent groups is limited, the number of

agents in a group is in principle infinite. In other words, we can use a limited number of agent groups to demonstrate an infinite number of agents within one screen.

In the experiments, the request generation module automatically generates agent requests. During the request generation process, the *Randomisation* module generates some random values and returns these values to the request generation module to distinguish different requests. In other words, each request is generated based on the random value. Generated requests are recorded together with other agent information in an agent database.

Once the request generation process is complete, the *Connection* module will be called. Meanwhile, the Requests Generation module delivers all the agent requests to a Matrix request database. The connection module initialises the interface number of the Matrix and calls the *Matrix control* module.

In the Matrix control module, for each transaction time period, the Matrix control module calls the *Requests Delivery* module to deliver *N* requests (*N* is the Matrix's capacity, which indicates the number of requests processed by the Matrix each time). Once the Matrix receives requests, it starts to match the requests by calling the *Agent-Matching* module.

In the agent-matching module, *N* requests in the Matrix database are selected (all the requesting agents' availability attributes are set to 'Unavailable') and all the agents are selected for matching. Once a round of matching processes is completed for *N* requests, the agent-matching module returns the results to the Matrix control module. For the matched requests (where their corresponding agents are available) and for the unmatched requests, the Matrix control module starts to call the *Results Delivery* module. The Results Delivery module delivers all the matched results to their corresponding agents and the unmatched results will be sent to the requesting agents directly. For the matched requests, where the corresponding agents are unavailable, these requests will be passed to the *Waiting Queue* module. The Waiting Queue module delivers all the unprocessed requests to a waiting list. The current matching process is based on a numeric match, which only matches the agent's ID. A numeric-based agent searching mechanism seems to be unrealistic in the real world, as many problem-solving processes in most organisational systems cannot be simply expressed by numbers. Our continuous development of the agent matching process will involve a more advanced agent-matching algorithm [168], such as a semantic agent matching process through adopting the PageRank algorithm mechanism [117].

However, the structure of the AOCD topological experiment will remain the same. This means that the current experimental results are unlikely to be affected because our focus is on the topological issues. The change in the individual matching process might increase or decrease the matching time, however, it should not affect the results of comparing different topologies. In other words, if the agent matching time is increased in the centralised topology by using semantic matching methods, it should also be increased in the hybrid and decentralised topologies by using the same methods.

To explain our design methodology, we illustrate the following Pseudocode in the agent-matching module, which is one of the core modules of the AOCD topological experiment.

Pseudocode of the *agent-matching* module

```
1   READ Processing_Requests_File
2   IF  Requests_Number > 0 THEN  //Requests are not empty.
3      CALL Stop_Sending_RequestsTo_Matrix
4      DO LOOP UNTIL Request_Number = 0
5        FOR  X = 0 to Matrix_Capacity - 1
6          IF Corresponding_ID = RequestID THEN
7            IF Corresponding_availability = 'available' THEN
8              //**Update Requests Records.
9              SET  Request_ProcessValue = 'True'
10             SET  Request_SuccessValue = 'True'
11             SET  Request_Status = 'Processed'
12           ELSE   //** Corresponding agent unavailable.
13             ADD  RequestID to Waiting_Queue
14             SET  Request_ProcessValue = 'False'
15             SET  Request_SuccessValue = 'False'
16             SET Request_Status = 'Unprocessed'
17           END IF
18         ELSE   //** Not found Corresponding agent.
19             //** Update Requests Records.
20           SET  Request_ProcessValue = 'True'
21           SET  Request_SuccessValue = 'False'
22           SET  Request_Status = 'Processed'
23         END IF
24         Request_Number = Request_Number - 1
25         IF  Request_Number = 0  THEN EXIT FOR
26       END FOR
27     END LOOP
28   END IF
29  Call Matrix_Control_Module  //** Matching process completed.
```

In the above algorithm, the 'Request_ProcessValue' is used to identify whether a request has been processed or not; the 'Request_SuccessValue' is used to identify whether a request has been processed successfully or not.

The differences between the designs of the experimental programs for the two topologies are: (*i*) in the centralised agent topology, the results delivery module is called 4 times whereas this module is called 3 times in hybrid agent topology because the centralised topology adopts a four-way communication mechanism and the hybrid topology adopts a three-way communication mechanism (Request agents – Matrix – Corresponding agents – Request agents). (*ii*) Another major difference between the centralised and hybrid design is in the results sending process. In the hybrid design, the corresponding agents will be released after they send their results. In the centralised design, the corresponding agents will not be released until the requesting agents receive the results from the Matrix. The reason for this design is that: in the hybrid agent topology, the results are directly delivered between the requesting agents and the corresponding agents. In other words, once the corresponding agents send their results, this means the requesting agents are going to receive the results and the process is over. On the contrary, the Matrix delivers the results in the centralised agent topology. In other words, the corresponding agents must not be released until the Matrix delivers the results to the requesting agents, which is to ensure the system consistency.

8.1.4. Experimental Results Based On the Program

We conducted a set of AOCD topological experiments to observe the transmission performance of hybrid and centralised agent networks. Our experiments were based on the centralised and hybrid agent systems. We conducted a number of experiments for each system. When all the randomly generated requests (according to the agent group number) are matched then one set of the experiments is completed.

Our experiments were conducted in three different computing systems. 1) MS Windows XP (SP1), CPU PIII 700MHZ; 2) MS Windows Server 2003, CPU AMD 900MHZ; and 3) MS Windows XP (SP1), CPU Celeron 1.5GHZ.

The experiments are divided into three sessions for each topology. Each session contains 100 sets of experiments and the experimental parameters are amended in each session. Therefore, there are a total of 600 sets of the experiments. Based on these experimental results, we found that the average processing time for one request

in the centralised topology is 7.82 seconds, whereas it is 3.63 seconds in the hybrid topology.

Figure 8.8 shows the comparisons of the average time consumptions of processing a request in the hybrid and centralised topologies with different experimental parameters. The following sections illustrate the detailed results of both the topologies.

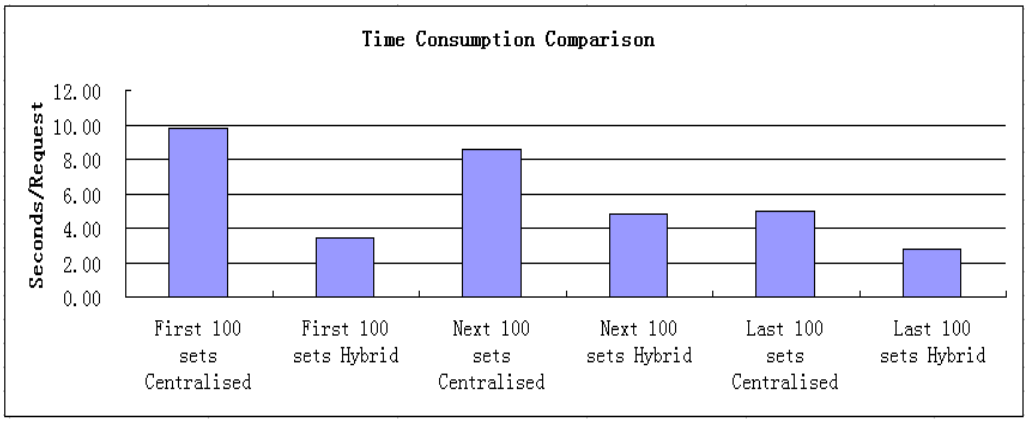


Figure 8.8. Comparisons of the Time Consumption in Hybrid and Centralised Topologies with Different Experimental Parameters.

Results of the hybrid agent network

I) First 100 set of experiments. In the first 100 sets of the experiments, the experimental parameters were listed as follows:

- Matrix Capacity: 3;
- Agent number in a group: 3;
- Deadlock recovery parameter: 4;
- Transmission interval: 1 second.

A total of 2406 requests were produced in 100 sets of the hybrid experiments and the total time consumption for processing these 2406 requests was 8200 seconds. The average time consumption for one request in the hybrid system was 3.408 seconds. The total participating agent number was 5160 and the average success rate for these 100 sets was 95.5%.

In the hybrid agent network, processing time is affected by the success rate as shown in Figure 8.9. We calculated the average processing time for each matching process and measured it by its success rate. We found that when the success rate of a matching process decreases then the average processing time of the matching process increases. In other words, if there is no deadlock in a matching process then the processing time will be much less than where there are some deadlocks. As shown in

Figure 8.10, the total request number also affects the processing time. The processing time increases when the total request number increases. However, the total processed request number does not affect the success rate, as shown in Figure 8.11, and the distributions of the success rates are relatively even among the different request numbers.

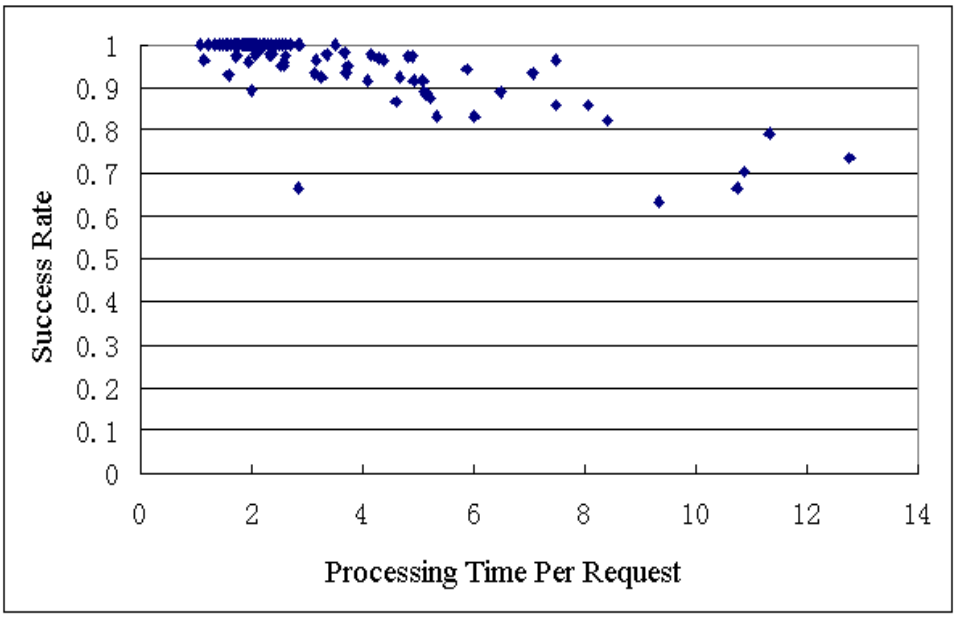


Figure 8.9. Average Processing Time Affected by Success Rate in Hybrid Agent Networks [173].

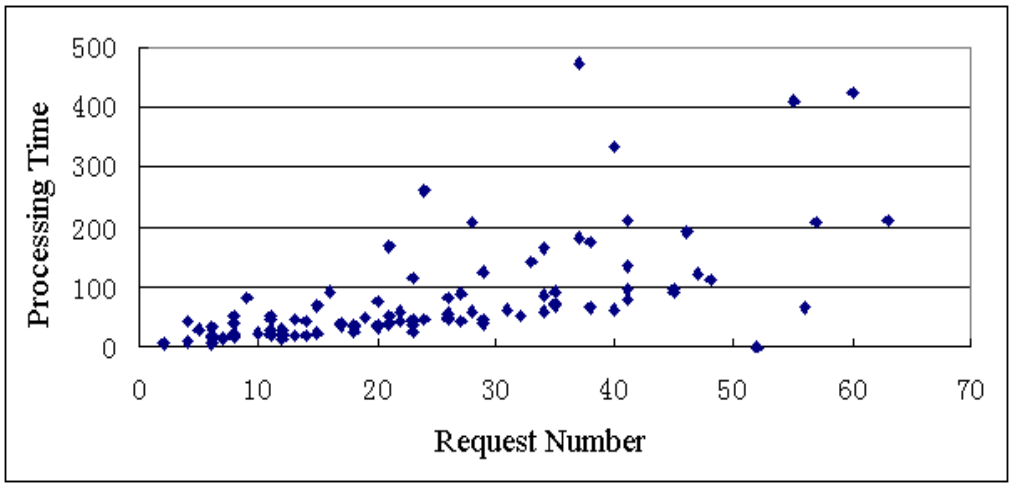


Figure 8.10. Processing Time Affected by Request Number in Hybrid Agent Networks [173].

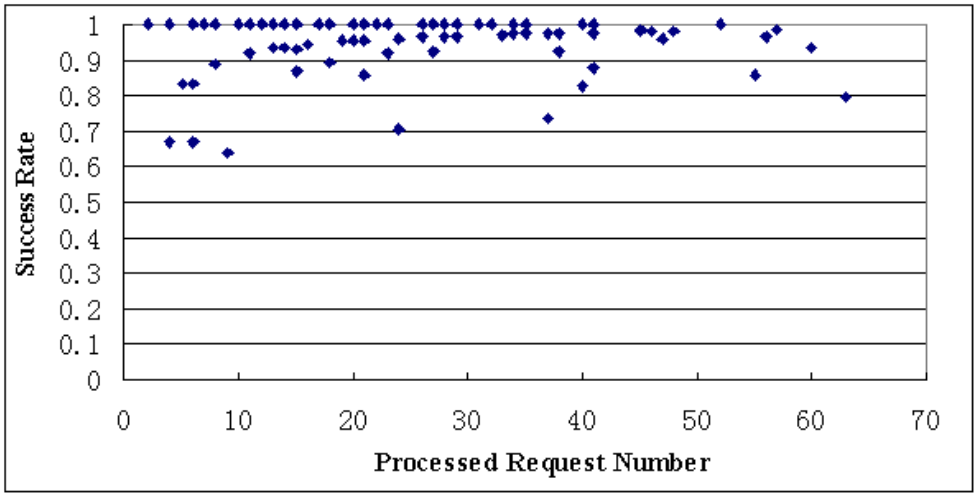


Figure 8.11. Processed Request Number and Success Rate in Hybrid Agent Networks [173].

II) Next 100 Experiments. In the next 100 sets of the experiments, we altered the experimental parameters as follows:

- Matrix Capacity: 3;
- Agent number in a group: 3;
- Deadlock recovery parameter: 2;
- Transmission interval: 1 second.

In total, 2537 requests were generated. The total time consumption for processing these requests was 12246 seconds. The average success rate of the matching process was 92.7%. The total number of the participating agents was 5406.

III) Last 100 Experiments. For the last 100 sets of the experiments, the experimental parameters were amended as follows:

- Matrix Capacity: 5;
- Agent number in a group: 3;
- Deadlock recovery parameter: 2;
- Transmission interval: 1 second.

In total, 2756 requests were generated. The total time consumption for processing these requests is 7538 seconds. The average success rate of the matching process is 94%. The total number of the participating agents is 5748.

Results of the centralised agent network

I) First 100 Experiments. The first 100 sets of the experiments, the experimental parameters are listed as follows:

- Matrix Capacity: 3;
- Agent number in a group: 3;
- Deadlock recovery parameter: 4;
- Transmission interval: 1 second.

A total of 2917 requests were produced by our centralised agent network in 100 sets of centralised experiments and the total time consumption for processing these 2917 requests was 28477 seconds. The average time consumption for one request in the centralised system was 9.762 seconds. The total participating agent number was 5520 and the average success rate for the 100 sets of centralised experiments was 86.7%.

In the centralised agent network, we also discovered that average processing time is affected by the matching success rate. When the success rate of a matching process decreases then the average processing time of the matching process increases. However, the influence of the success rate on the processing time in the centralised agent network is not as strong as it is in the hybrid agent network as shown in Figure 8.12. In some cases, the success rates are low but their average processing time is not much. For instance, the average processing time is 8 seconds when its success rate is 0.5.

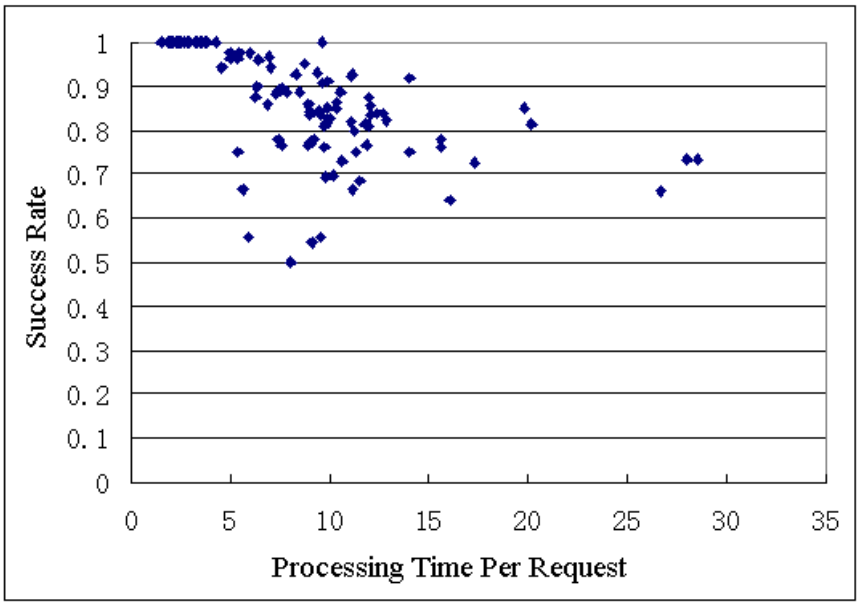


Figure 8.12. Average Processing Time Affected by Success Rate in Centralised Networks [173].

As shown in Figure 8.13, the processing time of the centralised agent network is also affected by the total request number. In general, more requests consume more processing time. However, there are some exceptions in which a high volume of

requests consumes little processing time because the success rate is high. For example, the processing time of 70 requests in Figure 8.13 is 131 seconds (its success rate is 1) compared to 68 requests consuming 1908 seconds in Figure 8.9 (its success rate is 0.735).

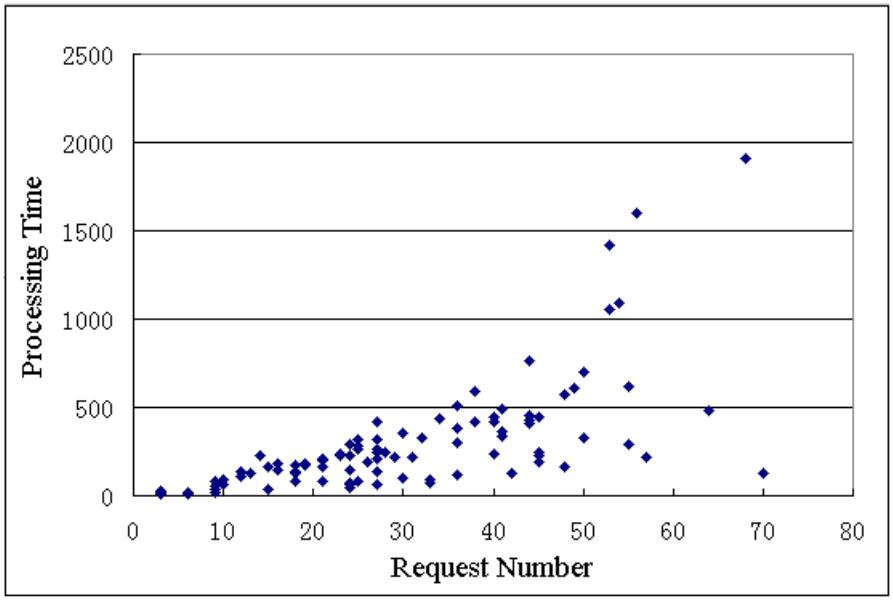


Figure 8.13. Processing Time Affected by Request Number in Centralised Agent Networks [173].

In the centralised agent network, the processed request number also does not affect the success rate because the distributions of processing time based on different success rates are relatively even.

II) Next 100 Experiments. We altered the experimental parameters as follows:

- Matrix Capacity: 3;
- Agent number in a group: 3;
- Deadlock recovery parameter: 2;
- Transmission interval: 1 second.

In total, 2760 requests were generated. The total time consumption for processing these requests is 23735 seconds. The average success rate of the matching process is 85.2%. The total number of the participating agents is 5367.

III) Last 100 Experiments. In the last 100 sets of the experiments, the experimental parameters were amended as follows:

- Matrix Capacity: 5;
- Agent number in a group: 3;

- Deadlock recovery parameter: 2;
- Transmission interval: 1 second.

In total, 2756 requests were generated. The total time consumption for processing these requests was 13772 seconds. The average success rate of the matching process was 85.6%. The total number of the participating agents was 5643.

Centralised Versus Hybrid

The hybrid agent network presents a superior performance in AOCD frameworks compared to the centralised agent network. The average processing time for one request in the hybrid agent network is 3.408 seconds, whereas it is 9.762 seconds in the centralised agent network.

As mentioned in section 7.3.4 of Chapter 7 (page number), "the gradient of the 'worst case' line of the centralised topology is far higher than the other lines, which reflects that the increase in the number of requests would cause much higher transmission time compared to the other two topologies." Although this conclusion is based on the theoretical analysis, our experiments show it is accurate. Figure 8.14 shows the processing time of the centralised agent network is above the processing time of the hybrid agent network on average.

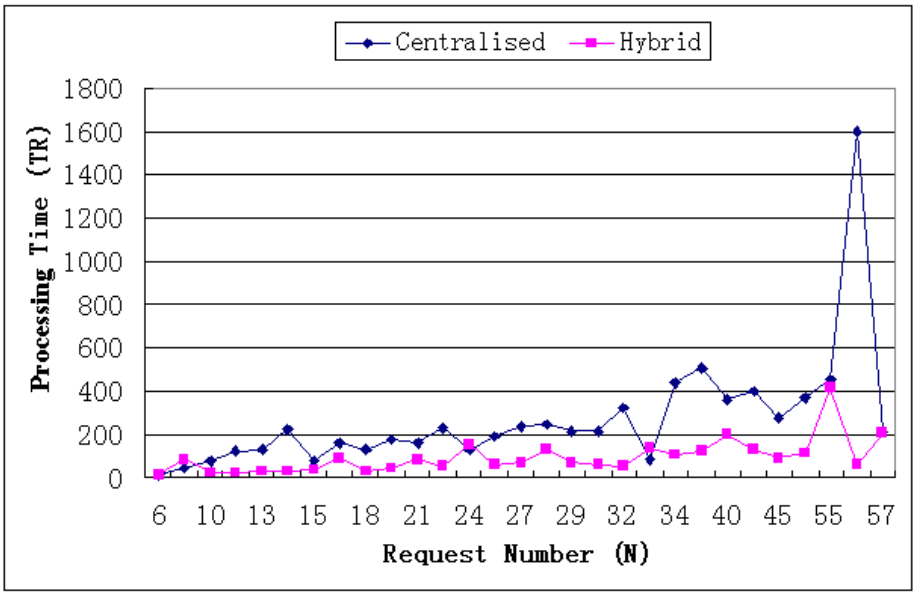


Figure 8.14. Processing Time for Centralised and Hybrid Agent Networks [173].

In section 7.3.4, we also indicate that the transmission (processing) time in both centralised and hybrid networks is relatively stable and the centralised topology is slightly more stable than the hybrid topology. However, it seems hybrid topology is

the most stable topology in Figure 8.14. The reason for increasing the hybrid topology's stability is that we use a deadlock protection mechanism, which eliminates the negative factors caused by deadlock (worst case). Nevertheless, both topologies have very close stability. Our experiments also prove that the theoretical analysis in section 7.3.4 is generally accurate, especially, when we compare Figure 8.14 (in Chapter 8) to Figure 7.16 (in Chapter 7).

To further evaluate various factors affecting topological performance, we concentrate more specifically on the issues of:

(a) The impact of deadlock recovery parameter on transaction time.
(b) The relationship between the success rate of agent matching and (Processed Request Volumes) /(Participated Agents Number).
(c) The impact of the Matrix's processing capacity on the average success rate of agent matching.
(d) The relationship between the Matrix's processing capacity and the request processing time.
(e) The relationship between the processing time and the request volume.

The transmission time and the success rate are the major criteria to evaluate the performance of both centralised and hybrid topologies. The Matrix capacity, deadlock recovery parameter and the request volume are the major factors affecting the processing time and the success rate.

(a) Impact of the deadlock recovery parameter on average processing time

The results show when the deadlock recovery parameter decreases, the processing time consumptions for both centralised and hybrid topologies also decrease. The deadlock recovery parameter is used when a deadlock occurs in the agent matching process. The deadlock recovery parameter determines how many rounds a matching process needs to wait until a request in the waiting queue is forced to complete its task. If an agent matching process runs through one round and the total requests in the queue do not decrease then the deadlock recovery value increases by 1. The increase of the deadlock recovery value indicates a possible deadlock has occurred. Sometimes, an agent might just complete its task and change its status to 'available' but the status change is not updated in time for the matching process. Therefore, we normally wait for 1 or more rounds to ensure all the changes of agent status have been received. This is the reason to use the deadlock recovery parameter.

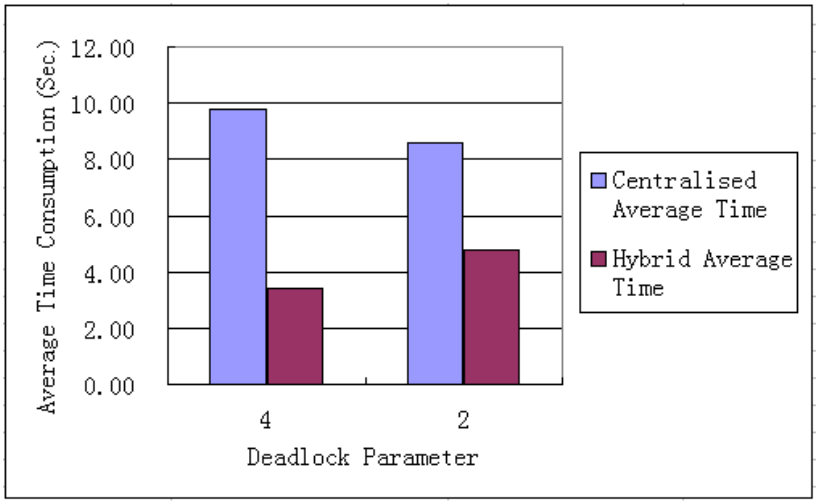


Figure 8.15. Impact of Deadlock Parameter on Processing Time [172].

As we can see from Figure 8.15, when the deadlock parameter decreases from 4 to 2, the average time consumption of the centralised topology drops from 9.76 seconds to 8.60 seconds. It is about a 12% decrease. On the contrary, the average time consumption of the hybrid topology increases from 3.41 seconds to 4.80 seconds when the deadlock parameter decreases from 4 to 2. It is about a 40.8% increase.

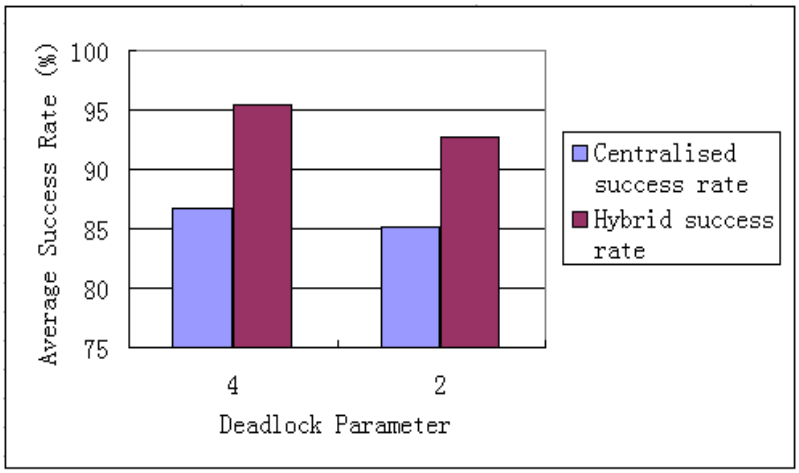


Figure 8.16. Impact of Deadlock Parameter on Success Rate [172].

It indeed violates our prediction, which is that the decrease of the deadlock parameter in all the topologies will decrease the average processing time of all the topologies. It does not seem to be a very logical result, however we find that the average success rate of the hybrid topology decreases from 95.5% to 92.7%, which is a 2.8% decrease. It is only a 1.5% decrease in the centralised topology (see Figure 8.16). We believe this is the main reason why the average processing time of the

hybrid topology decreases. However, the reason for the decrease of average success rate remains uncertain. We expect to solve this question when the experiments for the decentralised topology are completed.

Nevertheless, we predict that the success rate of matching process has a very strong impact on the average processing time. A slight change of success rate would cause a big difference on the results of the processing time.

(b) Relationship between success rate of agent matching and (PRV/PAN)

Before conducting these experiments, we predicted that the products of (PRV/PAN) will more or less impact on the success rate of the matching process. PRV stands for Processed Request Volume, PAN stands for Participated Agent Number. In opposition to the prediction, the experimental results show that the PRV/PAN results are distributed evenly over the success rates axis (see Figure 8.17 and 8.18). However, we found that there are some decreasing trends of PRV/PAN in both centralised and hybrid topologies when success rates are increasing. In other words, when the procedures of PRV/PAN decrease that the success rates are likely to be increasing.

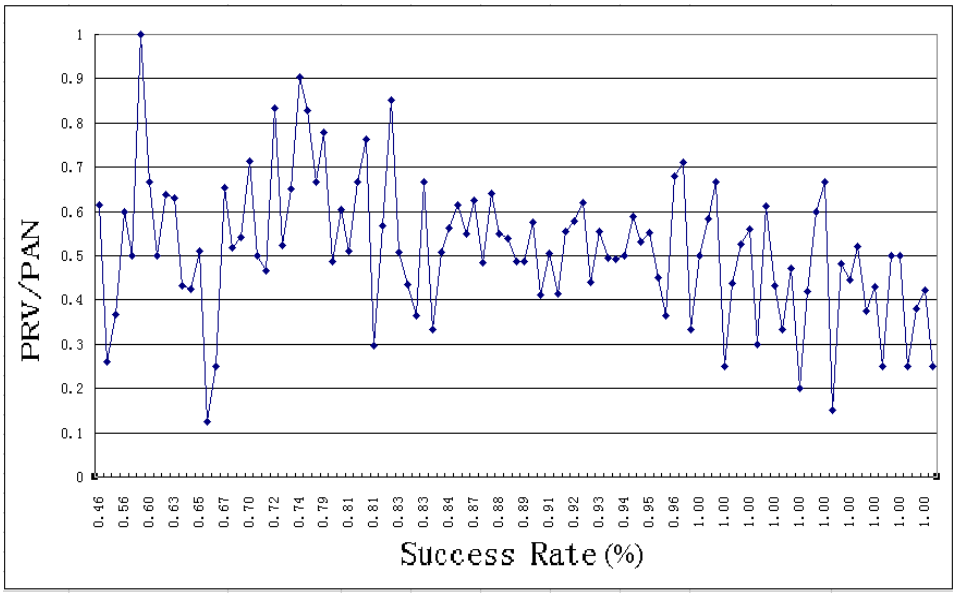


Figure 8.17. Relationship Between Success Rates and (PRV/PAN) in Centralised Topologies (Deadlock = 2) [172].

The success rate of a matching process is calculated as Eq.8.1. As we can see from this equation, the success rate is indeed affected by *PRV*. Logically, the more agents participating in the process and the lesser request volume would help to increase the success rates.

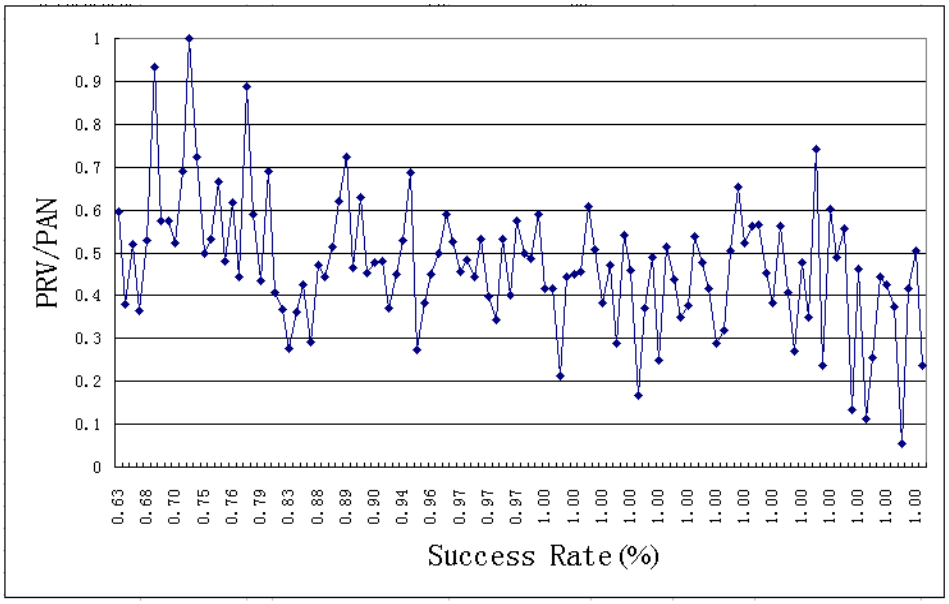


Figure 8.18. Relationship Between Success Rates and (PRV/PAN) in Hybrid Topologies (Deadlock = 2) [172].

(c) Relationship between the Matrix's capacity and the average requests processing time:

The Matrix capacity indicates the number of requests that the Matrix can handle at any one time. In a matching round, the Matrix can only process a certain number of requests. This number denotes the Matrix capacity.

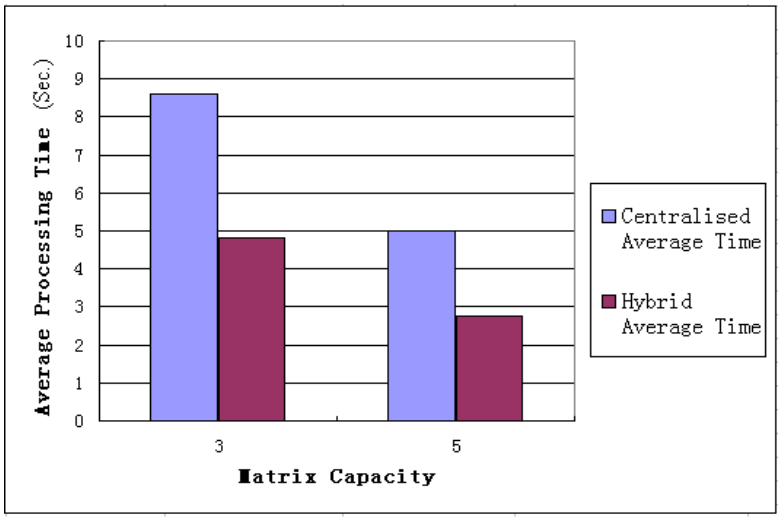


Figure 8.19. Relationship Between the Matrix's Capacity and the Average Requests Processing Time [172].

The Matrix capacity has a strong impact on the average processing time. When the Matrix capacity value is increased and the other parameters remain unchanged (see Figure 8.19), the average processing time decreases in both centralised and hybrid

topologies. The degrees of the decreases in both topologies are very similar, they are 8.6/5 = 1.72 (centralised) and 4.83/2.74 = 1.76 (hybrid), respectively.

These results show the Matrix capacity has relatively equal impacts on both centralised and hybrid topologies. In other words, the average processing time decreases when the Matrix capacity value increases. The impact of the Matrix capacity on the average processing time is similar in both centralised and hybrid topologies.

(d) The impact of the Matrix's processing capacity on the average success rate of agent matching

We change the value of the Matrix capacity and maintain the other parameters unchanged. As shown in Figure 8.20, the increases of the Matrix capacity value in both topologies raise the average success rates in both topologies. However, the increases of success rates in both topologies are slight. There was 0.4% increase in the centralised topology and 1.3% increase in the hybrid topology. We believe the Matrix capacity has an impact on the success rates of matching processes, however its impacts are slight in both topologies.

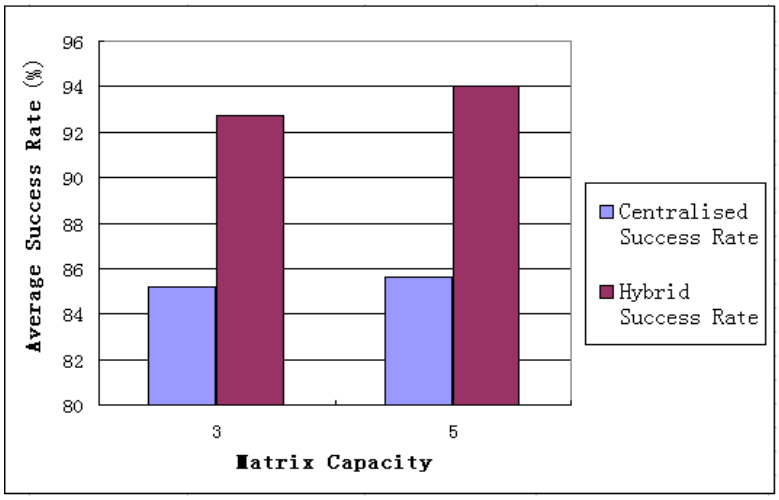


Figure 8.20. Impact of the Matrix Capacity on the Average Success Rate [172].

(e) The relationship between the processing time And the request volume

The request volume has an undoubtedly strong impact on the processing time of matching as we can see from Figures 8.21, 8.22, 8.23 and 8.24. In spite of the different topologies and experimental parameters, the processing time increases rapidly when the request volume is over 50 (approximately).

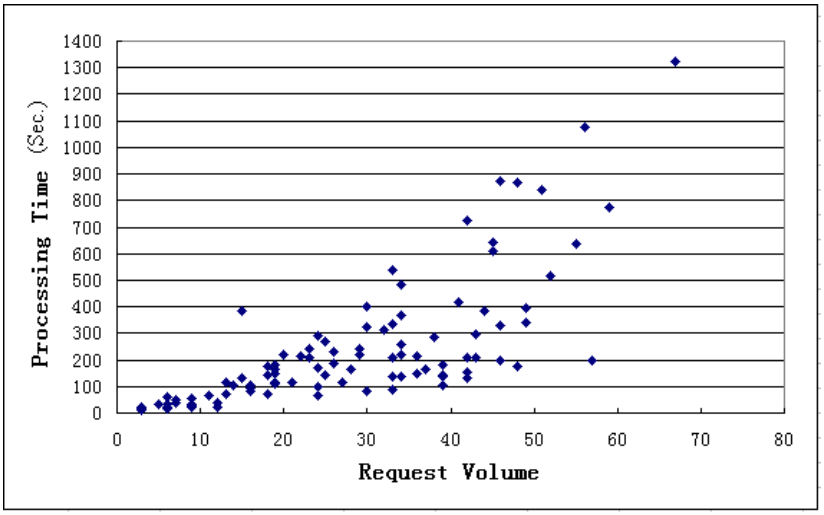


Figure 8.21. Processing Time Affected by Request Volume in Centralised Topologies (DP[1] = 2, MC[2] = 3) [168].

The shapes of the four figures are very similar and emanate from the centres of the coordinates. We believe that the request volume has a strong impact on the request processing time. However, the hybrid topology seems to have a slightly superior performance to the centralised topology. Nevertheless, the impacts of request volume on the both topologies are still relatively equal.

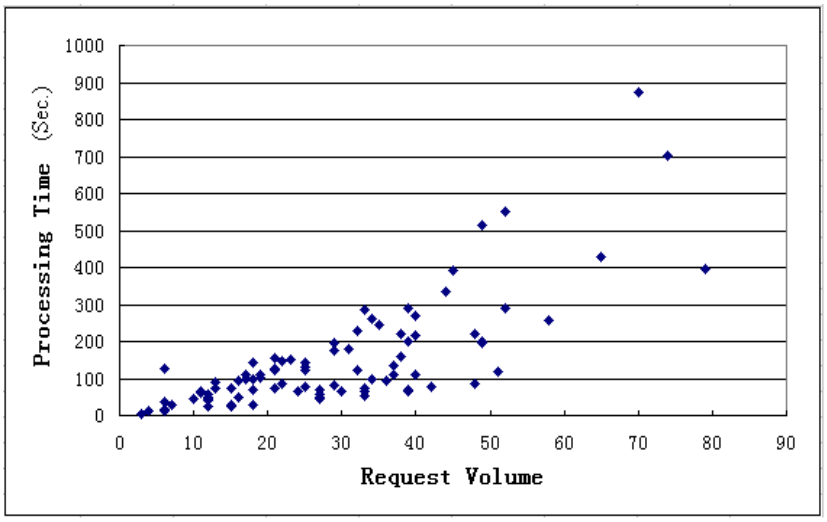


Figure 8.22. Processing Time Affected by Request Volume in Centralised Topologies (DP = 2, MC = 5).

[1]& [2] DP denotes the Deadlock Parameter, MC denotes the Matrix Capacity.

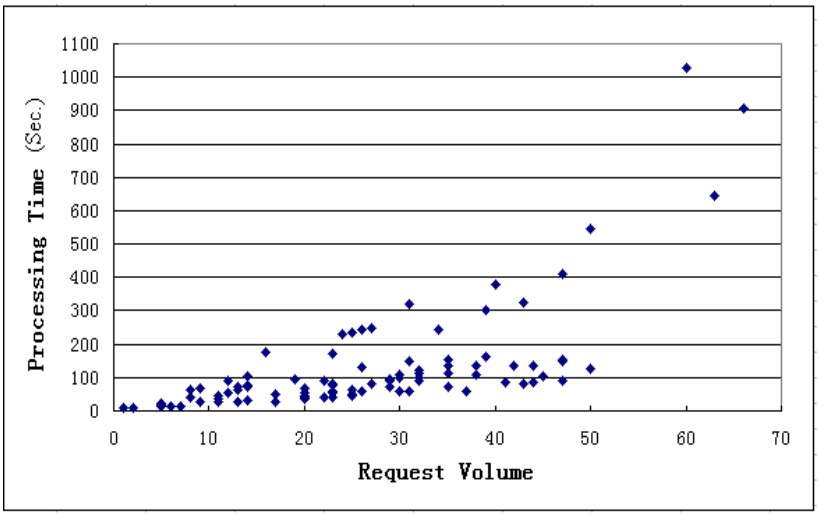


Figure 8.23. Processing Time Affected by Request Volume in Hybrid Topologies (DP = 2, MC = 3) [168].

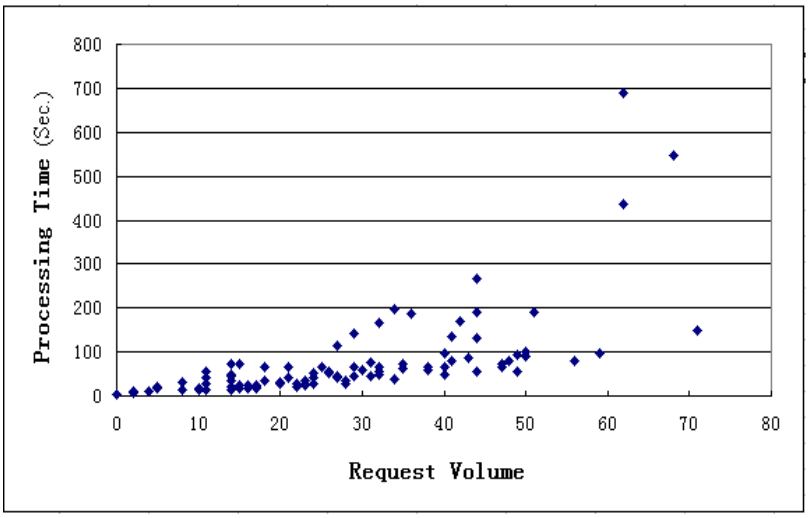


Figure 8.24. Processing Time Affected by Request Volume in Hybrid Topologies (DP = 2, MC = 5).

Overall, the AOCD topological experiments show that the hybrid agent network topology has a superior performance to the centralised agent network topology (see Table 8.1).

Table 8.1. Results of 600 Sets of the AOCD Topological Experiments.

Topology	Matrix capacity	Agent number	Deadlock recovery	Interval (sec.)	Total requests	Total time consumption (sec.)	Average success rate
Centralised	3	3	4	1	2917	28477	86.7%
Centralised	3	3	2	1	2760	23735	85.2%
Centralised	5	3	2	1	2756	13772	85.6%
Hybrid	3	3	4	1	2406	8200	95.5%
Hybrid	3	3	2	1	2537	12246	92.7%
Hybrid	5	3	2	1	2756	7538	94%

8.2. Experimental Computation of Agent-Rank Algorithm

To further explain the general ranking algorithm and the request-based ranking algorithm introduced in Chapter 6, the following two examples are deployed to demonstrate the computation processes. These two examples are based on Figure 8.25. Drawing an *Agent Association Graph* (AAG) is always the first step in analysing the general ranking process. Figure 8.25 illustrates an example to explain the general ranking process. This figure is a Small world topology graph [157] formed by an *R-agent* (Requesting agent) and its *P-agent* (Service provider agent) candidates. R-A denotes the R-agent.

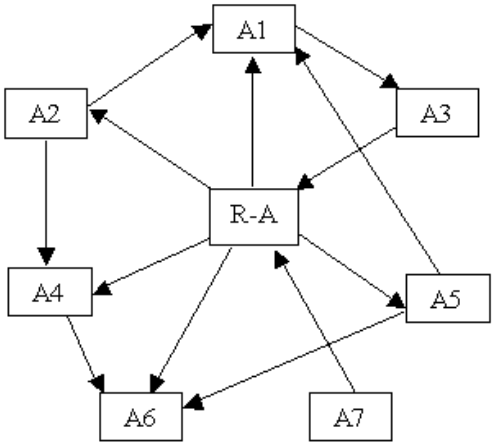


Figure 8.25. A Case-based Agent Association Graph.

8.2.1. Results and Examples of the General Ranking Calculation

The values of the P-agent candidates in Figure 8.25 are listed in Table 8.2. V_i, T_i, and L_i are based on previous simulation values; G_i and C_i are based on Figure 8.25. The meaning of each value can be found in Chapter 6 (Section 6.3).

Table 8.2. The Example Values of the P-agents in An AAG.

P-agent	G_i	C_i	V_i	T_i (sec.)	L_i
A1	1	3	22	60	1
A2	2	1	8	102	3
A3	1	1	- *	-†	-†
A4	1	2	2	396	4
A5	2	1	30	90	2
A6	0	3	11	50	1
A7	1	0	- *	-†	-†
R-agent	5	2	-	-	-

* $f(V_{A3,7}) = 1$ (due to the inbound link from R-agent = 0)

† T_i = average of all the other non-null values. The column L_i gives the number of cooperating agents involved in solving a request for a selected P-agent; the values in this column are given and not explicitly indicated in Fig. 1. For instance, $L_i = 3$ in A2 signifies that: A2 needs three other cooperating agents to solve the request from the R-agent.

The general ranking score of the agent A1 in Fig. 1 can be calculated as follows.

$E = G_{A1...7,R} + C_{A1...7,R} = 26/2 = 13;$

$D = C_{A1} / E = 3/13 = 0.23;$

The coefficient of visit frequency can be calculated as follows:

$f(V_{A1}) = 1 + (V_{A1} / (V_{A1} + V_{A2} + V_{A4} + V_{A5} + V_{A6})) = 1+22/(22+8+2+30+11) = \approx 1.3;$

$f(V_{A2}) = 1.11;\ f(V_{A3}) = f(V_{A7}) = 1;$

$f(V_{A4}) \approx 1.03;\ f(V_{A5}) = 1.411;\ f(V_{A6}) \approx 1.15;$

The average response time can be calculated as:

$T_{avg} = (T_{A1} + T_{A2} + T_{A4} + T_{A5} + T_{A6}) / 5 = 738 / 5 = 147.6;$

Thus, the response qualities of the P-agent candidates are:

$Q_{A1} = (T_{avg} / T_{A1}) \times (0.5 + 1/2 \times L_{A1}) = (147.6 / 60) \times (0.5 + 1/2) = 2.46;$

$Q_{A2} \approx 0.97;\ Q_{A3} \approx 0.73;\ Q_{A4} \approx 0.23;\ Q_{A5} = 1.23;$

$Q_{A6} \approx 2.95;\ Q_{A7} \approx 0.73;$

Table 8.3. Final General Ranking Scores of the Example.

P-agent	General ranking score
A1	5.7329442598609184 ≈ 5.73
A2	1.4505471897424649 ≈ 1.45
A3	1.518395609691344 ≈ 1.52
A4	0.36396377935335605 ≈ 0.36
A5	2.3502480177507725 ≈ 2.35
A6	5.8039935056997995 ≈ 5.80
A7	0.55979000000000001 ≈ 0.56

We have developed a system to calculate the general ranking scores of the above example iteratively. The initial values of all the P-agent candidates are 0. In this example, the preliminary iterations value is 50. This means that for each general ranking process, all the P-agent candidates' general ranking scores are calculated 50 times until there is no further change to all the scores. The loop value 50 is based on the number of total P-agent candidates and previous experience. In the previous

calculation experiments, it takes approximately 50 loops to achieve the stabilisation of ranking scores for 7 participated agents. If the total number of P-agent candidates becomes larger, then the loop value needs to be increased accordingly.

After 50 iterations, all the general ranking scores become stable. Table 8.3 shows the final general ranking scores of the P-agent candidates based on the above example.

We see that A6 ranks highest, followed closely by A1. There is a substantial gap between A1 and the next agent, A5, while the lowest ranking agents are A7 and A4. Figure 8.26 shows the stabilisation in the computation of the general ranking scores, after running 50 iterations. We see the ranking scores stabilise rather quickly after 10 iterations, although some agents (A4 and A7) reach the stable values much sooner.

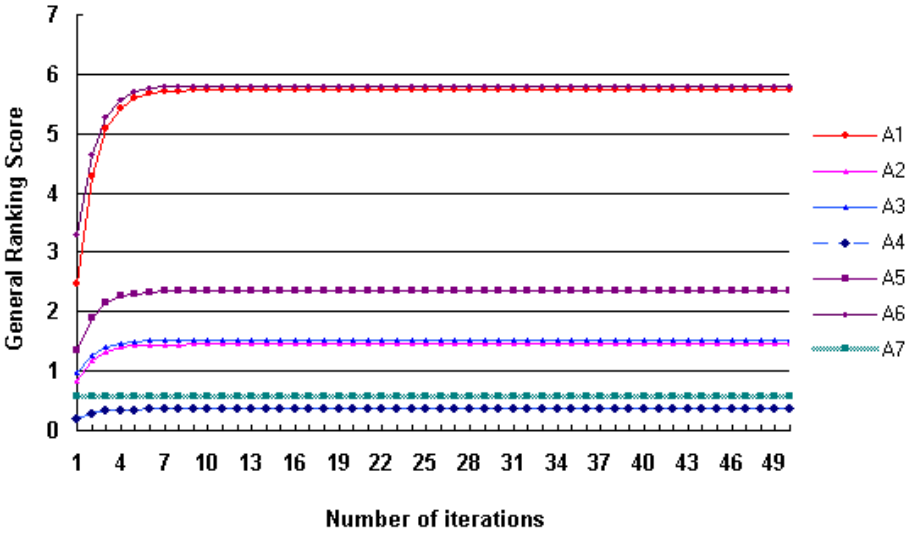


Figure 8.26. 50 Loops of General Ranking Calculations.

The general ranking process can expand to some more complex AAGs such as a quasigroup form of AAG (see Figure 8.27) or other complex network graphs with a central node (R-agent). In spite of the varied forms of AAGs, the calculation process in the general ranking process remains the same. In addition, a two-way link (with inbound and outbound links) between two agents in an AAG is regarded as two links, which include one inbound and one outbound link.

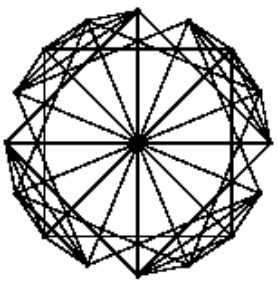


Figure 8.27. A Quasigroup Form of AAG [152].

8.2.2. Results and Examples of the RBR Calculation

The following example illustrates the RBR-based similarity score calculation. Figure 8.28 shows an RBR-based semantic extension tree, which is based on a request from the R-agent (R-A).

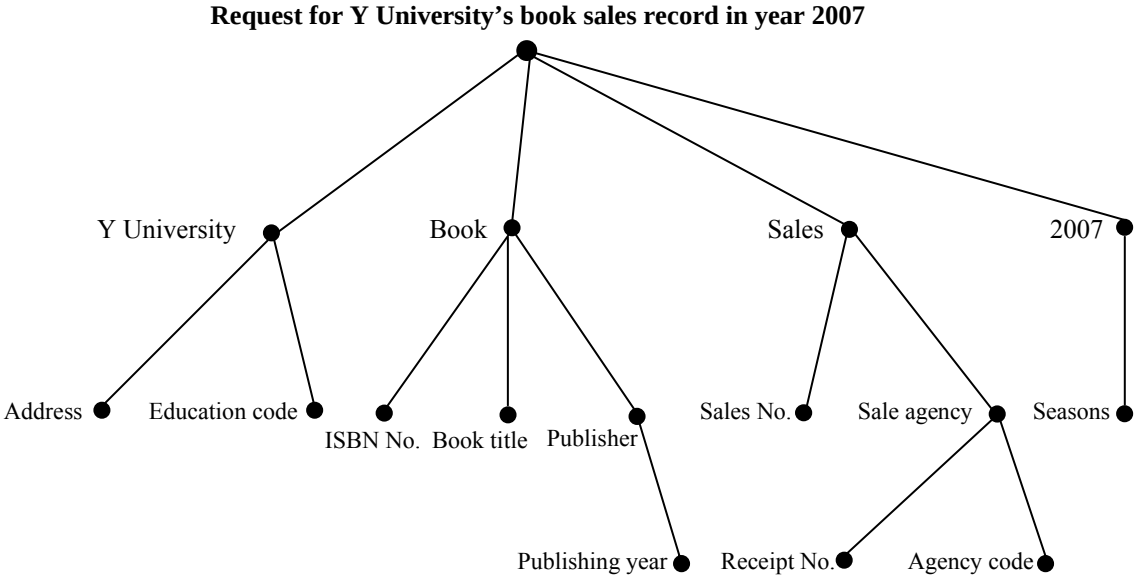


Figure 8.28. An RBR Semantic Extension Tree From R-A.

Continuing the example in Figure 8.25, we have the following agent capability description trees. We only select a few example trees to illustrate the RBR-based similarity calculation process. The example only includes P-agent candidates A1, A4, A6 and A7. In real world, the RBR-based similarity calculation process may involve all the available P-agents. A list of the similarity relationships for these examples is shown in Table 8.4.

Table 8.4. Similarity Relationships Used in the RBR Calculations.

Comparison words	Similarity relationships
Y University, University	IS-a
Y University, Education	IS-a
Y University, Education code	Has-a
Book, Book price	Has-a
Book sales statistics, Book	IS-a
Book, Books price index	Has-a
Book, ISBN index	Has-a
Book circulation, Book	Has-a
Book, Book information	Has-a
Book, Books	Synonym

Book, Publication institution	Has-a
Library, Book	Has-a
New book, Book	IS-a
Old book, Book	IS-a
Sales, Sales amount	Has-a
Sales, Finance	IS-a
Sales, Business & mgmt.	IS-a
Sales, Price	Has-a
Personal details, Address	Has-a
Employee information, Address	Has-a
University, Education code	Has-a
ISBN No., ISBN index	Synonym
Books, ISBN No.	Has-a
New book, ISBN No.	Has-a
Old book, ISBN No.	Has-a
Book title, Book information	IS-a
Sales, Sales No.	Has-a
Sales agency, Business & mgmt.	IS-a
Sales agency, Logistic	IS-a
B-Agency, Sales agency	IS-a
Sales agency, Distribution statistics	Has-a
Sales agency, Staff roster	Has-a
Publisher, Publication institution	Synonym
Publisher, Public service	IS-a
Receipt No., Book price	Has-a
Receipt No., Price	Has-a
Book sales statistics, Receipt No.	Has-a
Distribution statistics, Agency code	Has-a
Book sales statistics, Agency code	Has-a

The example illustrated here aims to explain the RBR-based similarity calculation process; it does not necessarily reflect a real case. In other words, a real case-based agent capability description tree or a real case-based RBR semantic extension tree might vary with the example provided in this paper. Figures 8.29, 8.30, 8.31 and 8.32 show the agent capability description trees of agents A1, A2, A6 and A7, respectively.

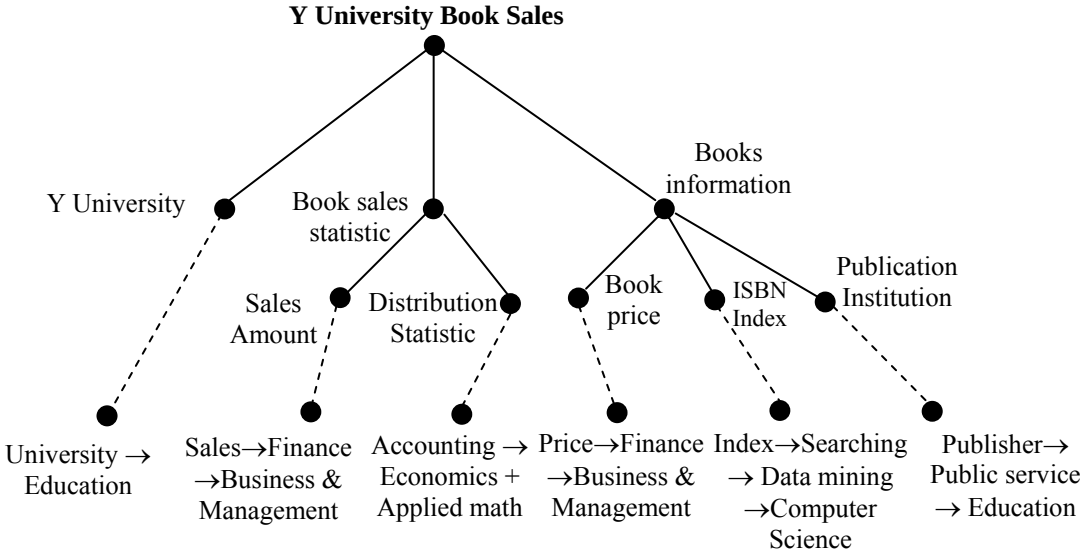


Figure 8.29. Agent Capability Description Tree of P-agent Candidate A1.

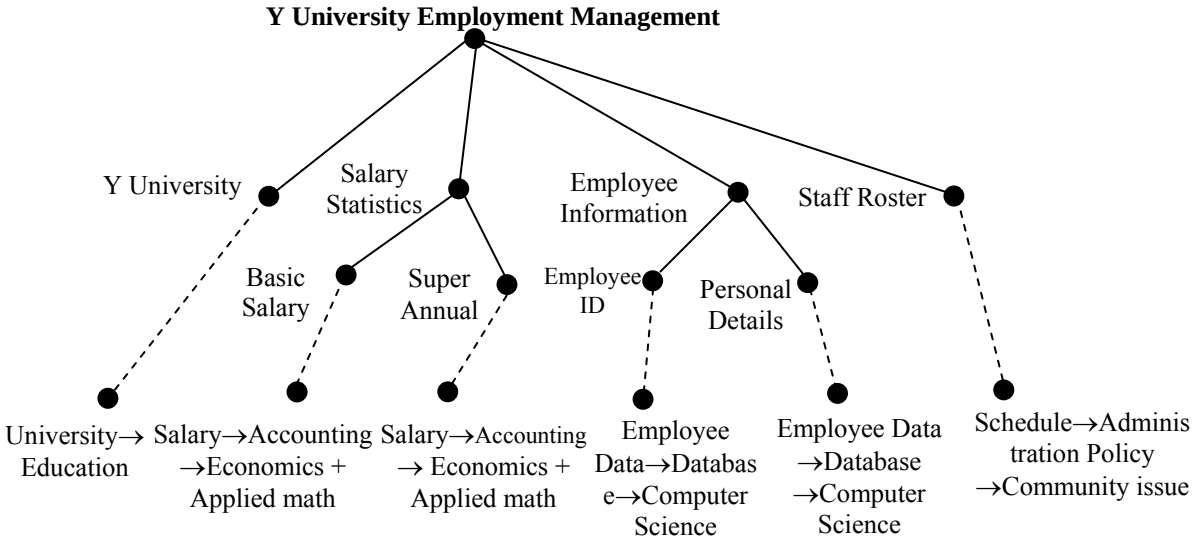


Figure 8.30. Agent Capability Description Tree of P-agent Candidate A2.

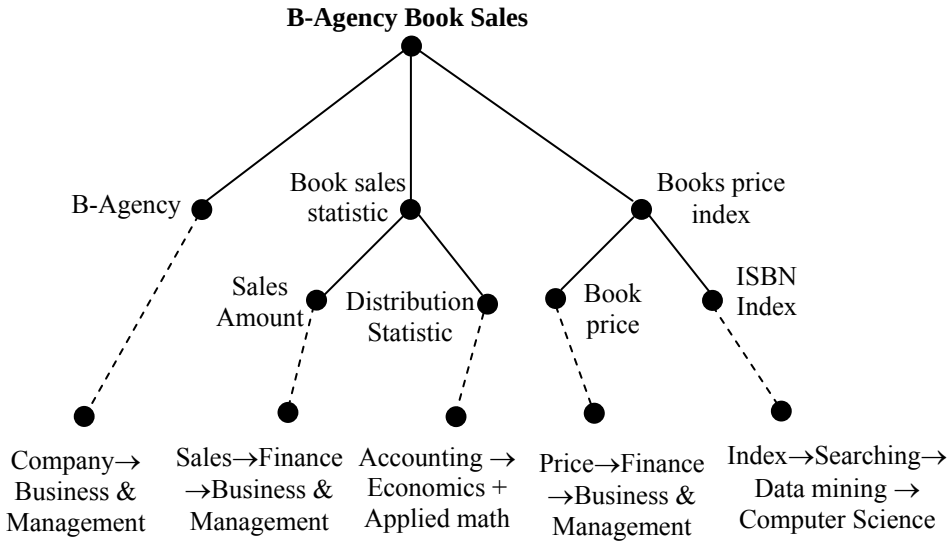


Figure 8.31. Agent Capability Description Tree of P-agent Candidate A6.

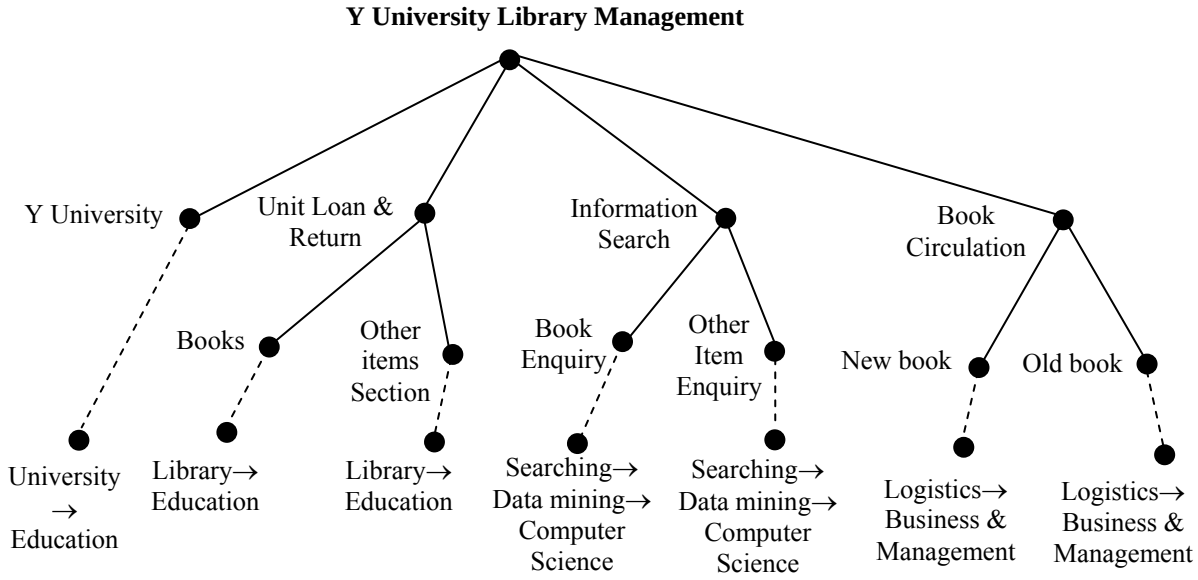


Figure 8.32. Agent Capability Description Tree of P-agent Candidate A7.

We have implemented an experimental system based on the request-based ranking methodology. Figures 8.33 and 8.34 illustrate the GUIs of the computational programs for these two calculation processes. The first step of the program design process, all the agent capability description trees are converted into a database, which

contains the information of each node. In the second step, a request is generated and decomposed for matching process. In the third step, the request is matched with each agent stored in the database through adopting the General-ranking and RBR-based ranking algorithms. The final step incorporates the calculation results generated by the General-ranking and RBR-based ranking algorithms respectively; and selects an agent with highest score as the P-agent.

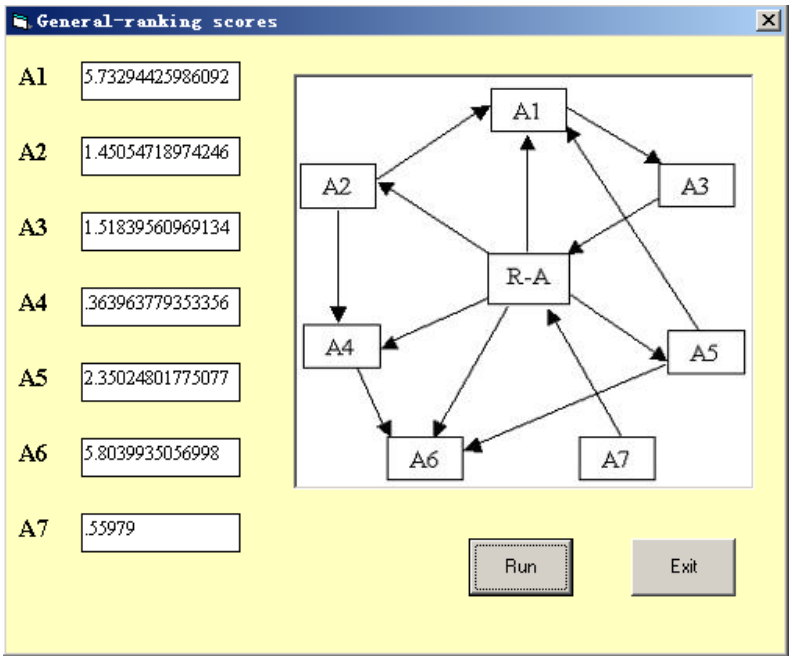


Figure 8.33. The GUI of the Computation Program for the General-ranking Calculation.

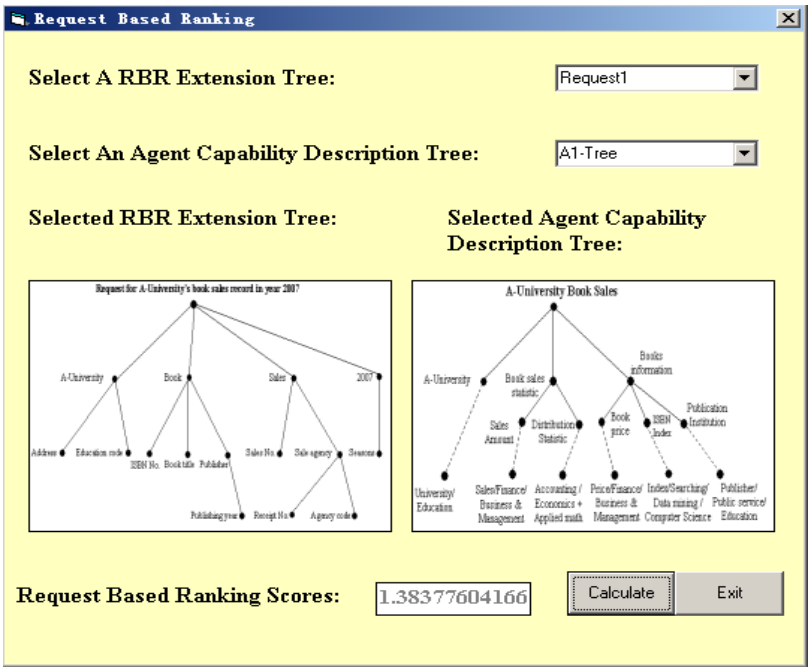


Figure 8.34. The GUI of the Computation Program for the RBR Calculation.

Using the system together with the similarity relationship (see Table 8.4), the RBR-based similarity scores of the above examples in the different levels have been generated and shown in Figures 8.35 to 8.38.

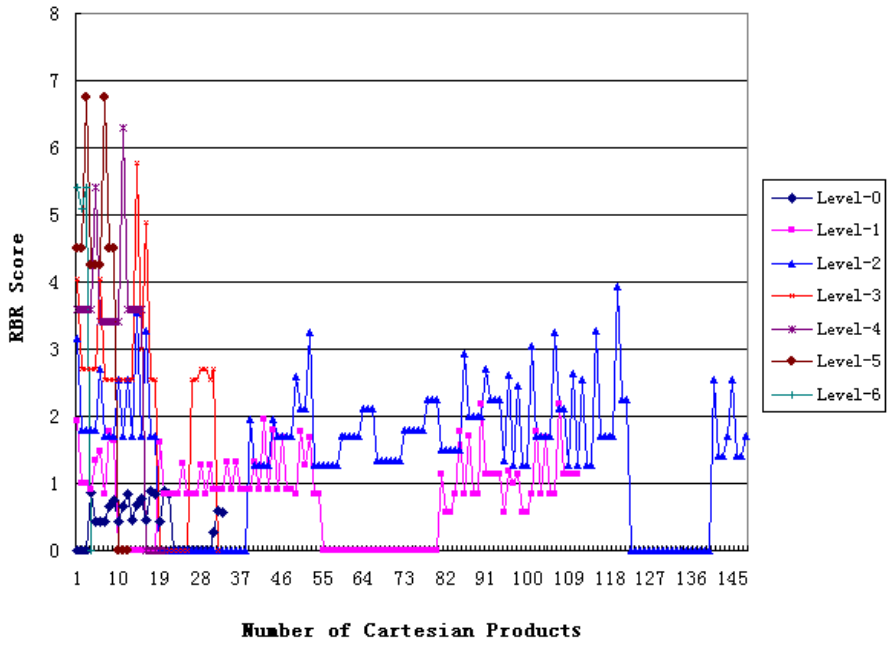


Figure 8.35. RBR Similarity Calculation Scores in Different Levels Between R-A Semantic Tree and P-agent A1 Tree.

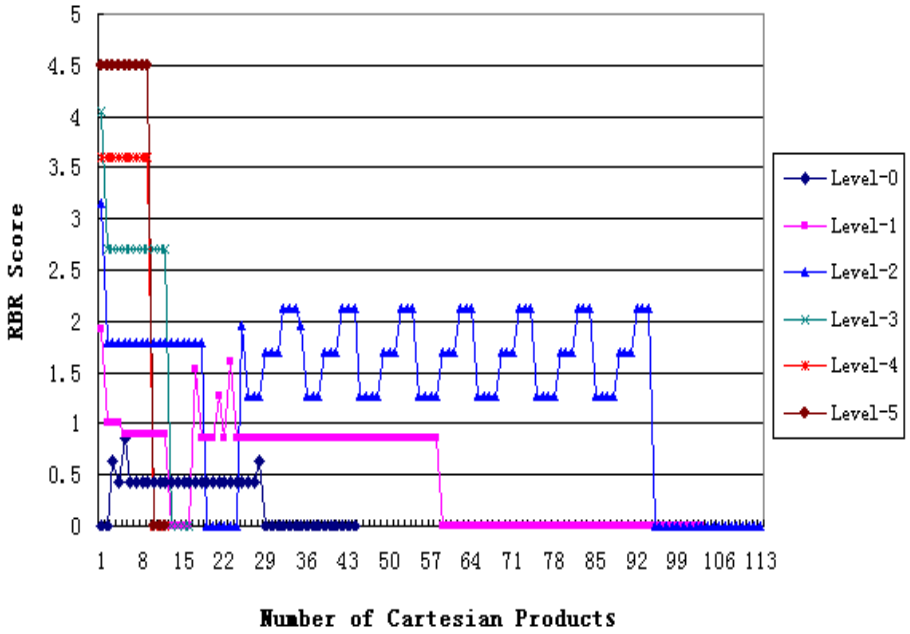


Figure 8.36. RBR Similarity Calculation Scores in Different Levels Between R-A Semantic Tree and P-agent A2 Tree.

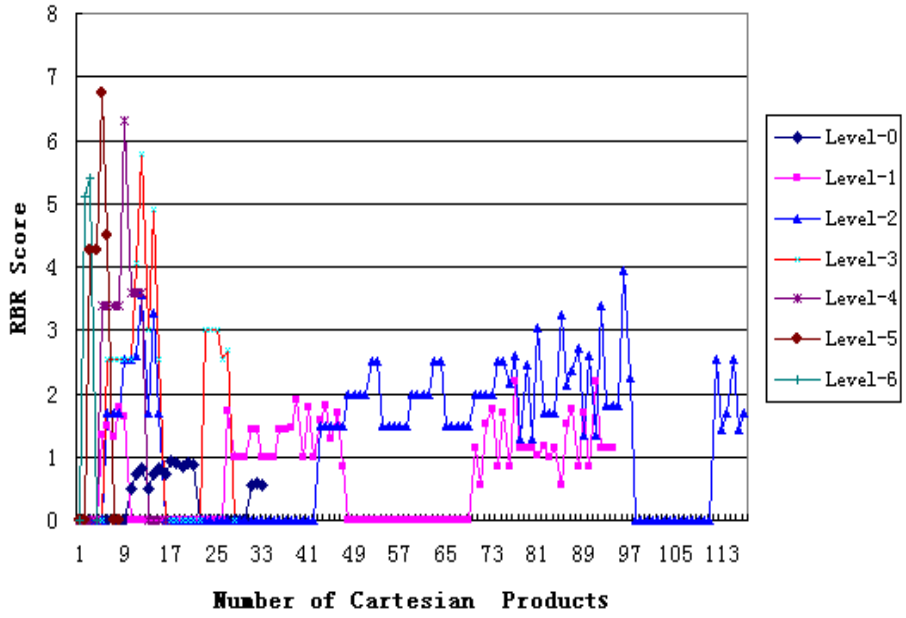


Figure 8.37. RBR Similarity Calculation Scores in Different Levels Between R-A Semantic Tree and P-agent A6 Tree.

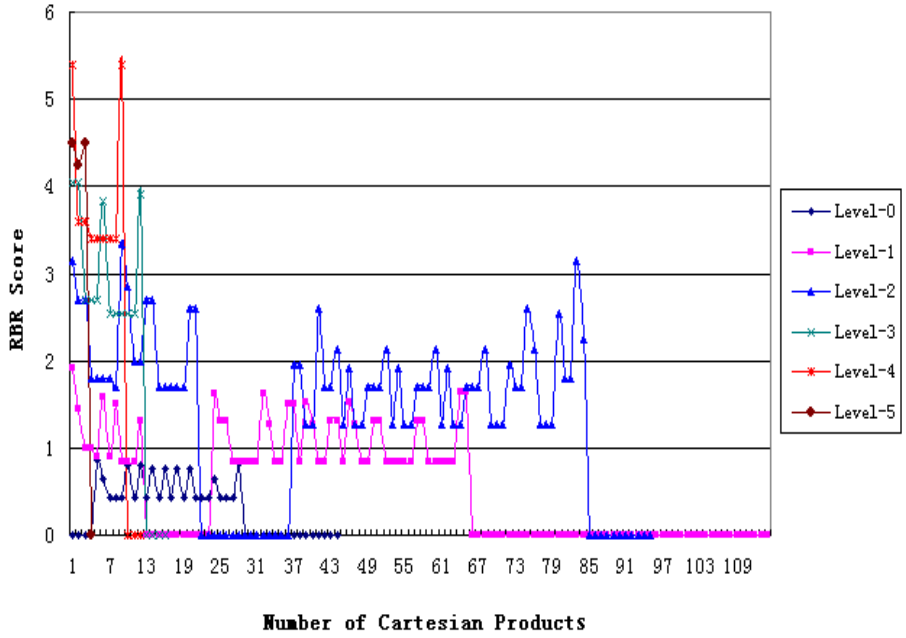


Figure 8.38. RBR Similarity Calculation Scores in Different Levels Between R-A Semantic Tree and P-agent A7 Tree.

The node numbers in different tree levels of the above examples are different and this affects the RBR similarity scores in different levels as shown in Figures 12 to 15. There are more nodes in levels one and two than in the other levels in the above four

P-agent candidate trees. The general scores in Figures 12 and 14 are higher than those in Figures 13 and 15. In other words, agents A1 and A6 offer more relevant functions to solve the request from requesting agent R-A than the other agents. The average scores of the above agents to R-A are listed in Table 8.5 (in descending order of scores):

Table 8.5 RBR Similarity Scores.

Agent Name	RBR similarity score
A1	1.38377604166 ≈ 1.38
A6	1.10300347222 ≈ 1.10
A2	1.06457291666 ≈ 1.06
A7	1.01195175438 ≈ 1.01

Thus, A1 is the most relevant agent in dealing with the request from R-A; A6, the second most relevant agent to R-A; A2, the third most relevant; and A7, the least relevant.

8.2.3. Discussion

The Agent-Rank algorithm consists of the general ranking score and the request-based ranking score. The proportion of each ranking score in the overall ranking score depends on specific applications. In other words, the proportion of both the general ranking score and the request-based ranking score in overall ranking scores can be adjusted according to different needs. For instance, in a large-scale multi-agent system, the general ranking score is likely to receive more weight in the overall ranking score. In general, the overall agent-ranking score can be calculated using the convex combination:

$$AR = \alpha \times GR + (1 - \alpha) \times RS \qquad \text{(Eq. 8.2)}$$

where α ($0 < \alpha < 1$) is the weight assigned to *GR*, which denotes the general ranking score; *RS* denotes request-based ranking score which receives a weight of $(1–\alpha)$; and the *AR* gives the overall agent-ranking score.

Supposing the examples of the previous section are based on a large multi-agent system, then the general ranking score should be assigned more weight (i.e. $\alpha > 0.5$). Using $\alpha = 65\%$, which is the proportion based on the calculation of the total number

of agents and average request number, the scores in the example of the previous section can be summarised as:

$AR1 = 5.73294426 \times 0.65 + (1\text{-}0.65) \times 1.383776 \approx 4.21$,

$AR2 = 1.45054719 \times 0.65 + (1\text{-}0.65) \times 1.064573 \approx 1.32$,

$AR6 = 5.80399351 \times 0.65 + (1\text{-}0.65) \times 1.103003 \approx 4.16$,

$AR7 = 0.55979 \times 0.65 + (1\text{-}0.65) \times 1.011952 \approx 0.72$.

Thus A1 is the most appropriate P-agent to R-agent RA among the four trees, although it has the second highest score in general ranking. A6 has highest score in general ranking but is less relevant to RA in request-based ranking. Overall, A1 is selected as a P-agent.

8.3. Summary

In this chapter, we illustrate and evaluate the results of the AOCD topological experiments and the computation processes of the agent-rank algorithm. The evaluations based on the experimental results provided in this chapter demonstrate the efficiency of the AOCD framework. The experimental results indicate that the system performance is enhanced through adopting the AOCD framework. The current experimental results are based on laboratory simulations. However, these results refer to practical scenarios; and the experimental design is also real case oriented. Therefore, the experimental analysis based on these results can generally and accurately reflect the practical cases to a certain extent.

Through the AOCD topological experiments, we discover three major factors that affect AOCD-based agent network performance. These factors are the Matrix capacity, the deadlock recovery parameter and the request volume for matching. The Matrix capacity and request volume factors basically have equal impacts on both topologies (the hybrid topology has slightly stronger impact). The decrease of the deadlock recovery parameter increases the processing time in the hybrid topology but decreases the processing time in the centralised topology and this result needs to be further evaluated. The average processing time for the hybrid topology is 3.63 seconds and it is 7.82 seconds in the centralised topology. The average success rate of the hybrid topology (94.07%) is also superior to the centralised topology (85.83%). Overall, the hybrid topology presents better performance than the centralised topology.

The process of generating the agent-rank scores consists two procedures including: the general ranking calculation process and the request-based ranking calculation process. Based on the computational results, the P-agent candidate with the highest ranking-score will be selected as the P-agent for a specific R-agent. This ranking process enhances the matching and re-matching processes in a large-scale multi-agent system.

CHAPTER 9
CONCLUSIONS

In this thesis, a novel AOCD framework is suggested in order to accommodate the demands for innovative and effective DSSs. This chapter summarises how the research objectives of this thesis has been attained and discusses the limitations in this research. It also suggests the possible directions for future work.

9.1. Summary of Contributions

The basic structure of the AOCD framework is illustrated in Chapter 3, which consists of the Matrix-agent connection structure (see Figure 3.1) and the unified Matrices structure (see Figure 3.2). In this section, we briefly review the innovations and significant work in this thesis based on the AOCD framework.

An Open System Structure

We developed some new mechanisms to support the open structure of the AOCD framework. This framework deploys intelligent agents as the major units for problem solving processes. It highly modularises the framework and reduces the dependency among sub-systems since agents offer superior flexibility, adaptability, cooperativity, mobility, and reusability. Thus the objective of establishing an open system structure is achieved. To simplify the managing processes of agents, which plague most of the agent-based systems, the Matrix concept is contributed. The Matrix design provides a standardised virtual platform for intelligent agents through managing the Matrix-agent connections in an agent society [168], and the Matrix-agent connection adopts the hybrid structure. These designs further simplify the AOCD framework and improve the connectivity of the framework.

In brief, the AOCD framework provides a flexible plug-and-play architecture, which eliminates the difficulties of integrating different sub-systems or upgrading a DSS.

Efficiency of the AOCD System Structure & Experimental Results

We developed an agent-topology-based experimental program to evaluate the efficiency of the hybrid Matrix-agent connection structure that is employed in the

AOCD framework. The experimental results show that the hybrid agent network topology presents a superior performance in AOCD frameworks compared to the centralised topology. In general, the hybrid topology provides *(i)* stability in requests transmission, *(ii)* low memory consumption, and *(iii)* relatively low waiting time. Experimental results reveal that the overall success rate of processing requests in the hybrid topology is higher than it is in the centralised topology. The hybrid topology in the AOCD framework also highly improves the system manageability compared with the decentralised topology because a central control is adopted in the hybrid topology to tackle the concurrency problem. The Matrix-agent connections also present reasonably superior mobility since the peer-to-peer connections between agents only exist temporarily.

Agent Network Topologies

One of the most significant contributions of this research is that it makes a systematic treatment in clarifying and organising the current topological theory in the multi-agent field. Topological theory in agent networks is an important but somehow underdeveloped research area. Based on the experimental results and theoretical analysis, we systematically classified agent network topologies into two major categories: simple agent network and complex agent network. In general, we can view a complex agent network as an assembly of several simple agent networks. This classification theory provides a fundamental method for helping agent network designers construct an efficient agent network.

Existing mechanisms for describing agent network topologies are graph-based, which are unable to provide detailed information about each agent and its relationship with other agents on a network. Therefore we proposed a new topological description language, i.e. TDLA, for constructing agent networks and performing tasks, such as searching and inserting agents in a network. TDLA offers the intelligent capability of generating a description of an agent network through using statistical results provided by its three sections, which include IAD, MAD and OAD [171]. TDLA enables an individual agent to actively perceive other agents' information and the structure information on the network that the agent resides in. Therefore, an agent can be proactive before performing any actions, such as moving or matching, in an agent network.

The Novel Matrix and Agent Designs

Based on existing agent-based DSSs, the Matrix and agent designs in the AOCD framework utilise several new mechanisms, which include the novel DSC design, the Matrix register, the unified Matrices structure, the Matrix network maintainer, and the Travel Control Centre (TCC) design.

The DSC design allows the AOCD-based agents to perform efficient self-upgrading as well as construct agent capabilities (see Chapters 4 and 5). The DSC usage centre in the DSC design enhances the system reusability. The two-section design in the DSC usage centre improves the efficiency of searching DSC items for agents.

The Matrix register records the information about all the agents that have been plugged into the Matrix (see Chapter 4, Section 4.1.3). The information provided by the Matrix register can be used for agent matching and cooperation. The Matrix network maintainer provides the information about the Matrices' distribution and their functionalities. It has similar functionality as the Matrix register; nevertheless, the Matrix register is mainly used for agent cooperation within a Matrix, but the Matrix network maintainer is used for the cooperation among various Matrices.

The unified Matrices structure enables the cooperation among Matrices (see Chapter 4, Section 4.2). Normally a Matrix (with task agents) represents an AOCD application in an organisation; the unified Matrices structure allows different organisational AOCD applications to cooperate efficiently and economically. Therefore, redundant developments for similar applications can be avoided in the unified Matrices structure.

The TCC design provides the travel information for agents, which allows agents to travel and enter into a remote Matrix (See Chapter 5, Section 5.2.4). The TCC design contains a number of extendable plug-in slots and each slot contains a travel itinerary. The TCC employs the TDLA to generate the travel paths for an agent. This unique design helps AOCD travelling agents acquire the information about the connectivity between origin and destination sites, and the preferable travel routes.

Searching and Matching Mechanisms for Cooperation

Providing an efficient cooperation process for agents and sub-systems is a crucial issue for most multi-agent systems. The AOCD framework deploys two major

searching mechanisms for cooperation within the framework, which include the Matrix search algorithms and the agent-rank algorithm. The Matrix search algorithms (see Chapter 4, Section 4.2.3) consist of the most familiar partner method and the supplemental partner method, which are utilised for searching for a service-provider Matrix according to a requesting Matrix in a unified Matrices structure.

Once a (or several) service-provider Matrix is found, a more specific matching process will be performed to find a specific service-provider agent for cooperation. This matching process deploys the agent-rank algorithm (see Chapter 6), which provides an efficient methodology to match a requesting agent's request within numerous service-provider agents. This algorithm ranks all the service-provider agents according to the requests based on the previous cooperation factors and semantic similarities. A computational program has been developed to calculate the ranking results based on the agent-rank algorithm (see Chapter 8, Section 8.2).

9.2. Discussion and Future Work

9.2.1 Discussion

In view of the fact that current DSSs are becoming large, open and distributed, the application of the Matrix and agents in DSS will undoubtedly have a promising future. The significance of the Matrix design is that it combines centralised communication and decentralised communication in the same framework, which allows the agents to communicate in a more effective way. However the Matrix-agent connections are basically centralised, and this limits the capacity of the Matrix because of the bottleneck problem in centralised structures. The unified Matrices structure helps to overcome the limitations of the Matrix-agent connections through the cooperation among Matrices. In other words, a heavy-loaded Matrix can be decomposed into two (or more) Matrices, and these two Matrices are coordinated through the unified Matrices structure to solve problems cooperatively.

The agents can be deployed to solve semi-structured and unstructured problems because of their intelligent nature and the superior perception of environment; therefore, agents are particularly efficient for DSS applications. Nevertheless, the agent technology is currently underdeveloped; there is a long way to go before agent-based DSSs take over from traditional DSSs.

Overall, the AOCD framework is efficient but with some limitations at this stage. The major limitations of this research study can be summarised as follows:

- The agent network topological experiments are currently based on centralised and hybrid topologies. The analysis based on decentralised topology is considered as a major work for future AOCD research (see Section 9.2.2).
- The standardisation process for the DSC design. The DSC items can be used in different agents, however the current AOCD framework has limited capability to convert various inputs and outputs from different agents into a standardised format.
- The present agent-rank algorithm carries some ranking computation overheads. The agent-rank algorithm not only matches agent capability descriptions but also calculates all the agents' ranking scores. It could affect the overall computation overheads.

9.2.2 Future Work

Future work on the AOCD framework will mainly focus on overcoming the limitations described above. Basically, we outline the following guidelines for the future research on the AOCD framework, which include:

- Consolidating the topological theory based on agent networks, which includes developing an AOCD-based experimental program for the decentralised topology, providing the experimental evidence to evaluate the mobility factors that affect the agent network topologies, and consolidating TDLA for agent travelling.
- Developing comprehensive standards for the Matrix design, including further standardising inputs and outputs for DSC items, formatting the information in the Matrix register, and formatting the information in the Matrix network maintainers.
- Optimising the cooperation mechanisms in the AOCD framework, including discovering possible solutions to reducing the time-consumption of the agent-rank algorithm, and improving the Matrix searching algorithms based on the super-node structure.

In addition to the above guidelines, future work can also conduct a survey on current methodologies in the agent design pattern area to find efficient solutions for agent designs in different situations. It is also important to establish a comprehensive

theory for Matrix and agent development processes (development life-cycle). Meanwhile, investigating emerging technologies and incorporating them into the AOCD framework are very necessary for the development of the AOCD framework in practice. Particularly, the *WAVE-WP* model constructs a system *from the whole* [127] instead of breaking down to self-contained agents and reassembling them as a whole system, which is the common process in the AOCD-based systems and many other component-based systems. This model could avail to accelerate the development process of AOCD-based systems. Moreover, investigating the possible negative impacts of using multi-agent technology in DSSs is also important for future agent development in DSSs.

BIBLIOGRAPHY

1. Agent Oriented Software Ltd. (2005) *JACK™ Intelligent Agents: Agent Manual*, pp. 13 – 59. Agent Oriented Software Pty. Ltd. Publication.
2. Aguirre, C., Martinez-Munoz, J., Corbacho, F., and Huerta, R. (2000) "Small-World Topology for Multi-Agent Collaboration", *Proc. of DEXA2000*, pp. 231-235. IEEE Press.
3. Alter, S. (1999) *Information Systems: a management perspective* (3rd Edition), pp.173 – 177 & 23 – 27. Addison-Wesley Publication.
4. Alter, S. (1977) "A taxonomy of decision support systems", *Sloan Manage. Rev.* Vol. 19, No. 1, pp. 39 – 56. The MIT Press.
5. Androutsellis-Theotokis, S., and Spinellis, D. (2004) "A Survey of Peer-to-Peer Content Distribution Technologies", *ACM Computing Surveys (CSUR)*, Vol. 36, pp. 335 – 371. New York, ACM Press.
6. Angehrn, A.A. (1993) "Computers that Criticize You: Stimulus-based Decision Support Systems", *Interface*, Vol. 23, No. 3, pp. 3-16.
7. Aridor, Y. and Lange, D.B. (1998) "Agent Design Patterns: Elements of Agent Application Design", *Proc. of Autonomous Agents 98*. pp. 108–115. ACM Press.
8. Arnott, D., and Pervan, G. (2005) "A Critical Analysis of Decision support Systems Research", *Journal of Information Technology*, Vol. 20, No. 2, pp. 67-87. Palgrave Macmillan Publication.
9. Arnott, D., and Pervan, G. (2006) "Eight Key Issues for the Decision Support Systems Discipline", in: Adam, F. et al. (eds.) *Creativity and Innovation in Decision Making and Decision Support*, Vol. 2, pp. 946 – 968. Decision Support Press.
10. Barabási, A-L., and Albert, R. (1999) "Emergence of Scaling in Random Networks", *Science*, Vol. 286, pp. 509 – 512. AAAS Publication.
11. Bayardo, R., et al. (1997) "InfoSleuth: Semantic Integration of Information in Open and Dynamic Environments", *Proc. of ACM SIGMOD*, pp. 195 – 206. ACM Press.
12. Bellifemine, F., Caire, G., Trucco, T., and Rimassa, G. (2005) *JADE Programmer's Guide*, pp. 1 – 52. TILab Inc Publication.

13. Bellifemine, F., Poggi, A., and Rimassa, G. (2001) "JADE: A FIPA 2000 Compliant Agent Development Environment", *Proc. of AGENTS'01*, pp. 216 – 217. ACM Press.

14. Benbasat, I., and Zmud, R.W. (1999) "Empirical Research in Information Systems: The Question of Relevance", *MIS Quarterly*, Vol. 23, No. 1, pp. 3–16.

15. Bhavsar, V.C., Boley, H., and Yang, L. (2004) "A Weighted-Tree Similarity Algorithm for Multi-agent Systems in E-business Environments", *Computational Intelligence*, Vol. 20, No. 4, pp. 584-602. Blackwell Publication.

16. Bieszczad, A., Pagurek, B., and White, T. (1998) "Mobile Agents for Network Management", *IEEE Communications Surveys*, Vol.1, No.1, pp.2–9. IEEE Press.

17. Bill, V. (1998) "The First Global Car Colossus", *Business Week*, pp. 40 – 43.

18. Biskupski, B., Dowling, J., and Sacha, J. (2007) "Properties and Mechanisms of Self-Organizing MANET and P2P Systems", *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 2, No. 1, pp. 1 – 34. ACM Press.

19. Bollobás, B. (2001) *Random Graphs* (2nd Edition), pp. 20 – 190. Cambridge University Press.

20. Bradshaw, J.M., Dutfield, S., Benoit, P., and Woolley, J. D. (1997) "KAoS: Toward an Industrial-Strength Generic Agent Architecture", in: Bradshaw, J.M. (eds.), *Software Agents*, pp. 375-418, AAAI/MIT Press, Cambridge.

21. Bratman, M.E. (1987) *Intention, Plans and Practical Reason*, pp. 9 – 20. Harvard University Press, Cambridge, MA.

22. Bratman, M.E., Israel, D.J., and Pollack, M.E. (1988) "Plan and Resource-bounded Practical Reasoning." *Computational Intelligence*, pp. 349 – 355. Blackwell Publication.

23. Brazier, F.M.T., Jonker, C. M., and Treur, J. (2002) "Principles of Component-Based Design of Intelligent Agents", *Data & Knowledge Engineering*, Vol. 41, Issue 1, pp. 1-27, Elsevier Press.

24. Bui, T., and Lee, J. (1999) "An Agent-based Framework for Building Decision Support Systems", *Decision Support Systems*, Vol. 25, No. 3, pp. 225 – 237. Elsevier Press.

25. Burmeister, B. (1996) “Models and Methodology for Agent-oriented Analysis and Design”, in: Fisher, K. (eds.) *Working Notes of the KI’96 Workshop on AOPDS*. DFKI Document.

26. Camacho, D., Aler, R., Borrajo, D., and Molina, J.M. (2005) “A Multi-agent Architecture for Intelligent Gathering Systems”, *AI Communications*, Vol. 18, pp. 15 – 32. IOS Press.

27. Carlsson, C., and Walden, P. (1999) “Intelligent Systems and Active DSS – Modeling Technologies and Intelligent Systems Minitrack”, *Proc. of HICSS99*, pp. 210 – 211. IEEE Press.

28. Chaib-Draa, B. & Dignum, F. (2002) “Trends in Agent Communication Language”, *Computational Intelligence*, Vol. 18, No. 2, pp. 89 – 101. Blackwell Publication.

29. Chi, J., Sun, L., Chen, X., and Zhang, J. (2005) “Architecture and Design of Distributed Decision Support System Based on Agent Grid”, *Proc. of ICMLC05*, pp. 321 – 327. IEEE Press.

30. Chi, R., and Turban, E. (1995) “Distributed Intelligent Executive Information System”, *Decision Support Systems*, Vol. 14, pp. 117 – 130. Elsevier Press.

31. Colloc, J., and Sybord, C. (2003) "A Multi-agent Approach to Involve Multiple Knowledge Models and the Case Base Reasoning Approach in Decision Support Systems”, *Proc. of SSST03*, pp. 247 – 251. IEEE Press.

32. Cuena, J., and Ossowski, S. (1999) “Distributed Models for Decision Support”, in Weiss, G. (eds.): *Multiagent systems-A modern Approach to Distributed Artificial Intelligence*, pp. 459-504, The MIT Press.

33. Depke, R., Heckel, R., and Kuster, J. M. (2001) “Roles in Agent-oriented Modeling”, *Int. Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp. 281 – 302. World Scientific Publication.

34. Deugo, D., Weiss, M., and Kendall, L. (2001) “Reusable Patterns for Agent Coordination.” in: Omicini, A., *et al* (eds.), *Coordination of Internet Agents: models, technologies, and applications*, pp. 347 – 368. Springer-Verlag Press.

35. Dong, C.J., and Loo, G.S. (2001) “Flexible Web-based Decision Support System Generator (FWDSSG) Utilising Software Agents”, *Proc. of DEXA2001*, pp. 892 – 897. IEEE CS Press.

36. Du, H.S., and Jia, X. (2003) "A Framework for Web-based Intelligent Decision Support Enterprise", *Proc. of ICDCSW03*, pp. 958 – 961. IEEE CS Press.

37. Durfee, E.H. and Lesser, V.R. (1989) "Negotiating Task Decomposition and Allocation Using Partial Global Planning", in: Gasser, L., and Huhns, M.N. (eds.), *Distributed Artificial Intelligence, Volume II*, pp. 229 – 243, Morgan Kaufmann Publication.

38. Edwards, J. (2002) *Peer-to-Peer Programming on Groove*, pp. 5 – 69. Addison Wesley Professional Press.

39. Ellis, C., and Wainer, J. (1999) "Groupware and Computer Support Cooperative Work", in: Weiss, G. (eds.), *Multiagent systems – A modern approach to distributed artificial intelligence*, pp. 425 – 457, The MIT Press.

40. Elofson, G., Beranek, P.M., and Thomas, P. (1997) "An Intelligent Agent Community Approach to Knowledge Sharing", *Decision Support Systems*, Vol. 20, No.1, pp. 83 – 98. Elsevier Press.

41. Eom, S.B. (1999) "Decision Support Systems Research: Current State and Trends", *Industrial Management & Data Systems* 99/5, pp. 213 – 220. MCB University Press.

42. Eom, S.B., Lee, S.M., and Kim, J.K. (1993) "The Intellectual Structure of Decision Support Systems (1971-1989) ", *Decision Support Systems*, Vol. 10, No. 1, pp.19 – 35. Elsevier Press.

43. Erdós, P., and Rényi, A. (1960) "On the Evolution of Random Graphs", *publ. Math. Inst. Hung. Acad. Sci.* Vol. 5, pp.17 – 60.

44. Evertsz, R., Fletcher, M., Jones, R., Jarvis, J., Brusey, J., and Dance, S. (2004) "Implementing Industrial Multi-agent Systems Using JACK^{TM} ", *Proc. of ProMAS'03*, Vol. 3067 of LNAI, pp.18 – 48, Springer-Verlag Press.

45. Fellbaum, C. (eds) (1998): *WordNet: A Lexical Database for English*. Cambridge, MA. The MIT Press.

46. Ferguson, I., A. (1992) "TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents." *PhD thesis*, Clare Hall, University of Cambridge, UK.

47. Finin, T., Fritzson, R., Mckay, D., and McEntire, R. (1994) "KQML as an Agent Communication Language", *Proc. of CIKM94*, pp. 456–463, ACM Press.

48. Fiorano Software Inc. (2007) "Super-Peer Architectures for Distributed Computing", *Whitepapers*. Retrieved in August 2007, http://www.fiorano.com/whitepapers/superpeer.pdf.

49. Fischer, K., Chaib-draa, B., Muller, J. P., Pischel, M., and Gerber, C. (1999) "A Simulation Approach Based on Negotiation and Cooperation Between Agents: A Case Study". *IEEE Transaction on Systems, Man and Cybernetics - Part C*, Vol. 29, Issue 4, pp. 531-545. IEEE CS Press.

50. Foulser, D.E., Li, M., and Yang, Q. (1992) "Theory and Algorithms for Plan Merging". *Artificial Intelligence*, 57(2-3), pp.143 – 182. Elsevier Press

51. Foundation for Intelligent Physical Agents. (1997) *Agent Communication Language Specification [FIPA OC00003]*, FIPA 97, Part 2, Version 2.0. FIPA Organisation Publication.

52. Foundation for Intelligent Physical Agents. (2002) *FIPA ACL Message Structure Specification*, FIPA Organisation Publication, Geneva Switzerland.

53. Fowler, J., Nodine, M., Perry, B., and Bargmeyer, B. (1999) "Agent-base Semantic Interoperability in InfoSleuth", *SIGMOD Record*, Vol. 28, No. 1, pp. 60 – 67. ACM Press.

54. Freund, J. E. (1972) *Mathematical Statistics* (2nd Edition), pp. 149, Prentice-Hall Press.

55. Gachet, A. and Haettenschwiler, P. (2003) "A Decentralised Approach to Distributed Decision Support Systems", *Journal of Decision Systems*, Vol. 12, No.2, pp. 141 – 158. Edition Lavoisier Publisher.

56. Gachet, A., and Haettenschwiler, P. (2006) "Development Processes of Intelligent Decision-making Support Systems", in: Gupta, J. N. D., Forgionee, G. A., and Mora, T. M. (eds.) *Intelligent Decision-making Support Systems – Foundations, Applications and Challenges*. pp. 97 – 121, Springer-Verlag Press.

57. Gadomski, A. M., Bologna, S., and Costanzo, G. D. (2001) "Towards Intelligent Decision Support Systems for Emergency Managers: the IDA Approach", *Int. J. Risk Assess. & Man.*, Vol. 2, No. 3/4, pp. 224 – 242. InderScience Publisher.

58. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*, pp. 1 – 29. Addison Wesley Press.

59. Ganesan, P., Garcia-Molina, H., and Widom, J. (2003) "Exploiting Hierarchical Domain Structure to Compute Similarity", *ACM Transactions on Information Systems*, Vol. 21, No. 1, pp. 64 – 93. ACM Press.

60. Georgeff, M. P. and Rao, A. S. (1996) "A Profile of the Australian AI Institute." *IEEE Expert*, Vol. 11, No. 6, pp. 89 – 92. IEEE Press.

61. Gil, Y. and Blythe, J. (2000) "How Can a Structure Representation of Capabilities Help in Planning", *Proc. of AAAI Workshop on Representational Issues for Real-world Planning Systems*. AAAI Press / The MIT Press.

62. Glanzel, W., and Schubert, A. (2003) "A New Classification Scheme of Science Fields and Subfields Design for Scientometric Evaluation Purpose", *Scientometrics*, Vol. 56, No. 3, pp. 357 – 367. Akademiai Kiado & Kluwer Academic Publishers.

63. Glushko, R. J., Tenenbaum, J. M., and Meltzer, B., (1999) "An XML Framework for Agent-based E-commerce", *Communication of the ACM*, Vol. 42, No. 3, pp. 106 – 114. ACM Press.

64. Gomez, M., and Plaza, E. (2004) "Extending Matchmaking to Maximize Capability Reuse", *Proc. of AAMAS*, pp. 144 – 151. ACM Press, New York.

65. Gorry, G. A., and Scott-Morton, M. S. (1971) "A Framework for Management Information System", *Sloan Manage. Rev.* pp. 55 – 65. The MIT Press.

66. Gray, R. S., Cybenko, G., Kotz, D., and Rus, D. (2001) "Mobile Agents: Motivations and State of the Art." *Handbook of Agent Technology*, Bradshaw, J (eds.), TR 2000 – 365, Dartmouth College Publisher.

67. Gupta, J. N. D., Forgionne, G. A., and Mora, M. (eds.) (2006) *Intelligent Decision-making Support Systems: Foundations, Applications and Challenges*, pp. 3 – 24, 97 – 122. Springer-Verlag Press.

68. Halevy, A. (2001) "Answering Queries Using Views: A Survey", *VLDB Journal*, Vo. 10, No. 4, pp. 270 – 294. Springer-Verlag Press.

69. Han, J. and Kamber, M. (2001) *Data Mining: Concept and Techniques*. pp. 428 – 440. Morgan Kaufman Publishers.

70. Hector, A. and Narasimhan, V. L. (2005) *Proc. of ICITA 2005*, pp. 191 – 196, IEEE CS Press.

71. Hein, J.L. (2002) *Discrete Mathematics*, pp. 74 – 140. Jones and Bartlett Publishers.

72. Helsinger, A., Lazarus, R., Wright, W., and Zinky, J. (2003) "Tools and Techniques for Performance Measurement of Large Distributed Multiagent Systems", *Proc. of AAMAS2003*, pp. 843 – 850. ACM Press.

73. Hewitt, C., Bishop, P., and Steiger, T. (1975) "A Universal Mosular Actor Formalism for Artificial Intelligence", *Proc. of IJCAI*, pp. 235 – 245. Morgan Kaufmann Publishers.

74. Hoile, C., Wang, F., Bonsma, E., and Marrow, P. (2002) "Core Specification and Experiments in DIET: A Decentralised Ecosystem-Inspired Mobile Agent System", *Proc. of AAMAS2002*, pp. 623 – 630. ACM Press.

75. Horling, B., and Lesser, V. (2004) "A Survey of Multi-agent Organisational Paradigm", *The Knowledge Engineering Review*, Vol.19, Issue 4, pp. 281 – 316. Cambridge University Press.

76. Huhns, M.N., and Stephens, L.M. (1999) "Multiagent Systems and Societies of Agents", in Weiss, G. (eds.): *Multiagent systems-A modern Approach to Distributed Artificial Intelligence*, pp. 111 – 114. The MIT Press.

77. Javanovich, M.A. (2000) "Modelling Large-scale Peer-to-peer Networks and A Case Study of Gnutella", *Master thesis*, Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati.

78. Jennings, N.R. (1993) "Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems", *The Knowledge Engineering Review*, Vol. 2, No. 3, pp. 223 – 250. Cambridge University Press.

79. Jennings, N.R., Sycara, K. and Wooldridge, M. (1998) "A Roadmap of Agent Research and Development", *Autonomous Agents and Multi-Agent Systems*, Vol. 1, pp. 7 – 38. Springer – Verlag Press.

80. Jin, X., and Liu, J. (2003) "Agent Network Topology and Complexity", *Proc. of AAMAS'03*, pp. 1020 – 1021, ACM Press.

81. Kamali, K., Ventura, D., Garga, A., and Kumara, S.R.T. (2006) "Geometric task decomposition in a multi-agent environment", *Applied Artificial Intelligence*, Vol. 20, No. 5, June 2006, pp. 437 – 456. Taylor and Francis Publisher.

82. Kamel, M., and Syed, A. (1994) "Multiagent task planning method for agents with disparate capabilities", *Int. J. Adv. Manuf. Technol.*, Vol. 9, No. 6, pp. 408 – 420. Springer-Verlag Press.

83. Karwaczynski, P., & Kwiatkowski, J. (2005) "Analysis of Overlay Network Impact on Dependability", *Proc. of HICSS05*, pp. 290 – 299. IEEE CS Press.

84. KazaA Inc. (2006) "How Peer-to-Peer (P2P) and Kazaa Software Works", Retrieved on 19 August 2007. http://www.KazaA.com.

85. Keen, P. (1987) "Decision Support Systems: the Next Decade", *Decision Support Systems*, Vol. 3, No. 3, pp. 253 – 265. Elesvier Press.

86. Keen, P.G.W., and Scott-Morton, M.S. (1978) *Decision Support Systems: An Organizational Perspective*, pp. 79– 93, 167–185. Addison-Wesley Publishing.

87. Kinny, D., Georgeff, M., and Rao, A. (1996). "A Methodology and Modeling Technique for Systems of BDI agents", in: van der Velde, W. & Perram, J. (eds.) *Agents Breaking Away. Proc. of MAAMAW*, pp. 56–71. Springer-Verlag Press.

88. Kroenke, D. and Hatch, R. (1994) *Management Information Systems*, McGraw-Hill Publication, Watsonville, USA.

89. Kuhn, N. (1993) "Comparing Ranking of Heterogeneous Agents", *Proc. of COOCS*, pp. 1 – 12. ACM Press.

90. Kuhn, N., Muller, H.J., and Muller, J.P. (1993) "Task Decomposition in Dynamic Agent Societies". *Proc. of International Symposium on Autonomous Decentralised Systems*, pp. 165 – 171. IEEE Press.

91. Kuokka, D., and Harada, L. (1995) "Matchmaking for Information Agents", *Proc. of IJCAI*, pp. 672 – 678. Morgan Kaufmann Publishers.

92. Labrou, Y. and Finin, T. (2000) "History, State of the Art and Challenges for Agent Communication Languages", *Informatik – Informatique*, Vol. 1, pp. 17 – 24. Swiss Federation of Information Processing Societies Publisher.

93. Labrou, Y., Finin, T., and Peng, Y. (1999) "Agent Communication Languages: The Current Landscape." *Intelligent Systems and Their Applications*, Vol. 14, Issue 2, pp. 45 – 52. IEEE Press.

94. Lee, J.H., and Park, S.C. (2003) "Agent and data mining based decision support system and its adaptation to a new customer-centric electronic commerce", *Expert Systems with Applications*, Vol. 25, pp. 619 – 635. Elsevier Press.

95. Li, X., Montazemi, A.R., and Yuan, Y. (2006) "Agent-based Buddy-finding Methodology for Knowledge Sharing", *Information & Management*, Vol. 43, pp.283-296. Elsevier Press.

96. Li, Y., Bandar, Z. A., and McLean, D. (2003) "An approach for measuring semantic similarity between words using multiple information sources", *IEEE Transaction on Knowledge and Data Engineering*, Vol. 15, No. 4, pp. 871 – 882. IEEE CS Press.

97. Luo, Y., Liu, K., and Davis, D.N. (2002) "A Multi-Agent Decision Support System for Stock Trading ", *IEEE Network*, Jan/Feb, pp. 20 – 27. IEEE Press.

98. Maes, P. (1995) "Artificial life meets entertainment: life like autonomous agents", *Communications of the ACM*, Vol. 38, No.11, pp. 108–114. ACM Press.

99. Maher, M.L. and Gu, N. (2002) "Design Agents in Virtual Worlds: A User-centred Architecture Agent", in Gero, J.S. (eds.): *Proc. of WAID02*, pp. 23 – 38. The MIT Press, Cambridge, MA.

100. Maraks, G. M. (1999) *Decision Support Systems in the Twenty-first Century*, pp. 396, Prentice Hall Publication.

101. Martin, D.L., Cheyer, A.J., and Moran, D.B. (1999) "The open agent architecture: a framework for building distributed software systems", *Applied Artificial Intelligence*, Vol. 13 No. 1-2, pp. 91–128. Taylor & Francis Publisher.

102. Martin, F.J., Plaza, E., and Rodriguez-Aguilar, J.A. (2000) "An Infrastructure for Agent-Based Systems: An Interagent Approach," *International Journal of Intelligent Systems*, Vol. 15, pp. 217 – 240. Wiley InterScience Press.

103. Mayfield, J., Labrou, Y. and Finin, T. (1996) "Evaluation of KQML as an Agent Communication Language", in: Wooldridge, M., Mueller, J. P., and Tambe, M. (Eds.), *Intelligent Agents II: Agent Theories, Architectures, and Language*, Vol. 1037, pp. 347 – 360. Springer-Verlag Press.

104. Miller, P., and Cummins, M. (2000) *LAN Technologies Explained*, pp. 6 – 9. Butterworth-Heinemann Publication, U.S.A.

105. Minar, N. (2002) "*Distributed system topologies*". Retrieved on 20 July 2007, http://www.openp2p.com/lpt/a/1461.

106. Moulin, B., and Brassard, M. (1996) "A Scenario-based Design Method and An Environment for the Development of Multiagent Systems", in: Lukose, D., & Zhang, C. (eds.) *Proc. of DAI95*, pp. 216 – 231. Springer-Verlag Press.

107. Muller, J.P. (1997) "A Cooperation Model for Autonomous Agents", in: Muller, J.P., Wooldridge, M., and Jennings, N.R. (eds.), *Intelligent Agent III*, pp. 245 – 260. Springer-Verlag Press.

108. Nickles, M., and Weiss, G. (2003) "Empirical Semantics of Agent Communication in Open Systems", *Proc. of the 2nd International Workshop on Challenges in Open Agent Environments* (Position paper).

109. Nodine, M., Fowler, J., et al. (2000) "Active Information Gathering in InfoSleuth™", *International Journal of Cooperative Information Systems*, Vol. 9, Nos. 1-2, pp. 3 – 27. World Scientific Publisher.

110. Nodine, M., Ngu, A.H.H., Cassandra, A., and Bohrer, W.G. (2003) "Scalable Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth ™", *IEEE Transaction on Knowledge and Data Engineering*, Vol. 15, No. 5, pp. 1082 – 1098. IEEE CS Press.

111. Norling, E. (2004) "Folk psychology for human modelling: Extending the BDI paradigm", *Proc. of AAMAS*, pp. 202 – 209. ACM Press.

112. Oaks, P., ter Hofstede, A.H.M., and Edmond, D. (2003) "Capabilities: Describing What Services Can Do", *Proc. of ICSOC 2003*, LNCS 2910, pp. 1-16, Springer-Verlag Press.

113. OECD. (2004) "Information Technology Outlook 2004: Peer to Peer Networks in OECD Countries", *OECD Information Technology Outlook2004*, Chapter 5.

114. Ossowski, S., Fernandez, A., Serrano, J. M., Perez-de-la-Cruz, J.L., Belmonte, M.V., Hernandez, J.Z., Garcia-Serrano, A.M., and Maseda, J. M. (2004) "Designing Multiagent Decision Support System-The Case of Transportation Management," *Proc. of AAMAS*, pp. 1469 – 1469. ACM Press.

115. Padgham, L. and Lambrix, P. (2000) "Agent Capabilities: Extending BDI Theory", *Proc. of the 17th AAAI*, pp. 68 – 73. AAAI Press & The MIT Press.

116. Padgham, L., and Winikoff, M. (2002) "Prometheus: A Methodology for Developing Intelligent Agents", *Proc. of AAMAS*, pp. 37 – 38. ACM Press.

117. Page, L., Brin, S., Motwani, R., and Winograd, T. (1998) "The PageRank Citation Ranking: Bringing Order to the Web", *Technical report*, Stanford University, Stanford, CA.

118. Paolucci, M., Niu, Z., Sycara, K., Domashnev, C., Owens, S., and Velsen, M.V. (2000) "Matchmaking to Support Intelligent Agents for Portfolio Management", *Proc. of AAAI*, pp. 1125 – 1126. AAAI Press / The MIT Press.

119. Parameswaran, M., Susarla, A., and Whinston, A.B. (2001) "P2P Networking: An Information-Sharing Alternative", *IEEE Computer*, Vol. 34, No.7, pp. 31 – 38. IEEE Press.

120. Patil, R.S., Fikes, R.E., et al. (1997) "The DARPA Knowledge Sharing Effort: Progress Report", *Reading in Agents*, Huhns, M., and Singh, M. (eds.), pp. 243 – 254. Morgan Kaufmann Publishers.

121. Penner, R.R. (1998) "Automating User Interface Design", *Proc. of IEEE Int Conf Syst Man Cybern*, Vol. 1, pp. 1032 – 1037. IEEE Press.

122. Piliouras, T.C. (2005) *Network Design: Management and Technical Perspective (2nd ed.)*, pp.141-196. CRC Press LLC, U.S.A.

123. Raikundalia, G.K., and Zhang, H.L. (2006) "Document-related Awareness Elements in Synchronous Collaborative Authoring", *Australian Journal of Intelligent Information Processing Systems (AJIIPS)*, Vol. 9, No. 2, pp. 41-48.

124. Ripeanu, M., and Foster, I. (2002) "Mapping the Gnutella Network", *IEEE Internet Computing*, pp. 50 – 57. IEEE Press.

125. Russell, S. and Norgiv, P. (1995) *Artificial Intelligence: A Modern Approach*, pp. 31 – 49. Prentice-Hall Publication.

126. Salton, G. and Buckley, C. (1988) "Term-weighting approaches in automatic text retrieval", *Inf. Process. Manage*, Vol. 24, No. 5, pp. 513–523. Pergamon Press.

127. Sapaty, P. (2005) *Ruling Distributed Dynamic Worlds*, pp. 1 – 30. John Wiley & Sons Publication.

128. Sarit, K. (2003) "Strategic Negotiation for Sharing A Resource Between Two Agents". *Computational Intelligence*, Vol. 19, No. 1, February 2003, pp. 9 – 41. Blackwell Publication.

129. Sen, A., (1995) "From DSS to DSP: A Taxonomic Retrospection", *Publication of Lowry Mayes College of Business administration*, A & M University Texas.

130. Sen, A., Vinze, A., and Liou, S. T. (1994) "Role of control in the model formulation process", *Info. Syst. Res*. Vol.5, No. 3, pp. 219–248. Informs Press.

131. Shehory, O., Sycara, K., Chalasani, P., and Jha, S. (1998) "Agent Cloning: An Approach to Agent Mobility and Resource Allocation." *IEEE Communications Magazine*, Vol. 36, No.7, pp. 58 – 67. IEEE CS Press.

132. Shim, J. P., Warkentin, M. Courtney, J. F., Power, D. J., Sharda, R., and Carlsson, C. (2002) "Past, Present, and Future of Decision Support Technology", *Decision Support Systems*, Vol. 33, pp. 111 – 126. Elsevier Press.

133. Skype. (2005) "Skype Guide for Network Administrators", Retrieved on 19 August 2007.

134. Sowa, J.F. (eds.) (1991) *Principles of Semantic Networks – Explorations in the representation of knowledge*. Morgan Kaufman Publishers.

135. Strogatz, S.H. (2001) "*Exploring complex networks*", *Nature*, Vol. 410, pp. 268 – 276. Macmillan Magazines Publication.

136. Subrahmanian, V., et al. (2000) *Heterogeneous Agent Systems*, pp. 43 – 59. The MIT Press.

137. Sycara, K., and Widoff, S. (2002) "LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace", *Autonomous Agents and Multi-agent Systems*, Vol. 5, pp. 173 – 203. Kluwer Academic Publishers.

138. Sycara, K., Klusch, M., and Widoff, S. (1999) "Dynamic Service Matchmaking Among Agents in Open Information Environments", *SIGMOD Record*, Vol. 28, No. 1, pp. 47 – 53. ACM Press.

139. Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N. (2003) "Automated discovery, interaction and composition of Semantic Web services", *Journal of Web Semantics*, pp. 27 – 46. Elsevier Press.

140. Tari, Z. and Fry, A. (2001) "Controlling Aggregation in Distributed Object Systems: A Graph-Based Approach", *IEEE Transaction on Parallel and Distributed Systems*, Vol. 12, No. 12, pp. 1236 – 1255, IEEE CS Press.

141. Tate, A. (1998) "Roots of SPAR – Shared Planning and Activity Representation". *The Knowledge Engineering Review*, Vol. 13, No. 1, pp. 121 – 128. Cambridge University Press.

142. The Standish Group International, Inc. (2001) "Extreme Chaos", *CHAOS Report 2001*, The Standish Group International Inc Publication.

143. Truelove, K., and Chasin, A. (2001) "Morpheus Out of the Underworld", Retrieved on 10/08/2006, http://www.openp2p.com/pub/a/p2p/2001/07/ 02/morpheus.html.

144. Turban, E., Aronson, J. E., & Liang, T. (2005) *Decision Support Systems and Intelligent Systems* (7th edition), pp. 223. Prentice Hall Publication.

145. Vahidov, R. (2005) "Intermediating User-DSS Interaction with Autonomous Agents", *IEEE Transaction on SMC Part-A*, Vol. 35, No. 6, pp. 964 – 970. IEEE Press.

146. Vahidov, R. (2006) "Design Researcher's IS Artifact: a Representational Framework", *Proc. of ICDSIST*, pp. 19 – 33. Claremont, U.S.A.

147. Vahidov, R. (2002) "Decision station: A Notion for A Situated DSS", *Proc. of HICSS2002,* Hawaii, pp. 1338 – 1347. IEEE CS Press.

148. Vahidov, R. and Fazlollahi, B. (2004) "Pluralistic multi-agent decision support system: a framework and an empirical set", *Information & Management*, Vol. 41(2004), pp. 883 – 898. Elsevier Press.

149. Venables, W. N., Ripley, B. D. (1999) *Modern Applied Statistics with S-PLUS*, Springer-Verlag Press.

150. Vinoski, S. (1997) "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environment." *IEEE Communications Magazine*, Vol. 14, Issue. 2, pp. 46 – 55. IEEE CS Press.

151. Wagner, G. (2003) "The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior", *Information Systems*, Vol. 28, No. 5, pp. 475 – 504. Elsevier Press.

152. Walsh, T. (1999) "Search in a Small World", *Proc. of IJCAI*, pp. 1172 – 1177. Morgan Kaufmann Publishers.

153. Wang, M., Wang, H., Xu, D., Wan, K.K., and Vogel, D. (2004) "A Web-service Agent-based Decision Support System for Securities Exception Management", *Expert Systems with Applications*, Vol. 27, pp. 439 – 450. Elsevier Press.

154. Wang, X.F., and Chen, G. (2003) "Complex Networks: Small-World, Scale-Free and Beyond", *IEEE Circuits and systems Magazine*, Vol. 3, Issue. 1, pp. 6 – 20. IEEE Press.

155. Washe, H. et al., (2001) "Ontology-based Integration of Information- A Survey of Existing Approaches", *Proc. of IJCAI*, pp. 108 – 117. Morgan Kaufmann Publisher.

156. Watts, D.J. (1999) "*Small Worlds: The Dynamics of Networks between Order and Randomness*", Princeton University Press.

157. Watts, D.J., and Strogatz, S.H. (1998) "Collective Dynamics of 'Small World' Networks", *Nature*, Vol. 393, pp. 440 – 442. Macmillan Magazines Publication.

158. Weiss, G. (1999) "Prologue", in Weiss, G. (eds.): *Multiagent Systems – A modern approach to distributed artificial intelligence*, pp. 1 – 9. The MIT Press.

159. Weiss, M. (2003) "Pattern-driven Design of Agent Systems: Approach and Case Study", *Advanced Information Systems Engineering*, pp. 711 – 723. Springer-Verlag Press.

160. Whinston, A. (1997) "Intelligent Agents as a Basis for Decision Support Systems", *Decision Support Systems*, Vol. 20, No.1, pp. 1 – 2. Elsevier Press.

161. Wickler, G.J. (1999) "Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation", *PhD thesis*, University of Edinburgh.

162. Wooldridge, M. (1999) "Intelligent Agent", in: Weiss, G. (Eds.), *Multiagent systems – A modern approach to distributed artificial intelligence*, pp. 28 – 73, The MIT Press.

163. Wooldridge, M. and Jennings, N.R. (1995) "Agent Theories, Architectures, and Language: A Survey" in: Wooldridge, M. and Jennings, N. R. (Eds.), *Intelligent Agents*, pp. 1 – 22, Springer-Verlag Press.

164. Wooldridge, M., Jennings, N.R., and Kinny, D. (2000) "The Gaia methodology for agent-oriented analysis and design", *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, No. 3, pp. 285 – 312. Springer-Verlag Press.

165. Xu, D., Yin, J., Deng, Y., and Ding J. (2003) "A Formal Architectural Model for Logical Agent Mobility." *IEEE Transactions On Software Engineering*, Vol. 29, No.1, pp. 31 – 45. IEEE CS Press.

166. Xu, Z., and Hu, Y. (2003) "SBARC: A Supernode Based Routing and Caching Peer-to-Peer File Sharing System", *Proc. of ISCC03*, pp. 1053 – 1058. IEEE CS Press.

167. Zhang, H.L., Leung, C.H.C, and Raikundalia, G.K. (2005) "AOCD: A Multi-agent Based Open Architecture for Decision Support Systems", *Proc. of CIMCA*, Vol. 2, pp. 295 – 300. IEEE CS Press.

168. Zhang, H.L., Leung, C.H.C., and Raikundalia, G.K. (2006) "Matrix-Agent Framework: A Virtual Platform for Multi-agents", *Journal of System Sciences and Systems Engineering*. Vol. 15, No. 4, pp. 436 – 456. Springer-Verlag Press.

169. Zhang, H.L., Leung, C.H.C. and Raikundalia, G.K. (2006) "Towards the *Matrix* Concept: A Virtual Platform for Intelligent Agent Application in Decision Support Systems", *Proc. of The 7th International Symposium on Knowledge and Systems Science*, Lecture Notes in Decision Sciences 8, pp. 115 – 122. Global-Link Publisher.

170. Zhang, H.L., Leung, C.H.C. and Raikundalia, G.K. (2006) "Performance Analysis of Network Topology in Agent-based Open Connectivity Architecture for DSS", *Proc. of AINA 2006*, Vol. 2, pp. 257 – 261. IEEE CS Press.

171. Zhang, H.L., Leung, C.H.C., and Raikundalia, G.K. (2006) "Classification of Intelligent Agent Network Topologies and A New Topological Description Language for Agent Networks", *Proc. of 4th IFIP Conf. on Intelligent Information Processing III,* Vol. 228, pp. 21 – 31. Springer-Verlag Press.

172. Zhang, H.L., Leung, C.H.C. and Raikundalia, G.K. (2007) "Performance Evaluation of Agent Network Topologies Based on the AOCD Architecture", *Proc. of AINAW 2007*, pp. 605 – 610. IEEE CS Press.

173. Zhang, H.L., Leung, C.H.C. and Raikundalia, G.K. (2008) "Topological Analysis of Agent Networks and Experimental Results based on the AOCD Architecture", *Journal of Computer and System Sciences*, Vol. 74, No. 2, pp. 255 – 278. Elsevier Publication.

174. Zhang, Z. and Zhang, C. (2004) *Agent-Based Hybrid Intelligent Systems – An Agent-Based Framework for Complex Problem Solving*, pp. 7 – 68. Springer-Verlag Press.