Mobile Agent Computing in Electronic Business:
Potentials, Designs and Challenges

Xinfeng Yang

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy
School of Information Technology
Griffith University, Australia

January 2005
Statement of Originality

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

___________________
Xinfeng Yang
Abstract

The mobile agent computing and mobile agent-mediated e-business fields are exciting research areas which are full of potential and challenges. This research is prompted by two natural observations. On the one side, there are many computing paradigms available to support multiple hosts to communicate over a network such as client-server remote procedure calling, remote evaluation, code-on-demand, and object migration. So, why does the mobile agent paradigm still attract continued attention from the distributed computing community? Is there any peculiar application for the mobile agent paradigm that makes it outperforming all other existing technologies? On the other side, e-business has become more prominent in recent years. With the rapid development of information technology and the widespread popularity of the Internet, various techniques, such as Applets, Servlets, Common Gateway Interface, and Java Server Page, have been proven successful to help online businesses enhance their network functions. Therefore, what is the justification of employing mobile agents for e-business applications? And why is there no commercialised system so far to support mobile agent-mediated e-business in reality?

This study seeks to give a deeper analysis and discussion on these questions and work out meaningful solutions. Two types of people may have special interests in this
research. One is the distributing system designer: the analysis of advantages and disadvantages of mobile agent computing may assist them to choose the best available paradigm to achieve communication efficiency for network computing. The other is the e-business application developer: the proposed e-business models, design patterns and protection structures may convince them to employ mobile agents in the future for their business service improvement.

In this thesis, mobile agent computing is examined in two perspectives: the evolution of mobile agent paradigm, and the development of mobile agent platform. One major reason for this arrangement is that mobile agent technology is believed to have a natural evolution from a set of earlier technologies, and at the same time adds some new characteristics specific to the mobile agent world (e.g., disconnected operation and autonomous migration). Still most of these advantages have to be facilitated by the underlying execution platform.

The advantages and novelty of using mobile agents for e-business construction are analysed in three parts. Firstly, a collection of mobile agent-mediated e-business models is proposed. These models seek to help online businesses make use of agent-specific features to automate their transactions and improve their quality of service. Secondly, the applicability of employing competitive analysis for mobile agent-oriented decision-making is explored, with a view to improving a mobile agent's adaptability in an online environment with the guide of an optimised competitive ratio. Finally, a group of agent-based patterns for e-business system design is proposed, aimed at encapsulating empirical experiences and domain knowledge to avoid similar problem-solving strategies being constantly reinvented.

While mobile agent technology brings great potential to the e-business field, a series of challenging problems also arises. The most serious of these is with regards to
security. Without a proper security structure or protection scheme, one cannot expect mobile agents to be quickly adopted for commercial uses. In this thesis, mobile agent protection in an Internet-based e-business system is investigated. A number of agent-protection structures are suggested to enhance an agent's self-protecting capability in a competitive computing environment, as well as increase protection strength and lower development cost. In the conclusion, several future directions are suggested.
Acknowledgments

Firstly, thanks to my supervisor, Dr Anne Nguyen, who has provided me constant support, constructive suggestions and insightful guidance. I am lucky for this opportunity to work with her. Without her help, the current work would not have been accomplished.

I am also grateful to the School of Information Technology, Griffith University for offering me an enjoyable research environment and sponsored my attendance of various conferences.

Thanks to the teachers who helped me during my PhD study: Dr Jun Jo, Dr Jane You, Dr Vallipuram Muthu, Professor Abdul Sattar, Professor Chengzheng Sun, and Professor James Liu. I benefitted a great deal from their valuable comments and suggestions.

I can not forget all the people with whom I shared my PhD experience and research work: Dr Weimin Ma, Dr Yanchao Xing, Dr Michael Cowling, Dr Raymond Lee and many others. Working with them was truly a fond memory.

A particular thanks goes to my parents, for their love and encouragement.
## Contents

Abstract.................................................................................................................................................. iii

List of Tables ......................................................................................................................................... xi

List of Figures...................................................................................................................................... xii

1 Introduction......................................................................................................................................... 1
   1.1 Research questions ......................................................................................................................... 2
   1.2 Domain and scope ............................................................................................................................. 3
   1.3 Research approach and principal contributions ............................................................................. 3
   1.4 Outline of the thesis ......................................................................................................................... 6

2 A General Review of the Multi-Agent and Mobile-Agent Technologies........... 8
   2.1 Chapter introduction ......................................................................................................................... 8
   2.2 Variety in the software agent world ................................................................................................. 9
      2.2.1 Defining a software agent from different perspectives ......................................................... 10
      2.2.2 Facilitating co-ordination within multi-agent systems ....................................................... 13
      2.2.3 Constructing an intelligent agent with learning ability ...................................................... 16
      2.2.4 Equipping an agent with free mobility .................................................................................. 20
   2.3 The evolution of the mobile agent computing paradigm ......................................................... 23
      2.3.1 Peer-to-peer messaging and client-server remote procedure calling .................................. 23
      2.3.2 Process migration: weak and strong mobility ........................................................................ 25
      2.3.3 Remote evaluation and code-on-demand ............................................................................... 27
      2.3.4 Object migration and mobile objects ...................................................................................... 29
      2.3.5 Mobile agent computing: benefits and challenges ............................................................... 31
2.4 The development of mobile agent platforms ..................................................35
  2.4.1 Making use of mobile code languages .....................................................35
  2.4.2 Setting up common execution environments ...........................................40
  2.4.3 Providing supports for agent migration ..................................................46
  2.4.4 Enabling agent-to-agent communication .............................................51
  2.4.5 Providing necessary security mechanisms ...........................................53
2.5 Chapter summary ...........................................................................................57

3 E-Business Models with Mobile Agent Technology ........................................60
  3.1 Chapter introduction ....................................................................................60
  3.2 Agent-oriented e-markets ..........................................................................61
    3.2.1 General business activities within a virtual market ..............................61
    3.2.2 Using mobile agents for e-market development .................................65
    3.2.3 A generic architecture for mobile agent-based e-market .....................69
  3.3 Internet-supported e-business .....................................................................76
    3.3.1 Basic e-business models ....................................................................76
    3.3.2 Transplanted and native e-business models .......................................82
    3.3.3 Business models employing agent technology ..................................88
    3.3.4 Agent technology for the development of information economy ..........91
  3.4 Mobile agent-mediated e-business models .............................................93
    3.4.1 Single-buyer/single-seller model .......................................................94
    3.4.2 Single-buyer/multiple-sellers model ..................................................96
    3.4.3 Single-seller/multiple-buyers model ..................................................98
    3.4.4 Multiple-buyer/multiple-sellers model ..............................................99
    3.4.5 Mobile broker model .......................................................................101
  3.5 Chapter summary .....................................................................................103

4 Mobile Agent Design Patterns for Business Applications ..........................105
  4.1 Chapter introduction .................................................................................105
  4.2 Design patterns for software engineering ..............................................106
    4.2.1 Fundamental components for software pattern design ......................107
    4.2.2 Common problems for most agent applications developments ........111
    4.2.3 Applying object-oriented patterns to the agent world ......................113
4.3  Agent-oriented and e-business-specific software patterns............................115
  4.3.1 A layered pattern for a functional agent.........................................115
  4.3.2 An architectural pattern for an agent system...............................119
  4.3.3 Some specific patterns for Internet shopping..............................121
  4.3.4 A family of patterns for agent-mediated e-business ......................123
4.4  Design patterns for mobile agent-mediated e-business...............................125
  4.4.1 A refined pattern template and a set of general forces....................125
  4.4.2 Market-organiser pattern.................................................................129
  4.4.3 Agent-side comparison-shopping pattern.......................................134
  4.4.4 Server-side comparison-shopping pattern.......................................139
  4.4.5 Itinerary-balancing pattern.............................................................144
4.5  Chapter summary.....................................................................................148

5  Competitive Analysis of Online Agent-Mediated E-Business......................150
  5.1  Chapter introduction.............................................................................150
  5.2  Computing techniques for solving problems under uncertainty............151
    5.2.1 Soft computing for agent-mediated e-business..............................151
    5.2.2 Hybridisation of various soft computing techniques......................157
    5.2.3 Online problems and competitive algorithm...................................160
  5.3  Competitive analysis within business and mobile agent domains.............163
    5.3.1 Selecting different adversary models...........................................164
    5.3.2 Adopting general proof techniques..............................................166
    5.3.3 Addressing online problems within the business field....................168
    5.3.4 Employing competitive algorithm for mobile agent applications......172
  5.4  Online decision-making for mobile agent-mediated e-business..............177
    5.4.1 Market-oriented mobile agent planning.......................................177
    5.4.2 The travelling agent for travel agency problem............................178
    5.4.3 Competitive analysis and proposed online solution.......................181
    5.4.4 Competitive ratio and its lower bound.........................................186
    5.4.5 Experimental results and algorithm comparison............................190
    5.4.6 Applicability and limitations for the practice................................196
  5.5  Chapter summary.....................................................................................198
List of Tables

Table 2.1: Major benefits and potential applications of mobile agent technology ......32
Table 2.2: Several major mobile code languages.........................................................36
Table 2.3: Some mobile agent system examples ..........................................................41
Table 2.4: Benefits and drawbacks of Java for mobile agents’ development............44
Table 3.1: Basic e-business models and corresponding examples...............................77
Table 3.2: Transplanted e-business models with their sub-categories and examples..83
Table 3.3: Native e-business models with their sub-categories and examples ..........85
Table 3.4: E-business models and examples employing agent technology .............89
Table 3.5: Characteristics of value networks and dynamic markets .........................91
Table 4.1: Fundamental components for software pattern design .............................110
Table 4.2: Concrete patterns within a layered-like functional agent .........................117
Table 4.3: Some selected patterns for Internet shopping ...........................................121
Table 4.4: A template for mobile agent-mediated e-business patterns .....................125
Table 5.1: Property assessment of three soft computing techniques .........................157
Table 5.2: Incorporating RSE algorithm for proposed after-sale service system ......176
Table 5.3: Simulation results of the follower algorithm facing ADOF adversary ....193
Table 5.4: Simulation results of the random algorithm facing ADOF adversary .......193
Table 5.5: Simulation results of four algorithms within a practical environment ....194
Table 5.6: Simulation results of three algorithms when $\phi$ is fixed .................195
Table 6.1: A brief summary of potential attacks within a mobile agent system .......203
Table 6.2: A brief summary of potential strategies for agent host protection.........210
Table 6.3: Merits and drawbacks of partial result protection......................................233
Table 6.4: Merits and drawbacks of agent itinerary protection ...............................234
Table 6.5: Notation of the elements used in Equations (6.1) and (6.2) ......................237
Table 6.6: Notation of the elements used in the MATE protocol ..............................246
Table 6.7: Notation of the elements used in Equation (6.4)....................................247
Table 6.8: Notation of the elements used in MATE with KDC protocol .................248
Table 6.9: Notation of the elements used in VTTP protocol ....................................252
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>A graphical presentation of a software agent</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>A structural view of all models of a mobile agent</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>The transfer of data in peer-to-peer messaging</td>
<td>23</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>The message flow in remote procedure calling</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>A simplified working mechanism for process migration</td>
<td>25</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>The code migration in RE</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>The code migration in CoD</td>
<td>28</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>A simplified working mechanism for object migration</td>
<td>30</td>
</tr>
<tr>
<td>Figure 2.9</td>
<td>A sketch of the mobile agent computing</td>
<td>31</td>
</tr>
<tr>
<td>Figure 2.10</td>
<td>The execution environment in Telescript</td>
<td>42</td>
</tr>
<tr>
<td>Figure 2.11</td>
<td>The execution environment in Agent Tcl</td>
<td>43</td>
</tr>
<tr>
<td>Figure 2.12</td>
<td>The execution environment in Aglets</td>
<td>45</td>
</tr>
<tr>
<td>Figure 2.13</td>
<td>A process-based life cycle model of a mobile agent</td>
<td>46</td>
</tr>
<tr>
<td>Figure 2.14</td>
<td>An event-based life cycle model of Aglets</td>
<td>48</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>A set of generalised e-business processes</td>
<td>62</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>A mobile agent-oriented e-shopping centre architecture</td>
<td>66</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>A mobile agent-based e-market framework: TabiCan</td>
<td>67</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>A mobile agent-supported searching structure</td>
<td>68</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>A generic architecture for mobile agent-mediated e-market</td>
<td>69</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Classification of eleven basic e-business models</td>
<td>81</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Business models with value networks and dynamic markets</td>
<td>92</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Single-buyer/single-seller model</td>
<td>95</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>Single-buyer/multiple-sellers model</td>
<td>97</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>Single-seller/multiple-buyers model</td>
<td>99</td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>Multiple-buyers/multiple-sellers model</td>
<td>100</td>
</tr>
<tr>
<td>Figure 3.12</td>
<td>Mobile broker model</td>
<td>102</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Applying OO design patterns within an agent-based framework</td>
<td>113</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>A layered pattern for a functional agent</td>
<td>116</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Employing mobile agent technology for e-business construction has attracted more and more attention in recent years. On the one hand, e-business applications may require new technical models to realise their potential and be cost effective. On the other hand, mobile agents with flexible adaptability can assist their owners to participate in business activities automatically and deal with various e-business situations intelligently, thereby successfully help those businesses wishing to migrate to the Internet enhance their network functions. A future electronic market mediated by mobile agents can be represented on one side by selling agents displaying goods and providing services via extended distribution channels, the other side is comprised of buying agents who scan the markets selectively for information-gathering and negotiation.

However, in much of the current literature, the mobile agents computing and its potential applications within the e-business domain are always presented in a piecemeal manner. A specific research that takes the mobile agent-mediated e-business as an organic whole and examines associated benefits and challenges in a
systematic way is still lacking. This particular problem will be critically examined in this thesis. A comprehensive analysis of the merits and drawbacks of mobile agent computing compared with a set of earlier networking technologies will be conducted. Both the theoretical and realistic feasibility of employing mobile agents for e-business development will be discussed, with some proposed arrangements that should contribute to their commercial deployment.

In the remainder of this section, the research questions of this thesis will be raised, and the research domain and scope will be identified. Approaches and principal contributions derived from this thesis will also be given.

1.1 Research questions

The research questions arise from the problem outlined in the introduction, more specifically, these can be stated as follows:

- How does the mobile agents technique evolve from a set of traditional technologies and develop into a fully-functional computing paradigm, and what potential advantages does a mobile agent offer that may make a significant contribution for e-business development?

- How can online businesses benefit from incorporating mobile agent technology in their business modelling for value-network and dynamic-market implementations, and how can software developers make use of various design patterns and problem-solving strategies to construct those dynamic applications with industrial strength?

- What are the main reasons for industry's reluctance to adopt mobile agents for e-business construction, and how should those challenging tasks be addressed theoretically and the associated risks minimised experimentally, in order to
promote the widespread adoption of mobile agent-mediated e-business applications for commercial uses?

1.2 Domain and scope

The research domain of this thesis is mobile agent technology -- which is considered to be a special branch of distributed artificial intelligence. Specifically, its application for e-business is technically investigated. It is envisioned that an open electronic market in the future can be based on multi-agent and mobile-agent technology. Mobile agents representing their owners migrate automatically from one market to another for information retrieval and service comparison, while online businesses employ various agent-operators in their system design to improve their network functions and enrich their profit generation methods.

Within the above domain, the focuses in this thesis are on mobile agent-mediated e-business models, design patterns, decision-making algorithms and security protection schemes. The first three aspects seek to investigate how mobile agent-oriented advantages can be implemented to benefit all market participants. The last aspect aims for promoting wide acceptance of mobile agent technology for commercialised applications through minimising possible deployment obstacles.

1.3 Research approach and principal contributions

Before undertaking a comprehensive study on the mobile agent computing and mobile agent-mediated e-business, the following hypotheses are postulated:

1. The emerging mobile agent computing paradigm is ideal for network computing and e-business applications;
2. Online businesses need mobile agent technology to enrich their Internet functions and realise their potential;

3. Wide acceptance of mobile agent-mediated e-business can be achieved if security obstacles are overcome technically in the near future.

Towards proving these hypotheses, the research plan and principal contributions can be summarised as follows:

- **Research Plan and Aim**: A general review on the mobile agent computing evolution and a cross-language comparison on the mobile agent platform establishment are required. Through analysing several existing technologies for distributed computing, a number of major benefits brought by mobile agent computing and a collection of potential applications that may employ mobile agents can be identified. The study of the development of mobile agent platforms seeks to give a comparative view on various structural and functional components (such as mobile code language, agent-to-agent communication mechanism and host-to-host transportation facility) that may help mobile agents achieve their technical advantages in the network computing domain.

- **Research Plan and Aim**: It is necessary to examine which roles mobile agents can play within a generic e-market, and how business models can benefit from agent technology for value-added generation. The aim of this examination is to clarify the potential and impediments of using mobile agents for e-market development, and analyse mobile agents' contribution to online business modelling from a value-network implementation viewpoint.
**Principal Contribution:** A collection of mobile agent-based e-business models are proposed, which should enhance an online business' buying and selling process by making use of the mobile agents' self-executable and self-contained properties.

- **Research Plan and Aim:** It is required to investigate various soft-computing techniques that can be used for mobile agents to deal with all types of uncertainties intelligently, with the aim of enhancing both flexibility and adaptability in an ever-changing e-business environment.

**Principal Contribution:** The applicability of employing competitive algorithms for mobile agent-mediated task scheduling and decision-making is rigorously examined. A dynamic decision-making scheme guided by an optimised competitive ratio is developed, combining push-based technology with the competitive algorithm to benefit both the agent-owner and its potential clients in an online fashion.

- **Research Plan and Aim:** It is desirable to study in depth how different software developers can document their knowledge and experience for problem-solving and system-structuring in a standard style, in order to avoid duplicated efforts being made to address recurrent problems within the same area.

**Principal Contribution:** A set of general forces specific to the mobile agent-supported e-business pattern design is identified. A group of patterns that are related to virtual market management, agent-based shopping and agent-to-agent collaboration is proposed. Mobile agent application designers are expected to reuse those explored patterns to simplify the associated software development process.

- **Research Plan and Aim:** The last part of this thesis devoted to the analysis the major impediments preventing mobile agents from being employed for e-business
applications, and how the principal impediments can be addressed in a balanced way, giving consideration for both agents' security requirements and systems' computing performance.

Principal Contribution: The difficulty of protecting mobile agents against malicious hosts is theoretically clarified. A mobile agent's self-carrying and self-verification capability is identified. Several dynamic structures making use of these properties for agent protection are proposed, aiming at enhanced computational performance with increased protection items.

1.4 Outline of the thesis

The remainder of the thesis consists of six chapters. These are organised as follows:

Chapter 2 gives a general review on the multi-agent and mobile agent technology, focusing on the variety in the software agent world, the evolution of the mobile agent computing paradigm, and the development of mobile agent platforms.

Chapter 3 investigates how to incorporate mobile agent technology into the e-business models, and illustrates the active roles that mobile agents can take part within a virtual market to contribute to the evolution of the information economy.

Chapter 4 examines the common problems for most agent-based applications development, exploring how software patterns can be applied to the agent world and employed for mobile agent-mediated e-business system designs.
Chapter 5 analyses how decision-making strategies can help a mobile agent address various uncertainties when travelling around the Internet and working for its owner. Competitive analysis is quantitatively introduced for this purpose.

Chapter 6 deals with the security issues involved in mobile agent-oriented applications. Potential threats, taxonomy of attacks and security requirements for both agent and agent-host protection are identified. Special emphasis and solutions are placed on risks associated with a mobile agent being attacked by a malicious host.

Chapter 7 concludes this research, and discusses several future directions that can be followed from this work.
Chapter 2

A General Review of the Multi-Agent and Mobile-Agent Technologies

2.1 Chapter introduction

Agent technology has attracted a lot of interest in recent years, but the term ‘agent’ seems to be overused, and a standard definition has not been agreed upon. Many kinds of agents can be found within literature, such as autonomous agents, intelligent agents, collaborative agents, learning agents, mobile agents, information agents, believable agents and many more. Then, what is a software agent? What is the difference between an agent and a program? How many types of software agents are there? Such debate has lasted for some years and the issues seem to be how to define a software agent in a simple and extensible way, and how to classify agent applications in an effective and meaningful manner. Among various software agents, the mobile agent has drawn special attention from both industry and academia because of its significant feature of travelling around the Internet and performing a given task.
automatically. Then, which technology enables a mobile agent? Does the mobile agent paradigm outperform traditional computing mechanisms? This chapter will give a brief review on key aspects associated with software agent and mobile agent technologies (Section 2.2). A historical view of the evolution of mobile agent computing will be given in Section 2.3 and a comparative view of the development of the mobile agent platform will be presented in Section 2.4.

2.2 Variety in the software agent world

'Agent' itself is not a new concept. An estate agent or education agent has brought a lot of convenience to our lives. The Oxford English Dictionary (Murray et al, 1991: 248), defines "agent" as "one who does the actual work of anything, as distinguished from the instigator or employer, hence, one who acts for another, a deputy, steward, factor, substitute, representative or emissary". It is not difficult to see that acting as a representative for others is the basic function of an agent. An estate agent represents property owners to do business with its clients, while an education agent represents academic institutions by providing services to its customers. Software agents operating in a computing environment share the same function: assisting software users and acting on their behalf. Technically, agent technology belongs to a special branch of a broad scientific area -- distributed artificial intelligence. Reasoning theory, intentional systems and knowledge-based systems have all exerted great influences to the evolution of software agents (Nwana et al, 1996; Green et al, 1997). As early as the 1970s, Hewitt (1977) proposed an actor model for concurrent computation in distributed systems, an actor being described as an independent and active computational agent that exhibits behaviours. Both an actor's state and reaction rules are encapsulated, while communicating with other actors and interacting with its
environment are facilitated. As a self-contained entity, an actor exhibits some human abilities such as performing its actions autonomously and responding proactively to environmental changes.

2.2.1 Defining a software agent from different perspectives

Software agent technology developed smoothly through the 1980s. As more and more computing disciplines are integrated, agent definition seems to be becoming diversified. The following sections attempts to list several 'software agent' definitions given by the associated research groups in this area, and provides a simple discussion.

- From Smith et al (1994) at Apple

"An agent is a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from sub-routines; agents have their own ideas about how to accomplish tasks, and their own agendas. 'Special Purpose' distinguishes them from entire multi-function applications; agents are typically much smaller".

In this definition, 'persistent' and 'special purpose' are described as two distinct properties among software agents and program sub-routines and multi-function applications. Characteristics of an agent can be summarised as autonomous, reactive and goal-oriented.

- From IBM's Intelligent Agent Strategy Group (Franklin & Graesser, 1996)

"Intelligent agents are software entities that carry out a set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires".
IBM's definition is associated to intelligent agents; it describes an agent as a software entity. The characteristics that an intelligent agent possesses include delegacy, independence and autonomy. Reasoning ability is viewed as an important requirement for an intelligent agent to perform tasks for its owner; however, it does not cite 'persistence', which was regarded as a distinctive feature by Smith et al (1994).

- From Hayes-Roth (1995) at Stanford's Knowledge Systems Lab

"Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions".

Hayes-Roth's definition focuses on an agent's learning and reasoning ability as well as its reactivity, pro-activity and autonomy. Dynamic conditions and continuous performance are included as important properties of agency. Hayes-Roth (1995) believed this comprehensive description summarises the most essential features of a software agent that could be spelled out in many other examples of literature.

- From Lange (1998) at General Magic, Inc.

"An agent is a software object that is situated within an execution environment. It must possess the following mandatory properties:

- Autonomous: performing tasks on behalf of a user and self-directed in execution;
- Reactive: responsive to changes in the environment;
- Goal-driven: proactive, acting in advance to deal with an expected situation;
- Temporally continuous: continuously operating."
It may possess some of the following orthogonal properties:

- Communicative: communicates and co-operates with other agents;
- Learning: adapts its behaviour according to its previous experience;
- Mobile: transports from one host machine to the other;
- Believable: shows social and emotional believability”.

Lange was a chief architect for IBM's ASDK (Aglets Software Development Kit). A significant feature of his definition is dividing an agent's properties into primary and secondary parts. A program possessing those essential properties is considered a software agent, while those non-essential properties make a software agent versatile. This definition can be graphically presented in Figure 2.1:

![Software Computing Environment](image)

**Figure 2.1: A graphical presentation of a software agent**

Compared with other definitions listed above, at least two advantages can be noticed in this definition. Firstly, the simple, primary properties can be used to distinguish a software agent from other ordinary programs, while secondary properties provide a meaningful way to classify different types of agents within the same family. Secondly, those secondary properties that an agent may or may not posses can be extended when agent technology are well understood in the future, and various combinations are also possible.
Therefore, it seems that this definition has broadly answered the questions posed in the chapter introduction, and at the same time has allowed plenty of scope for further research. In the remainder of this thesis, this definition is chosen as a standard expression of software agents. A more detailed review of the three secondary properties identified in this definition, namely communicability, adaptability, and mobility, will be given in the following subsections.

2.2.2 Facilitating co-ordination within multi-agent systems

The ability of a single agent is limited, so it is desirable for an individual agent to communicate and co-operate with other agents to perform complex tasks that are beyond its own capability. Thus, Multi-Agent System (MAS) is formed. The purpose of MAS research is to enhance a system's speed, reliability and extensibility to solve problems where information sources are distributed, or the computation is too large to be carried out by a single agent (Green et al, 1997). According to Franklin and Graesser's classification (1996), a system without communication among its agent members is a discrete MAS, for example, the Sumpy system presented by Song et al in 1996. A system where communication is essential for all its agent members is called fully-connected, such as the Message-Board-based Information Sharing system (MBIS) suggested by Kearney et al (1994). Two more co-ordination styles can be further classified in a fully-connected system: centralised coordination, and distributed coordination. In the former structure, the co-ordinator is responsible for collecting information from its group members, analysing their separate plans, and detecting potential conflicts. After eliminating all conflicting schedules, the central manager generates a global solution and sends it back to individual members to ensure multiple agents' coherence (Sycara, 1989; Skarmeas & Clark, 1996; Seilonen
et al, 2000). In contrast, the distributed technique allows independent agents to model each other's plans. Local plans can be revised from time to time through inter-agent communication until all conflicts have been eliminated and a global plan formed. Advantages of this arrangement include enhanced real-time response, lowered processing costs, and increased system flexibility (Parunak, 1987; Sandholm, 1993; Sandholm & Lesser, 1995).

From a 'value addedness' viewpoint, the added value of a discrete MAS is simply the capability of its most functional member, whereas in a fully-connected MAS it is expected to be greater than the capability of any of its single members. Huhns and Singh (1994) expressed this conclusion as follows:

\[ V(\cup_{agent_i}) > \text{Max}(V_{agent_i}) \]  

(2.1)

where \( V \) represents 'value addedness' for a set of system attributes such as speed, reliability, and accuracy. \( \cup \) refers to the set of all the agents. Therefore, the left side of this equation is the added value of a fully-connected MAS, while the right side is that of the function of its most capable member. Unfortunately, the above formula does not seem to consider the added value that might come from its society members' collaboration. A direct observation is that whether the added value of a MAS can be equal to or even greater than the cumulative value of all its individual members. In other words, whether the communication and co-operation within the system's participants can help to generate more added values, as expressed in Equation (2.2):

\[ V(\cup_{agent_i}) > \sum_{i} V_{agent_i} \]  

(2.2)

This equation needs further proof to demonstrate its correctness and applicability, as presently it is merely a conjecture of what a fully-connected MAS is expected to perform.
Agent Communication Language (ACL) is an important factor for multi-agent collaboration. Interoperation facilities among miscellaneous agents are even viewed as the core of a new research domain called agent-based software engineering (Nwana, 1996). Potential benefits for having heterogeneous agent technology include enhanced interoperability with legacy system, improved value-added service implementation, and simplified software maintenance. FIPA (Foundation for Intelligent Physical Agents) is an organisation aimed at drawing up software standards for the interoperation of heterogeneous agent systems. Two major language standards have been proposed so far: KQML (Knowledge Query and Manipulation Language), and ARCOL (ARtimis COmmunication Language).

As a standard language to support inter-agent communication, KQML consists of three layers in concept: the content layer, the message layer, and the communication layer (Finin et al., 1994). The content layer contains the actual expression of a message, which can be written in any internal language agreed upon between two communicators. The message layer provides a set of performatives to describe the features of the message (e.g., a content message or a declaration one). The communication layer is the outmost wrapper that specifies the lower-level communication parameters (e.g., the identities of the sender and receiver). All three layers are established on the underlying network transport mechanisms. Therefore, KQML is both a message format and a language-interpreting protocol. However, the lack of precise semantics is an obvious shortcoming of KQML. Without explicit semantics, as Cohen and Levesque (1995) observed, different agent designers cannot be certain whether the interpretation they are giving to a performative has the same meaning. By contrast, ARCOL (developed by Sadek, 1992) has well-defined semantics. It not only defines a set of communicative acts (or message-types, for
example, inform, request, and confirm), but also incorporates a logic of mental
attitudes (such as intentions and beliefs) into the message level. This specification
overcomes one of the limitations of KQML, where the concept of performatives is
rather ambiguous and error-prone. With ARCOL, agents can express their mental
attitudes in a complete and clear style with the help of improved presentation logic.
However, determining whether a receiving agent believes a fact or not creates another
weakness for ARCOL (Wiederhold, 1992). Besides, neither KQML nor ARCOL
language carries semantics specification at the communication level. This remains an
open issue in the field. For more information, please refer to Chaib-draa and Dignum
(2002).

2.2.3 Constructing an intelligent agent with learning ability

Learning is a way for a software agent to gain its intelligence. Short-term learning can
effectively improve an agent's short-term performance (i.e., document retrieving and
differentiating), while long-term learning can be used to help its owner accomplish a
more complex task (i.e., business arrangement or personal assistant). According to
different manners of knowledge-gaining, three kinds of learning agents have been
broadly identified: user-programmable agent, AI-engineering agent, and self-
programming agent (Lewis, 1991).

Rules provided by a user are the most common resources for an agent to learn.
The OVAL (Objects, Views, Agents and Links) system developed by Malone et al
(1995) is a good example, in which agents perform their tasks based on a set of user-
specified criteria. It is easy for non-programmers to use, as they are required only to
specify a group of rules for various triggering events and associated responses. With
the help of a template-based interface, an end-user who is new to software
development can create powerful applications that might otherwise require professional programming skills. In addition, user-programmable agents can be found in a number of co-operative work applications. Examples include the gIBIS system (Conklin & Begeman, 1988) for decision-making support, the Co-ordinator system (Winograd, 1987) for conversation structuring, and the Information-Lens system (Malone et al, 1987) for mail system filtering. Agents equipped with user-specified rules free up their users from many repetitive tasks. However, as Cohen et al (1994) noticed, the rule creation method is a problem that needs to be seriously considered by the system designer. These should be easy for non-technical users to understand, while maintaining their powerfulness and predicability.

To some extent, user-provided rules can be viewed as a simplified knowledge base for an agent. Therefore, the effectiveness of this kind of learning is heavily dependent upon an individual user's experience. In order to overcome this limitation, the second agent-learning method makes use of traditional knowledge-engineering techniques for knowledge base development, where the construction task shifts from an agent-user to an experienced knowledge-engineer. Due to the completeness and authoritativeness of a well-designed knowledge base, this agent-learning approach is more suitable to be employed in many online consultant systems, such as the SINIX consultant system SC-UM (Nessen, 1989), the UNIX consultant system UCEgo (Chin, 1991), and the intelligent help system Aran (Fernandez-Manjon et al, 1988). These not only help users to access different kinds of information based on their queries, but also assist them to expand their knowledge within associated domains. However, at least two problems can be noted. Firstly, although it requires a huge amount of work from the system engineer to encode domain knowledge into the knowledge base, little of the former is reusable for building other applications.
Therefore, this kind of knowledge base is broadly application-specific. Secondly, the learning agent's knowledge base is fixed once and for all. It is not easy to encapsulate individual users' habits and preferences at run time, and is therefore not suitable for providing customised services.

The third learning approach allows an autonomous agent to learn by itself. Various machine-learning techniques, for example, symbolic classification (Cecconi et al, 1995), sub-symbolic classification (Miao et al, 2002) and other related skills are major means for these agents to gain their required knowledge. The symbolic classification tries to build a decision tree out of a set of training examples. The construction of the decision tree is always accomplished by recursively partitioning a data set into subsets until each leaf node in the tree shows uniform class membership. As Caragea et al (2000) noted, agent technology provides an efficient means of obtaining necessary information for the decision tree construction from distributed data sources. In the simplest manner, serial learning and parallel learning can help with the transmission of required information from distributed data sites to a learning agent at a central location. In scenarios where the access to raw data in each site is prohibited and only statistical summaries of the data is available, a multi-agent based decision tree construction approach is more appropriate (Caragea et al, 2000). The sub-symbolic approach is based upon Neural Network (NN) technique. Training patterns are injected into the input layer, while data analysis and classification are accomplished in the middle layer, and a number of categories will be sent out from the output layer. Connection strengths (or weights) between neurons are determined automatically during the training phrase. Agent technology is suitable for NN construction because each agent can be modelled as a self-contained neuron, which is expected to get its knowledge in either a supervised or an unsupervised fashion.
Examples of employing NN for agent learning can be found in the map-building system presented by Nehmzow and Smithers (1990), the meeting scheduling system proposed by Kozierok and Maes (1993), and the product manufacturing system suggested by Jiang and Mair (2002).

Co-operative agents always outperform independent agents. This conclusion remains valid for a number of agents during their learning process. Tan (1993) conducted an interesting experiment which considered several hunter agents capturing a prey agent. In this seek-and-capture area, three co-operation systems among hunter agents are examined. In the first case, a scout agent is added to send sensory information to the hunter agent. Experimental results show the average number of steps for the hunter agent to capture a prey is fewer than that for an independent hunter. This will be further improved when the scout's visual-field depth is increased. If the hunter agents are allowed to share learning policies or episodes (a sequence of sensation-action pairs), the learning process will speed up, because each agent can benefit from other agents' already-owned knowledge. While higher communication cost is a penalty, this falls within an acceptable upper boundary. Facing a joint task where prey can only be surrounded and captured by two hunters at the same time, co-operation between hunter agents is not only effective but also desirable, although initial training might take longer. Tan's experiment demonstrates that an agent's learning ability and its co-operative ability are not isolated. Learning agents always outperform non-learning agents; cooperation among learning agents can always improve their performance further. This conclusion can be expressed in the following equation:

\[ V(\text{agent}_{P_L+P_C}) > V(\text{agent}_{P_L}) \]  \hspace{1cm} (2.3)
where $V$ represents 'value addedness' for any system attribute with the same meaning as that in Equation (2.1). $P_L$ represents an agent's learning ability, and $P_C$ represents its collaboration ability. In fact, many kinds of hybrid forms of an agent's secondary properties are also meaningful. Therefore, it is quite reasonable to consider more agent-properties in an extended formula. If an agent's four secondary properties, namely Communicative, Learning, Mobile, Believable are denoted as $P_C$, $P_L$, $P_M$ and $P_B$ respectively, and forming a set $P = \{P_C, P_L, P_M, P_B\}$, for $\forall P_i \in P \ (1 \leq i \leq 4)$, it has:

$$V(\text{agent} \sum_{P_i}) > V(\text{agent}_{P_i})$$

(2.4)

This means an agent incorporating more properties can always generate more added values to a system than an agent only equipped with a single property. This may happen in many ways, at least, learning ability has been incorporated into many agent-based applications to help various agents proactively respond to environmental changes and arrive at intelligent decisions automatically.

### 2.2.4 Equipping an agent with free mobility

Equipping a software agent with mobility is not a trivial breakthrough in technique. In order to get a clearer view of what a mobile agent is required to be or expected to perform, a variety of definitions will be examined in this subsection. Some of these have come from agent research groups, others from software companies.

- From White (1994b) at General Magic Inc.

"A mobile agent is a program that can migrate from machine to machine in a heterogeneous network. The program chooses when and where to migrate. It can
suspend its execution at an arbitrary point, transport itself to another machine and resume execution."

White has become a prominent name in the agent community since Telescript, a very early mobile agent system, was released by General Magic Inc. in 1994. A mobile agent acts like a secretary, a messenger, or a personal manager installed on a personal computer, and can roam freely around the Internet on behalf of its owner. The major motivation behind the Telescript agents, as Cockayne and Zyda stated in 1997, is trying to treat the Internet as a public computing platform, and transferring business decision-making power from a service provider to an individual user.

• From Lange (1998) at IBM's Tokyo Research Laboratory

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

Lange designed another well-performing mobile agent system (Aglets). Under his description, a stationary agent is "executing only on the system where it begins execution and typically using a communication mechanism such as remote procedure calling to interact with an agent on a different system". Therefore, mobility is viewed as an orthogonal property of a more generalised software agent. Making use of resources from another host on the network is the major motivating factor for a mobile agent's migration.
From Agent Research Group at the University of Dublin (Green et al, 1997)

"A mobile agent is a software entity which exists in a software environment. It must contain all of the following models: an agent model, a life-cycle model, a computational model, a security model, a communication model and a navigation model. ...A mobile agent environment is a software system which is distributed over a network of heterogeneous computers. Its primary task is to provide an environment in which mobile agents can execute and support access to other mobile agent systems and non-agent-based software environment."

This definition actually includes both a mobile agent and its working environment description. Based on existing literature, Green et al (1997) believed their definition sufficiently characterised the essence of a mobile agent system. A structural view on a mobile agent based on this description can be depicted in Figure 2.2:

![Figure 2.2: A structural view of all models of a mobile agent (Green et al, 1997)](image)

The computational model at the centre defines how a mobile agent performs its computational task in its working environment. Both the life cycle and the security models exist close to the core. The life cycle model helps an agent effectively arrange its problem-solving plan within its lifespan, while the security model deals with mobile agent protection. The outermost layer defines a mobile agent's other capabilities, most importantly, communication and navigation.
2.3 The evolution of the mobile agent computing paradigm

From the above figure, it can be noted that navigation and communication are two closely related properties of an agent. Navigation can be viewed as a specific form of communication -- jumping to another platform and communicating locally with that host. Then, what is the conventional way for two hosts to communicate over a network? Does the mobile agent paradigm outperform traditional approaches? This section will further examine the evolution of mobile agent computing.

2.3.1 Peer-to-peer messaging and client-server remote procedure calling

Traditional network communication might happen in two principal manners: peer-to-peer messaging (Cypser, 1991), and client-server Remote Procedure Calling (RPC) (Hamilton, 1984). In both methods, specific data needs to be transferred through the network and processed at a remote CPU.

Peer-to-peer messaging involves the transfer of data from a sender's memory to a receiver's memory, which is naturally asynchronous. The sender might continue its execution after handing off the message, as shown in Figure 2.3:

```
  Sender
    Memory
    Data

  Receiver
    Memory
    Data
```

Figure 2.3: The transfer of data in peer-to-peer messaging
(Adapted from Cypser, 1991)

The sending process is required to specify the transferred data's location, size, type and its destination, and the receiving process has a corresponding matching
mechanism. Transaction states must be maintained and restored when the sending process receives a response message from the receiver. This communication technique is widely employed in many electronic mail systems with the aid of other assistant services such as MQS (Message Queuing Series), SMTP (Simple Mail Transport Protocol), and VIM (Vendor Independent Messaging). For general scientific computing, explicit message-passing provides an efficient way to attain optimum parallel performance (Tanenbaum & Steen, 2002).

Client-server-based RPC is another well-known communication technique within distributed computing. If a client wants to use the resources provided by a server, it sends a request to that server. Parameters will be transferred across the network for execution, and results will be sent back following computation. As a type of synchronous communication, the calling environment has to be suspended when a remote procedure is triggered, as depicted in Figure 2.4:

![Figure 2.4: The message flow in remote procedure calling](Adapted from Hamilton, 1984)

Technically, five pieces of program are involved in a RPC message flow: a client application, a client-stub, a RPC communications package (or RPC Runtime), a server-stub, and a server application. The client stub is responsible for placing its arguments and a specification of the target procedure into one or more packets. The server stub is responsible for unpacking these packets and making a normal local call
to invoke the appropriate procedure in the server application. The RPC Runtime in both machines is responsible for packet routing, acknowledgments and retransmissions. RPC has been widely adopted as a basic service in many distributed computing environments, for example, SUN RPC, ONC (Open Networking Consortium) RPC, and DCE RPC (Tanenbaum & Steen, 2002). Unfortunately, it has a typical latency of a few milliseconds, while a local procedure call can be performed in a few microseconds (Birrel & Nelson, 1984).

2.3.2 Process migration: weak and strong mobility

The parameter-passing mechanism in conventional RPC is insufficient for distributed system design, as some situations require a code segment to be transferred over the network and executed in another machine. Code migration (in the form of process migration) is designed to meet such requirements. Improved system performance is a major advantage, as processes can move from heavy-loaded processors to light-loaded ones. Other advantages include simplified system administration, extended data availability, and enhanced fault resilience (Smith, 1988). A simplified working mechanism for process migration can be illustrated in Figure 2.5:

![Figure 2.5: A simplified working mechanism for process migration](Adapted from Dejan et al, 2000)

According to Fuggetta et al’s investigation (1998), a process in a computational environment contains three segments: a code segment including a set of executable
code to perform a specific task, a state segment including a data space to record a process' current state, and a resource segment describing external entities that might be essential to task completion. In the process migration, both its code segment and movable state segment will be transferred over the network. Therefore, a process of detaching from its current computing environment will first be required, then its transferable state will be extracted and its communication redirected. A destination process instance will be created, and will be ready to be activated as long as the required execution state is transferred. Various algorithms to support homogeneous and heterogeneous process migration have been presented, such as the total copy algorithm used in Amoeba (Tanenbaum & Mullender, 1981), the demand page algorithm for Accent (Rashid & Robertson, 1981), and the freeze-free algorithm suggested by Roush (1995).

Process migration produced a fundamental impact on network computing. If one views process migration as a starting point, then Remote Evaluation (RE), Code-on-Demand (CoD), mobile object, and mobile agent will form a natural line of the technology evolution. From the state mobility viewpoint, at least two basic forms can be identified: weak mobility and strong mobility (Fuggetta et al, 1998). For weak mobility, only the code segment is transferred from one host machine to another, perhaps with some initialised data. Simplicity is an obvious advantage, as the destination machine is only required to execute the code from a foreign site. The code migration in RE and CoD belong to this type. By contrast, strong mobility allows both the code segment and the execution state segment to migrate across the network. Powerfulness and complexity are two principal characteristics, because a running process must be suspended at the source machine and execution resumed at the destination. Platform supports and security concerns are two major issues that have to
be considered when migration takes place in a heterogeneous environment. Most mobile object and mobile agent systems support this kind of mobility.

2.3.3 Remote evaluation and code-on-demand

As a type of weak mobility, remote evaluation gives a programmer enough flexibility to execute a procedure at a remote server. With the conventional RPC, the client has to make multiple calls for carrying out a given task. Longer completion time is a major inconvenience because the time is roughly proportional to the number of calls, although in most circumstances only the results from final calls are useful. The remote evaluation seeks to address this problem, as shown in Figure 2.6:

![Figure 2.6: The code migration in RE](Adapted from Wong et al, 1999)

RE is a type of sender-initiated migration -- that is, the mobile code owner initiates the request and uploads the code segment to the server side for computation (Volpano, 1996). The code segment that a client sends to the remote server must be self-containable. That is, all functions and variables referenced in the procedure must be included. The format of transmitted procedure relies on the expected level of security and the heterogeneity of the network. As Vigna (1997) noted, RE always takes place where the client holds the know-how (i.e., the code segment needed for execution), and the server holds the resources (i.e., some sort of relatively static components to support the service). In contrast to the conventional client-server computing (where
the server side holds both resources and know-how, while the client side can only send a passive request), the RE technology improves the availability of server-side customisable services. PostScript printer is one of the successful examples as the PostScript programs can be viewed as know-how, while the remote printing devices are server-held resources. The PostScript file is transmitted from the client to the server and executed by the interpreter hosted by the PostScript printer (Dearle et al, 1991). Some database systems also support executing a set of user-provided SQL commands and sending the final results back after the computation has been completed (Borr & Putzolu, 1988). NCL (Network Command Language) and SUPRA (SUbprogram PaRAmeters)-RPC are two variants of RE, with the major differences being in client and server stubs handling. More information can be found in Falcone (1987) and Stoyenko (1994).

Compared with the sender-initialised code migration in RE, the code migration in CoD is receiver-initialised. From the know-how/resources viewpoint, the client in CoD holds the resources while the server holds the know-how, which is desirable due to the fact that knowledge is always distributed over the network and shared by multiple users. When a client needs such know-how to perform computations, it simply sends a request to the server. Consequently, the active code will be downloaded to the client side and executed in the client’s machine, see Figure 2.7:

![Figure 2.7: The code migration in CoD](Adapted from Danny & Oshima, 1998)
For security reasons, a server that provides RE service always needs its clients authenticating themselves before running their uploaded code. By contrast, the CoD service is open to nearly all anonymous users, and interestingly, a similar security consideration shifts to the client side. If allowing the foreign code to be executed at the target directly, the client machine would face the same risks associated to its computing environment. Running the incoming code in a separate process is a common solution, however, increased communication costs are an obvious drawback. Java Applets is a classic example of the CoD implementation. When a client's browser reaches it, the applet is downloaded to the client machine and executed there. Java applets are compiled to a bytecode, which is not allowed to access any file in the client machine without proper authorisation. Another example of the CoD implementation can be found in ActiveX technology (Birngruber et al, 1999). The primary aim of ActiveX is to enable software components to interact with each other within a networked environment. ActiveX-control works as the server components holding compound documentation services. It is visible but not activated until a client interacts with it, thus saving time and resources needed for network transmission. Due to the distributed nature of knowledge over the Internet, the CoD technology has shown its great potential to be employed in many upcoming network-based applications.

2.3.4 Object migration and mobile objects

Despite multiple advantages introduced by RE and CoD, at least one drawback can be noted. The migrating process always encounters difficulties in encapsulating more state information at the source machine and re-presenting these at the destination site. The main reason is that hosts running within a distributed network may have different
Operating Systems (OS), and processes are always OS-dependent with some of the variables linking directly to the underlying OS data structure. Adding process migration functionality without impacting the stability of the existing OS is not an easy task. Object-Oriented (OO) computing makes this task simpler. By defining the boundaries of all system entities with the concept of objects, more program behaviours can be encapsulated within an object, and manipulated through a standardised invocation interface. As OO programming techniques mature, object migration (or mobile object) starts to attract growing attention, and can be viewed as an OO version of process migration. Encapsulating executable code and data in the form of object-specific methods and properties, a mobile object is expected to carry both of them to a remote platform for task fulfilment, as illustrated in Figure 2.8:

![Figure 2.8: A simplified working mechanism for object migration](Adapted from Wong et al, 1999)

The Emerald system is a classic example that supports object migration, and as Wong et al (1999) argued, might lead most directly to the development of mobile agents. Under Emerald's classification, objects that only contain data are termed passive objects, while those containing processes are termed active objects. Both passive and active objects can migrate from a source machine to a destination machine in a local network. For those objects with active invocations, all activation records will be identified and moved to that specified node with the object migration. Related references will be updated, while unreachable objects will be garbage-collected. Three
major benefits for object migration are observable: optimised data movement, improved object invocation, and simplified garbage collection (Jul et al, 1988). However, reduced system performance is always a drawback of implementing object mobility. Compared with local invocation, a mobile object only shows its advantage when communication is frequent. Some other systems that support object migration can be found within the SOS (Secure open multimedia workstation Operating System) presented by Shapiro et al (1989), and the COOL (Chorus Object-Oriented Layer) system developed by Amaral et al (1992). The major difference lies in that the SOS uses proxies to maintain intra-object references during an object migration, while the COOL system manages all internal references through pointer swizzling.

2.3.5 Mobile agent computing: benefits and challenges

Being different from an arbitrary object migrating on the network, a mobile agent always represents its owner. From the know-how/resources viewpoint, Vigna (1997) identified the mobile agent technology as moving an agent's know-how to the server's resources. Actually, mobile agents more likely work on a peer-to-peer network. A sketch of the mobile agent computing is illustrated in Figure 2.9:

![Figure 2.9: A sketch of the mobile agent computing](Adapted from Lange & Oshima, 1998)

Agent migration is initialised by a sender platform. Once an agent is ready to leave, the agent's execution thread will be halted, then the agent class and its execution state
will be serialised by the platform engine. Based on certain transport protocol between
the sender and the receiver, the serialised agent object will be further encoded. On the
receiver side, an incoming data stream will be decoded at first. If everything proceeds
well, the agent object will be de-serialised, the agent class will be instantiated and the
agent state restored. The re-created agent instance will ask for a new thread for
execution at the destination platform. To support the migrated agent resuming its task,
auxiliary classes may be also needed. The transferred agent can make local use of
those classes available at the destination site. For classes not available at the
destination, an additional network connection will have to be made to retrieve them
on a code-on-demand fashion.

Just as the process migration has significant advantages over the conventional
RPC, mobile agent technology seeks to improve network programming on a deeper
level. A number of major benefits brought by mobile agent computing have been
identified, while a collection of applications that might employ mobile agents has also
been specified (Green et al, 1997). A brief summary is given in Table 2.1:

<table>
<thead>
<tr>
<th>Major benefits</th>
<th>Potential applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reducing network load</td>
<td>Parallel processing, data mining…</td>
</tr>
<tr>
<td>Overcoming network latency</td>
<td>Remote searching and filtering, information dissemination…</td>
</tr>
<tr>
<td>Encapsulating protocols</td>
<td>Workflow applications and groupware…</td>
</tr>
<tr>
<td>Naturally heterogeneous</td>
<td>Distributed information retrieval, secure brokering…</td>
</tr>
<tr>
<td>Dynamic adaptation</td>
<td>Semantic routing, electronic commerce…</td>
</tr>
<tr>
<td>Asynchronous and autonomous</td>
<td>Personal assistance, monitoring and notification, electronic commerce…</td>
</tr>
<tr>
<td>execution</td>
<td></td>
</tr>
<tr>
<td>Robust and fault-tolerant</td>
<td>Telecommunication network services…</td>
</tr>
</tbody>
</table>

Table 2.1: Major benefits and potential applications of mobile agent technology

Reduced network latency and improved network traffic are two major benefits
inherited from process migration. The client-specific know-how is normally small in
size. When such a component moves over a low-bandwidth network and interacts with large volumes of data stored at the remote site, computational performance can obviously be improved. However, as Straer and Schwehm (1997) argued, agent migration outperforms RPC only when the amount of data to be transferred is larger than the size of the agent. For a sequence of interactions where an agent can use either migration or RPC to interact with its partners, a combination arrangement always performs better than a pure sequence of any single form. These observations may help an agent developer to optimise mobility design while seeking to employ an agent to perform a given task in a communication-efficient consideration.

Protocols encapsulation and heterogeneous operation are inherited features from mobile objects. A mobile agent is expected to capture its execution state at the current platform, encapsulate transmission protocol and travel to a different platform to restart its operation. This OS-independent nature makes mobile agents extremely desirable for distributed information retrieval and system integration. However, mobile agents are still dependent upon their execution environments. The mobile agents and their platforms must be written in the same language that supports code migration. In addition, wide differences between system architecture also need to be considered if seeking seamless integration between mobile agent applications and various legacy systems.

Asynchronous execution, dynamic adaptation, and fault-tolerance are extended properties of generalised software agents. As Harrison et al (1997) noted, these significant improvements enable mobile agent technology to provide better support for mobile clients, where connection always has a relatively low bandwidth, and access to a server is intermittent. With the help of the mobile agent paradigm, mobile clients can dispatch an agent during its brief connection to the network and then let
the agent carry out tasks asynchronously while disconnected. Actually, asynchronous interaction exists within many message-buffering systems to deal with unreliable network connections (Lamport & Lynch, 1990), but asynchronous execution means more to a self-contained mobile agent. A mobile agent equipped with intelligent components can better sense the changes within its working environment and adapt itself dynamically for task completion. Applications clearly benefiting from these mobile agent-specific properties include: semantic routing, remote real-time control, electronic commerce, and many more.

However, Lange (1999) argued, "Do not waste your time searching for the killer application for mobile agents, there is no mobile agent applications". Harrison et al (1997) seemed to arrive at a similar conclusion. Facing almost every expected advantage brought by mobile agents, there is an equivalent solution that already existed. On the one hand, they pointed out a mobile agent does not gain an obvious upper hand compared with other alternative means while assessing all of its individual advantages. On the other hand, they also acknowledged that mobile agent technology provides an aggregate advantage that no individual technique possesses, that is, a mobile agent framework seems to address all of these at once.

Therefore, it is necessary to point out that the technology of mobile agents for e-business may be significantly advantageous and thus preferred due to time criticality, infinite flexibility, and cost effectiveness although it is difficult to find 'killer' applications for mobile agents. For example, in Internet trading of the financial markets, agents may reside on the market exchange data server to monitor the market movement, the agents then communicate alert signals back to the user only when certain profitable patterns of market behaviour appear. This requires time criticality.
for profit opportunities and risk management, and for saving the cost of keeping real-time quote data streams.

2.4 The development of mobile agent platforms

In order to investigate mobile agents' aggregate advantages more comprehensively, this section seeks to view mobile agent platform development in an outside-to-inside manner: from mobile code language support to mobile agent execution environment, from host-to-host transportation mechanisms to agent-to-agent communication services. Such an arrangement is not intended to review agent systems based on their availability and certain features, as Nwana (1996) and Green et al (1997) did, but to focus on how various structural and functional components help mobile agents achieve their technical advantages.

2.4.1 Making use of mobile code languages

Mobile code language is a fundamental component for mobile agent platform development. A mobile agent execution environment not only facilitates its agent's computation, but also manages their communication and migration. Indeed, much of these basic functions come from their written languages natural support. Mobile agents are supposed to make good use of those language-specific properties to better implement their computational advantages. Therefore, classifying mobile agent systems based on their created language is quite meaningful. Several major languages that support code mobility are listed in Table 2.2:
<table>
<thead>
<tr>
<th>Languages</th>
<th>Code portability</th>
<th>State mobility</th>
<th>Execution security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facile</td>
<td>Multiple representations</td>
<td>Weak</td>
<td>Inadequate</td>
</tr>
<tr>
<td>MØ</td>
<td>Interpreted</td>
<td>Weak</td>
<td>Inadequate</td>
</tr>
<tr>
<td>Obliq</td>
<td>Interpreted</td>
<td>Weak</td>
<td>Inadequate</td>
</tr>
<tr>
<td>Tycoon</td>
<td>Intermediate TML</td>
<td>Strong</td>
<td>Three-level support</td>
</tr>
<tr>
<td>Telescript</td>
<td>Interpreted</td>
<td>Strong</td>
<td>Four-level support</td>
</tr>
<tr>
<td>Tcl</td>
<td>Interpreted</td>
<td>Strong</td>
<td>Safe-Tcl model</td>
</tr>
<tr>
<td>Java</td>
<td>Intermediate JBC</td>
<td>Weak</td>
<td>Sand-box model</td>
</tr>
</tbody>
</table>

Table 2.2: Several major mobile code languages

In fact, most languages are not specifically created for mobile agent development. Some of them are claimed to be high-level functional languages supporting concurrency and distribution computing (such as Facile and Tycoon). Others, like MØ and Obliq, try to facilitate the development of communicating applications, and language like Java is natively network-oriented. But all the languages suitable for mobile agent programming must share one basic characteristic -- that is, the support for code portability and state mobility. Code portability helps an application program writing once and executing anywhere without the need to worry about differences on the underlying hardware. State mobility helps a migratable unit capture sufficient execution states and resume computation at a destination machine.

**Facile**  The primary aim of Facile is to support concurrent and distributed programming through defining a set of first-class functions for code manipulation (Knabe, 1995). As a higher-level functional language, an agent in Facile is simply represented as a function. Facile language supports transmission of multiple representations. On the receiver side, a lazy run-time compilation approach is adopted (Chambers *et al*, 1991), where the functions that make up an agent will be compiled only when they are called for the first time. With respect to state mobility, Facile language's support is not very strong. Only the code pointer and some collection of
singular values can be captured, marshalled, and re-stored in a different computing environment. For those states that must be shared, a separate library for distributed references has to be introduced explicitly. Reference cells pointed to the bound resources at the sending machine will be copied and re-established at the destination site. Additionally, some other features such as strong typing, automatic memory management and stand-alone execution are also good for mobile agent applications construction. Although these mechanisms are valuable for pieces of mobile code running at different computers more safely, Facile language does not impose any explicit security model in its design. In summary, as Knabe (1995) argued, Facile is a good starting point for developing agent language, but not complete per se.

**MØ** (M-Zero) means the first-generation language for Messengers (Tschudin, 1994). Messengers are not real mobile agents, but mobile code fragments that can execute at a remote platform. MØ is a pure interpreted language, but is not object-oriented. Within the MØ platform, its interpreter helps an incoming messenger execute on a heterogeneous environment and access local resources straightforwardly. Each messenger contains a header, a code segment, and a separate data field. Because the original motivation of messengers is to provide code-exchange functionality, a programmer may submit a messenger to a remote computer for execution in the same way as sending a message through traditional communication channels. Relevant data necessary for remote execution is copied and carried with the messenger to the destination host. After reaching the receiver side, each messenger becomes an independent process, and works within its own memory space. The receiver platform provides a shared memory area in the form of a global dictionary for incoming messengers to deposit their private data. In addition, a process queue is created based
on individual process's identifiers in order to synchronise multiple messengers' activities. MØ does not have any explicit security protocol. Basically, a messenger from any origin would be accepted. However, some protection mechanisms in shared memory do exist, each entry in the shared memory is encrypted by its creator's private key. It is readable by all other visitors, but modifiable only by its owner. In summary, MØ language provides a minimal core to support the operation of a mobile code system, but the functionality of a messenger is far from what expected mobile agent in many ways such as execution-state capture, automatic navigation, and secure computation.

**Obliq** This language can be summarised in four major features: interpreted, untyped, object-oriented, and lexically-scoped (Cardelli, 1994). In order to support distributed computing, each Obliq object is network-transparent. These can be cloned to different sites and dynamically allocated across the network. Unfortunately, Obliq language does not directly support object migration. An object created at a site is local to that site. In other words, the identifier for that object is bound to the data space of its original computing environment. When an object moves, references to this object are automatically translated into network references. Such a binding arrangement (or lexical scoping) makes distributed computation much simpler, because migrated computation will have a precise meaning which is determined by their original binding location, not by their current execution site. An application programmer never needs to worry about the physical location of a migrated object, as operations will be transparently redirected to its original site and executed there. A procedure with no free identifiers can be viewed as a self-contained agent, thus enabling disconnected operations. From an execution security viewpoint, an incoming agent can only access
the resources they hold references via free variables, that is, no further operations can be performed outside its lexical scope unless it is granted to do so. But lexical scope rules are only an implicit form of safe execution, so no explicit security mechanism can be found in Obliq. In summary, Obliq performs relatively well in dealing with OO programming with the aid of distributed lexical scoping mechanism. However, as Cardelli (1994) claimed, as Obliq is still in its infancy, many mature features such as authority delegation, state mobility, and security arrangement are still lacking.

**TYCOON** It is short for 'TYped Communicating Objects in Open eNvironment' (Mathiske et al., 1997), and its primary aim is to integrate a highly generic language kernel into an open database-programming environment. On the linguistic level, Tycoon defines data, code, and threads all first-class computational entities. They can be passed as arguments, returned as results or migrated freely among different network nodes. The Tycoon compiler at the originating site is responsible for generating abstract machine code. At the target machine, a separate TML (Tycoon Machine Language) evaluator works as an interpreter in charge of loading the target code into local address space for execution. Tycoon is classified as a strong-mobility language with respect to state mobility (Matthes & Schmidt, 1994). In Tycoon, a running thread with various binding information might easily migrate to another network node with a simple 'migrate to …do' command. The basic semantics behind the scene is a deep-copy operation between two address spaces and a remote migration engine helps the migrated thread perform dynamic bindings to the local site resources for computation resumption. As for language security, Tycoon provides three-level support: application level, machine level, and object-store level. On the application-level, an authentication service is facilitated through a type-safe Tycoon
language interface with the help of a backend security library. On the machine-level, a Tycoon-kernel controlled Tycoon Machine ensures that only specified principals are able to manipulate objects with linked applications. On the object-store level, an explicit evaluator protocol is employed to ensure all accesses to stored objects are well controlled. In summary, Tycoon language provides a very good support for code portability, state mobility, and execution security, and the complete binding mechanism makes it quite suitable for mobile agent programming. As a mobile agent prototype, a migratable thread visiting different network hosts to perform a long-term activity has been broadly outlined (Rudloff et al, 1995). However, Tycoon language, together with three other languages presented above, did not develop into a fully-fledged mobile agent language. One possible reason is that their initial motivation was to improve certain communication functionality in distributed programming, not to build complete mobile agent platforms. Their supports for mobile agents' development are merely for research purposes.

2.4.2 Setting up common execution environments

A complete mobile agent system is required to make use of various language-specific characteristics, establish a common execution environment and support all mobile agents-related activities (some examples are listed in Table 2.3). The following subsections do not attempt to review all these systems, but to select three (Telescript, Agent Tcl and Aglets) and draw a cross-language comparison. Focuses will be placed on their language selection and running environment construction, migration arrangement (see Subsection 2.4.3), communication facilities (Subsection 2.4.4), and security mechanisms (Subsection 2.4.5).
<table>
<thead>
<tr>
<th>Languages</th>
<th>Mobile agent system examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telescript</td>
<td>Telescript</td>
</tr>
<tr>
<td>Tcl</td>
<td>Agent Tcl (D'Agents), Ara, Tacoma.</td>
</tr>
<tr>
<td>Java</td>
<td>Aglets, Concordia, Mole, Grasshopper, Odyssey, Voyager, Jumping Beans, FarGo…</td>
</tr>
</tbody>
</table>

Table 2.3: Some mobile agent system examples

Telescript supports mobile agent development at both language level and system level. Released by General Magic Inc. in 1994, it was claimed to be a very early commercialised mobile agent system (Cockayne & Zyda, 1997). Unfortunately, Telescript has largely been withdrawn from the market nowadays. As Gray et al (2000) analysed, the rapid popularity of Java and its successful support for mobile agent construction might be a major reason. However, Telescript introduced a collection of mobile agent-specific concepts, which have been well recognised as very fundamental to mobile agent computing. From the language perspective, White (1994b) summarised six good qualities of Telescript language such as object-oriented, complete, portable and safe, dynamic, persistent, and communication-centric that are ideal for communication applications development. From the code portability perspective, a Telescript application is compiled into bytecode first, which is much like Java. The slight difference is that there is no strong security control performed during the compilation process. An interpreter (or Telescript engine) at each receiving host is responsible for safely executing these bytecodes. As a computing environment abstraction, Telescript engine maintains two other important execution units: places and agents, as depicted in Figure 2.10:
A Telescript engine can host many virtual places, with each virtual place supporting many local and foreign agents. In fact, places do not execute agents directly, they simply provide a conceptual space for agents' execution. It is the Telescript engine that is in charge of linking the underlying communication network for agents' transportation, and facilitating their access to local resources for their computation. The concept of 'engine' is quite clear in many existing agent systems, but 'places' might not be clear, such as in Agent Tcl. For that case, as Lange (1998) suggested, the engine itself could be viewed as a place.

Tcl (Tool Command Language) is another language that has been successfully used for mobile agent platform construction (Ousterhout, 1994). Agent Tcl (Gray, 1997), ARA (Agents for Remote Access) developed by Peine (1997), and TACOMA (Tromsø And COrnell Mobile Agents) constructed by Johansen et al (1995) are all written in Tcl. Unlike Telescript that includes 'script' in its name but not a script language, Tcl is a script language. Ousterhout (1994) summarised four benefits of employing Tcl language for mobile agent applications: cross-platform, rapid development, extensible, and secure. The cross-platform nature ensures that an application does not need to be re-compiled to meet different execution requirements for each target machine. This extensibility enables some useful functionality to be created in an added-on fashion. For example, the standard Tcl interpreter does not
have facilities to capture the internal state of a running script. However, such a function is essential for mobile agent's migration. To this end, Agent Tcl system explicitly adds a stack to the Tcl original interpreter for saving and restoring the internal state of an executing script. Unfortunately, slow performance is an obvious drawback with such an arrangement, as modified safe-Tcl always runs slower than standard Tcl interpreter. From the execution environment viewpoint, Agent Tcl employs a five-layer architecture, as shown in Figure 2.11:

![Figure 2.11: The execution environment in Agent Tcl](Gray, 1997)

The lowest layer provides a common interface to the underlying transportation facilities, which is similar to that in the Telescript architecture (although the external application API is not included). The engine layer within the Telescript structure is divided into three parts in Agent Tcl (see Figure 2.11). The basic reason is that Agent Tcl supports multi-language interpretation, currently including Tcl, Java and Scheme (Rees & Clinger, 1986). For each incoming agent, there is an appropriate interpreter to execute. Each interpreter incorporates a component for language interpretation, a special module for state capture, a module for security-enforcement, and a server API. The generic core works together with the server layer to implement many other agent-associated functions such as receiving an incoming agent, dispatching an outgoing
agent, providing the agents with appropriate computational environments, and facilitating agent-to-agent communication.

Aglets, Concordia (Wong et al, 1997), and Grasshopper (Breugst et al, 1998) are all written in Java. In addition the examples listed in Table 2.3, there are many other systems available in literature. A comprehensive listing of mobile agent systems supported in Java can be found in Kiniry and Zimmerman (1997). Roughly estimating, the total number is more than twenty. As a network-oriented programming language in nature, why is Java extensively adopted for mobile agent development? As Lange and Oshima (1998) observed, there are six positive properties that make Java ideal for mobile agent programming. Interestingly, they also listed four associated shortcomings, see Table 2.4 for detail.

<table>
<thead>
<tr>
<th>Benefits of Java for mobile agent programming</th>
<th>Drawbacks of Java for mobile agent deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform independence</td>
<td>Inadequate support for resource control</td>
</tr>
<tr>
<td>Secure execution on the network</td>
<td>No protection of references</td>
</tr>
<tr>
<td>Dynamic class loading via the network</td>
<td>No object ownership of references</td>
</tr>
<tr>
<td>Multithread programming</td>
<td>No support for preservation and resumption of the execution state</td>
</tr>
<tr>
<td>Built-in serialisation mechanism</td>
<td></td>
</tr>
<tr>
<td>Reflection</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.4: Benefits and drawbacks of Java for mobile agents' development

Clearly, Java-based mobile agent system designers are expected to take advantage of the positive properties in the list and overcome the drawbacks. IBM's Aglets is such an example. Several phrases can be used to characterise an aglet agent: built with persistent support, lightweight objects migration, and event-driven. Compared with the three-layer architecture in Telescript, and the five-layer architecture in Agent Tcl, Aglets provides a four-layer architecture like a pyramid (see Figure 2.12).
Interestingly, if looking at this figure in reverse order, it is not difficult to see the basic structure is almost the same as that in Telescript, but simpler than that in Agent Tcl (as Aglets does not support multi-language implementation). Engine is also the core of the Aglets infrastructure. It is described as a workhorse used to support two other lower-level components: places and agents (Lange & Oshima, 1998). In Aglets' development kit ASDK, engine is implemented as the Tahiti server. Multiple Tahities could run on a single host at different ports, and each Tahiti hosting multiple places with different names. The major function of Tahiti is to establish the links with the underlying network and other resources hosted by the local machine. There is not much difference between the server/engine in Telescript or Agent Tcl, but Tahiti provides a graphical user interface, through which application developers can monitor and manage an aglet's behaviour more conveniently. The concept of place is very clear in Aglets, as that in Telescript. In ASDK, a place is implemented as a context, each context has a unique name to identify itself within the engine. Conceptually, Aglets' context is much like Java Applets' context, also in charge of protecting the hosting system from being attacked.

From the above discussion, it can be noticed that the execution environment works as a glue to link mobile agents with their underlying hardware structures. The
engine (or server) undertakes much of the hard work to support mobile agents working in a heterogeneous environment (such as agent migration, inter-agent communication, and system security mechanism). Therefore, an execution unit is not necessary to be bound to a single operating system in its whole life. It can choose to migrate to any remote host if only a uniform execution environment is provided.

2.4.3 Providing supports for agent migration

Agent migration is one of the most important tasks for an agent platform to facilitate. As with the introduction of associated migration-specific functions, a mobile agent's lifecycle model also has to make certain adjustments. Two types of lifecycle models are therefore suggested. One is the process-based model adopted by Telescript, Agent Tcl and ARA (see Figure 2.13), the other is the event-based model adopted by Aglets (see Figure 2.14).

![Figure 2.13: A process-based life cycle model of a mobile agent](Adapted from Green et al, 1997)

In Telescript, every piece of information can be coded as an object with a standard external interface and an explicit internal implementation. This four-state life cycle is a special property that can be used to distinguish a mobile agent from other objects (White, 1994b). A mobile agent starts its life with initialised values within a local place. In its running state, the agent carries out a given task, where a persistent process is in its full execution. When a mobile agent wants to move, it enters the frozen state where its running state will be check-pointed, and all its code, data, and
thread state will be captured and packaged. After arriving at the destination, the agent resumes its execution at the point it left off, and re-enters a new running state. In Telescript, such an operation is termed as 'travel', with a simple 'go' command. During the agent's travel, the bound resources owned by the agent will be deleted from the originating host and a new copy of these resources will be established at the destination. At the transportation level, two protocols will have been identified: agent encoding/decoding and engine-to-engine authentication, which guarantee a mobile agent navigating over the network safely.

A similar situation happens in Agent Tcl. It also provides strong mobility for agent migration. The complete state of an agent will be captured and shipped to a new machine. The counterpart command of 'go' instruction in Telescript is the 'agent_jump' within Agent Tcl. Such simplification hides many details of the state-information collection process from an application programmer. Actually, as Tanenbaum and Steen (2002) analysed, an agent's state contains four tables and two stacks recording its global definitions of variables and its current command execution status. All of this information has to be marshalled into a byte array and sent with the agent to a destination machine. On the target machine, these machine-independent data will be un-marshalled in reverse order, and the agent will resume its execution at the exact point at which it was suspended. The Agent Tcl engine is in charge of running a proper interpreter to help the agent continue its computation. In addition to the 'agent_jump' instruction, Agent Tcl provides two other kinds of code migration through two additional commands: 'agent_submit' and 'agent_fork'. The 'agent_submit' instruction lets an agent submit a script to a remote machine for execution, the working mechanism being much like a sender-initiated RE. From the state mobility viewpoint, this code migration style belongs to weak mobility, because
with the exception of some procedure definitions and copies of variables the submitted script does not need to carry its execution state to the destination machine. The 'agent_fork' command allows an agent to clone a process at the remote site while continuing its execution at the source machine. As Tanenbaum and Steen (2002) noted, this code migration style belongs to a special kind of strong mobility, which is like the 'fork' command in UNIX. Compared with the single 'go' command in Telescript, a mobile agent in Agent Tcl has more choices to fulfil its migration tasks.

In contrast to the process-based model in Telescript and Agent Tcl, the life cycle model adopted by Aglets is truly event-driven. Seven events might happen in an aglet's life: creation, cloning, dispatching, retraction, activation/deactivation, and disposal. All these events will take place within (a) context(s), as depicted in Figure 2.14:

![Event-based life cycle model of Aglets](image)

Figure 2.14: An event-based life cycle model of Aglets (Lange & Oshima, 1998)

Creating and cloning events may bring an aglet to life in a local context. A cloned aglet is no different from an original except it has a different identifier. Dispatching and retracting events are closely related to the agent migration. When an aglet wishes to move, it calls a 'dispatch' method, which is similar to the 'go' instruction in Telescript or the 'agent_jump' command in Agent Tcl. An aglet object will be removed from its current working context and re-created at a remote context. The
'retract' operation works in the reverse manner, withdrawing an aglet from a remote context when necessary. Disposal is a useful way to remove an aglet from its working context in order to release its holding resources, while deactivation is another way to reduce aglets' resources consumption (where an aglet with its data and execution state can be pushed out of current context temporarily and held in secondary storage). A deactivated aglet can be activated and restored back in the same context when needed. The 'deactivated' and 'activated' operations are special characteristics of Aglets, which cannot be found in either Telescript or Agent Tcl. They significantly improve an aglet's fault tolerance capability.

In concept, an aglet's migration belongs to weak mobility, which is different from the agent migration manner in Telescript or Agent Tcl. The basic reason can be traced back to its written language -- Java. Java takes a Java Byte Code approach to achieve code portability. When these architecture-neutral bytecodes are shipped to another machine, a Java Virtual Machine (JVM) is responsible for their interpretation. Therefore, running mobile agents in a heterogeneous environment is not a problem. However, as for the state mobility, Java shows both strengths and weaknesses. On the up side, as Java has a built-in serialisation mechanism, an object structure can be conveniently saved in a serialised form, transferred over the network and reconstructed in another machine. On the down side, Java does not support strong mobility because the full execution state of an object cannot be completely captured. The program counter and frame stack information are never retrievable due to obvious security concerns. As an alternative, a migrated agent needs to rely on its internal attributes and external events to help it resume computation on a remote platform. However, such weakness seems to have no serious effect on an aglet's migration. In fact, execution stack information may not have the same meaning at different JVM,
even if they are captured and transferred across the network. For that reason, as Lange
and Oshima (1998) claimed, a weak mobility arrangement is sufficient for an aglet
migration. A standard Java object serialisation mechanism is used to marshal an aglet
object with part of its execution state into a stream. On the receiver machine, two
other methods 'onArrival' and 'run' are employed to perform application-specific
initialisation and execution resumption. In addition, an Agent Transfer Protocol
(ATP) is also specified for communicating engines to encode and decode the
serialised agents and make their transfer independent of the underlying hardware
facility.

In summary, although concrete migration instructions are different in various
platforms, the underlying platform engine carries out similar functions: suspending an
agent's work, capturing transferable state, packing all relevant data, and shipping the
marshalled byte stream to a remote machine through an established communication
channel. On the receiving site, a reverse procedure will be performed to help a
migrated agent resume its execution. From the state mobility viewpoint, Telescript
provides strong mobility and Aglets provides weak mobility. Agent Tcl stands out in
this perspective because it provides an agent with three choices for remote execution:
direct migration, remote submission, and process cloning. From the mobile agent
lifecycle viewpoint, Telescript and Agent Tcl adopt the process-based model, while
Aglets employs the event-based model, which is outstanding in this perspective as it
provides both an initiative and a passive means of mobility. Both methods can help an
agent handle migration as a routine procedure in its life. No further work needs to be
done at application level to collect migration-related information for migration.
2.4.4 Enabling agent-to-agent communication

Enabling agent communication is another important task for mobile agent system design. The primary aim is to keep an agent informed about its working environment effectively or to help it obtain remote services efficiently, because direct migration is sometimes a costly operation, especially when the agent is large in size.

Telescript defines two basic concepts as its communication services: 'meetings' and 'connections'. The 'meetings' operation enables agents residing at the same place to call one another's procedure, while the 'connections' operation helps agents residing at different places to establish connection. For the 'meetings' instruction, a meeting place and the identity of a partner must be specified. For the 'connections' operation, additional terms such as quality of service and connection deadline have to be included. With the help of these two communication methods and the agent migration mechanisms (presented within the previous subsection), a mobile agent-mediated e-business scenario is easily outlined. A buyer mobile agent is 'travelling' to several 'places' (e.g., e-markets) in succession, 'meeting' with local stationary agents (e.g., market operators or seller agents) while keeping 'connections' with other distant agents (e.g., an agent at home). That might be a very early prototype for mobile agent-supported e-business application presented in the literature (White, 1994b).

Compared with the simple communication methods in Telescript, Agent Tcl makes a significant improvement. It provides two communication mechanisms: direct connections and message passing. Interestingly, the direct connection method uses a similar instruction 'agent_meet' as in Telescript, but it is used not for two agents at the same place calling each other's procedure, but for bulk data transfer between two communicating agents. So, its function is more like the 'connections' operation within Telescript (ignoring the connection deadline). Such direct connection is especially
good for long-time interaction. As for short-term interaction, Agent Tcl provides
several message-passing functions. Arbitrary strings or binary data can be sent as a
message with the aid of 'agent_send' and 'agent_receive' primitives. In case of remote
messaging, the engine is also responsible for buffering each incoming message and
selecting a proper mechanism for outgoing messages. To facilitate more complex
communication, event passing -- where an agent can send an event to another agent
with the help of an event handler -- is also supported. In addition to these low-level
communication methods, higher-level communication services such as 'yellow pages'
agents or RPC service can be introduced in an add-on fashion.

Communication mechanisms in Java-based mobile agent systems are also rich
and powerful. From the language viewpoint, Java not only supports multi-thread
programming, but also provides a set of synchronisation primitives to facilitate multi-
agents communication and collaboration. A Java object can call each other's public
methods if only holding a reference to it. Unfortunately, as Lange and Oshima (1998)
noted, this also opened an operational loophole. A disused agent could not be
automatically garbage-collected while live references to it still existed. In ASDK, the
major mechanism for multi-agent communication is through message passing, which
can be characterised as location-independent, extensible, and pattern-oriented (Lange
& Oshima, 1998). More specifically, one aglet can send a message to another by
invoking a 'sendMessage' method via its proxy, which successfully hides an agent's
real location and makes remote messaging being handled in exactly the same way as
local messaging. Inter-aglets messaging may happen in an either synchronous or
asynchronous way. For the former, further execution will be blocked until a reply is
fed back. For the latter, the sender may continue its working after message delivery. If
a sender does not expect any reply, he/she may simply send a one-way message in a
fire-and-forget fashion. In addition to these one-to-one communication approaches, a set of one-to-many communication patterns is also available in Aglets, message multicasting being a powerful example. Its power lies in that communicating agents do not need to know the identities of one another, but are required only to subscribe to one or more multicast messages and implementing associated handlers.

In summary, as one of the latest computing paradigms, mobile agents still need conventional mechanisms to communicate with one another, because agent migration is communication-efficient only when the mutual communication is extensive and the size of the agent is small. From the communication approaches viewpoint, message passing remains the basic one. The content of a message might range from arbitrary strings to serialised objects. Various messaging patterns presented in Aglets make it outstanding in this perspective compared with Telescript and Agent Tcl, because they provide not only an ideal mechanism for inter-agent communication but also an important channel for multi-agent co-ordination.

2.4.5 Providing necessary security mechanisms

For any mature mobile agent system, security mechanism always plays an important role. In addition to the written languages that provide natural supports for agent computational security, specific mechanisms also need to be facilitated at system level for both agent and agent host protection.

Telescript has a complete four-level arrangement: object runtime safety, process safety, system safety, and network safety (Tardo & Valente, 1996). At the object-safety level, Telescript language removes the error-prone pointer arithmetic, so that objects can only be manipulated via a standard interface and through an effective reference. Additionally, four other services are included at this level: runtime type
checking, automatic memory management, explicit garbage collection, and exception handling. Therefore, Telescript is regarded as safe from the language perspective. At the process-safety level, the Telescript system engine provides two distinctive security mechanisms: 'authorities' and 'permits'. Each agent and place has a unique authority denoting an individual or organisation it represents, and being enforced by the digital signature technique. A place may accept or reject an incoming agent based simply on its verified authority. The 'permits' mechanism restricts an agent's capability of resources consumption, and prevents it from performing ill-programmed instructions.

At the system-safety level, system resources such as file systems and network accesses can only be manipulated by its local management authority. That is, a foreign agent has no way to execute unauthorised commands directly on the underlying OS. Only an 'External File' and an 'External Handle' object could be instantiated and used for system-resource access purposes. Finally, the network safety in Telescript involves a set of interconnected engines. Instead of authenticating an individual agent, Telescript platform carries out authentication on a region-to-region basis when dealing with an agent’s transfer over the network. The places that belong to the same region may have the same security policy. If two places belong to different regions, additional service restrictions would be set for agents' transmission. Although the security supports in Telescript have been carefully designed, some limitations have also been reported. Employing different mechanisms in different situations is not easy work for the novice programmer, because most of the mechanisms are realised by explicit coding, not by default. The lack of authentication procedure on an individual agent is also a weak point when considering an open Telescript network in general.

As for Agent Tcl, security arrangements can be analysed at two levels: language level and system level. At the language level, all variables in Tcl are simply
represented as strings. Different operators are responsible for interpreting these strings into different type-values for computation. Such typeless features simplify the type-checking process when executing a foreign code, but at the same time data protection against mis-manipulation becomes more complicated (Cugola et al, 1996). A twin-interpreter-based safe-Tcl model tries to address this problem: a standard Tcl interpreter works as a master, while a subsidiary interpreter acts as a slave. Untrusted codes will be loaded into the slave interpreter for execution where only a sub-set of safe commands can be performed. All other unsafe instructions will be redirected to the master interpreter and examined with further control. At the agent system level, two other mechanisms are used for security enforcement. Firstly, a digital signature-based agent owner authentication scheme is employed. A receiving host is able to assign different access restrictions to an incoming agent based on its verified signature. Secondly, a resource-manager is explicitly set up to define a list of access-control restrictions for the agent executing in its interpretation environment. However, a direct control on an agent's total resource consumption does not impose in Agent Tcl. As Gray (1997) argued, four additional problems related to agent system security can be observed: individual machine protection, group machine protection, individual agent protection, and peer-agent protection. Although most problems can be addressed through the public-key cryptography technique, balancing agent protection with overall system performance remains a difficult issue.

Just as the Agent Tcl system utilising many advantages of the safe-Tcl model, Java-based systems benefit from a sandbox model within the Java's security architecture. Instead of trying to prevent all potentially unsafe codes from being loaded into a local computer, the sandbox model tries to provide an isolated execution environment for each possibly unsafe operation. A downloaded Java program can
only play within a well-defined sandbox space, but not beyond its boundaries, thus protecting a local system from being maliciously attacked by a foreign program. However, Java has no direct control on an object’s resource consumption either, which is similar to the situation in Tcl. A malicious agent may simply consume a host’s memory resources irresponsibly and take it out of service. Therefore, strict resource controls have to be enforced at the agent-system level. In ASDK, the security model is principals-oriented. Four major principals are identified in Aglets: aglet manufacturer, aglet owner, context master, and domain authority. Security policies enforced by these four principals form a hierarchical structure with the aglet manufacturer at the lowest level and the domain authority at the highest. Lower-level principals can refine but not overwrite security policies set by their upper-level principal(s). When an aglet is created, its owner may define a set of security preferences and coding into the aglet’s body, describing the actions that can be requested by other principals. When an aglet is ready to depart, this static part will be digitally signed by its working context. In the destination machine, the context master maintains a policy database to fulfil security restrictions on the incoming aglets’ activities, which is similar to restricting downloaded applets running on a local browser. Common security policies can define access control on a general, organisational, or per-aglet basis.

In summary, security mechanisms can always be enforced at mobile code language level and mobile agent system level. At the language level, all three languages (Telescript, Tcl and Java) have adequate arrangements. Telescript provides object-level and process-level safety to support object manipulation and agent execution. Safe-Tcl model making use of a twin-interpreter tries to execute potentially unsafe scripts within a separate environment, and controls what they can perform. Java implements the sandbox model to achieve a special execution environment,
which has been incorporated into many aspects of Java's security architecture, such as JVM, the class loader, and the security manager. At system level, Aglets' principal-oriented arrangement performs very well. Compared with Telescript and Agent Tcl, Aglets' hierarchical model clearly introduces security principals with respect to their preferences and security concerns. The context owner's policy-base and aglet owner's security preferences form a hierarchical framework to protect each principal in its deserved way. This is regarded as an early step toward fully removing potential security threats within mobile agent systems (Karjoth et al, 1997).

2.5 Chapter summary

This chapter gives a general overview of the multi-agent and mobile agent technology with focuses on a historical view of the evolution of mobile agent computing, and a comparative view of the mobile agent platform establishment. Several observations from this chapter are summarised below.

- Primary and secondary properties of an agent

Defining a software agent based on its primary and secondary properties not only helps to distinguish an agent from an ordinary program, but also provides a meaningful way to classify most of the agent applications. To some extent, the combination of both primary and secondary properties that makes the agent technology more attractive. On the one hand, various secondary properties are not exclusive, but closely correlated to each other. The more properties incorporated into an agent, the more added values it may be able to contribute to the whole system. On the other hand, collaboration among agent members means the value-addedness of a multi-agent system will always be greater than the maximum value of any individual
agent, and with the potential of outperforming the simple accumulative sum of all its
agent-members.

- Mobile agent technology evolution
In technology evolution, an emerging technology may not replace the previous one(s)
completely. The mobile agent paradigm is a good instance in that it is not intended to
outdate all previous techniques fundamentally, but works collaboratively with other
approaches within distributed computing. Peer-to-peer messaging and client-server
RPC are very early technologies for network programming, different forms of these
are still employed within mobile agent systems. Agent-to-agent messaging in Aglets
and RPC service in Agent Tcl are two obvious examples. RE and CoD are two special
types of process migration, both play roles within mobile agent computing. Examples
include the 'agent_submit' command in Agent Tcl and remote class-file retrieving
mechanism in Aglets. Mobile agent computing presents an aggregate advantage
compared with other alternative technologies. A mobile agent is supposed to flexibly
choose various computing and communication means for its tasks fulfilment.

- Successful construction of a mobile agent system
Three mobile agent systems have been chosen for cross-language comparison:
Telescript, Agent Tcl, and Aglets. Telescript is an early example, however, a number
of legal and ethical considerations (such as agent authentication, secrecy protection,
privacy maintenance, and responsibility design) were not adequately addressed.
Additionally, an application programmer has to learn another language, which is
another weak point when compared with the Java-based mobile agent systems, and
might be a major reason for having been retracted from the market. Agent Tcl's merits
come from its high-level scripting language support, stronger migration mechanism, efficient agent communication approaches, and effective security protection. It is quite suitable for creating small-to-medium-sized applications, but more advanced functionality to be a commercialised agent system is still lacking. Slow performance is a serious disadvantage, which makes it unattractive for speed-critical applications. Besides, the Agent Tcl server and its agent run as separate processes, which might be another cause of inefficiency. Java-based mobile agents perfectly incorporate the advantages of Java language into the agent world. Hence, it occupies an important position in the current mobile agent study, with a wide range of applications. However, many open issues still need to be addressed before they can be quickly deployed for commercial uses. Most of these issues will be investigated in the following chapters of this thesis.

In the coming chapter, focus will be placed on the mobile agent-mediated e-business domain. Associated e-business models will be examined, seeking to explore how various online businesses can benefit from incorporating agent techniques to enhance their Internet functions via the network.
Chapter 3

E-Business Models with Mobile Agent Technology

3.1 Chapter introduction

The electronic market (e-market) might be one of the biggest promises brought by the Internet. In contrast to the traditional market of bricks and mortar, the online market makes goods and services available 24 hours a day, thanks to the Internet that is blurring the spatial and temporal limits between suppliers and consumers. From the businesses' viewpoint, the virtual market helps them automate business transactions and improve service qualities. Companies like online bookstores, auction houses and stockbrokers are all successful examples. From the consumers' viewpoint, the virtual market helps them acquire preferred products and services via the network with reduced searching and communication costs. In short, the e-market is changing the way people traditionally do businesses. The previous chapter has demonstrated that mobile agent computing is quite suitable for Internet-supported application designs.
Actually, mobile agent-mediated network applications (especially e-business applications) are the major forces in aiding wide acceptance of this emerging technology within commercial markets. This chapter will concentrate on the multi-agent and mobile agent-mediated e-business domain. Some e-business systems using mobile agent technology will be reviewed, and a collection of mobile agent-based e-business models will be suggested.

3.2 Agent-oriented e-markets

Although the e-market is different in form from the traditional market, the basic function is unchanged: to facilitate the exchange of goods and services. There have always been providers of goods and services, and others who search for and consume them. The advent of agent-mediated e-market, however, creates selling agents who represent vendors and buying agents who perform hunting and bargaining. A virtual market is an organisation where numerous selling and buying agents participate. Competition among multiple service providers is encouraged, while communication and negotiation between all participants are supported.

3.2.1 General business activities within a virtual market

In order to further examine how agent technology can be employed for e-business processes and which business activities a software agent may take part in within a virtual market, a set of generalised e-business processes presented by Villinger and Burger (1997) might be helpful, as illustrated in Figure 3.1:
Three working stages within these processes have also been identified: information phase, agreement phase, and transaction phase. At the information phase, associated information is collected. Some information (such as brand and prices) is related to products, while others are specific to potential partners and the trading environment such as merchant reputation, delivery time, and value-added services. At the agreement phase, direct negotiation between market participants is undertaken. Contents of the negotiation may include terms of transactions such as delivery conditions, payment forms and after-sale services, as well as other issues of mutual concern. At the transaction phase, actual purchase and delivery activities are completed. Based on different characteristics of exchanged goods, different goods delivery approaches and monetary-information transfer methods will be determined. While complete e-business agents may have difficulty to be accepted due to security of transactions, agents lend themselves best to the information and negotiation stages.

During the information stage, a buying agent decides what to buy and from whom. As information agents can be popularised easily, they might be the most useful in e-business. Based on Guttman et al's examination (1998), two additional processes can be further classified during the information stage: product brokering and merchant
brokering. With the product-brokering process, a buying agent searches for its preferred product by evaluating a series of available alternatives. With the merchant-brokering process, merchant-specific information helps the agent determine the most suitable supplier.

PersonaLogic (Jonkheer, 1999) and MovieLens (Good et al, 1999) are two examples of buying agents being employed to undertake product brokering. Both systems allow a shopper to specify a list of features about his/her preferred products. A searching agent is expected to encode these hard constraints and to consult service providers for advice. The difference lies in that PersonaLogic employs a CSP (Constraint Satisfaction Problem) technique to perform information retrieval, whereas MovieLens uses a CF (Collaborative Filtering) technique to conduct information filtering. Both methods have proven successful within the website recommendation domain to recommend commodities such as music CDs and books (Shardanand & Maes, 1995). Bargain Finder (Bargain Finder, 2003) and Jango (Chavez et al, 1997) are two examples of agent-mediated merchant brokering, which may take place once a shopper has identified a specific product. The shopper is asked to enter some keywords to search for, then the searching agent delivers a query to a list of merchant sites simultaneously. The difference lies in that Bargain Finder sends all of its requests from a centralised browser, while the Jango system sends its query from a Jango-augmented browser, it therefore resembles a query originating from real customers. A more intelligent comparison-shopping scheme has been suggested by Etzioni and Weld (1994) in their proposed ShopBot system, where the comparison-shopping problem is separated into a learning phase and a searching phase. Artificial intelligence techniques such as heuristic searching, pattern matching, and inductive learning algorithms are employed within the learning phase. The learning result is a
generated vendor description. Once a searching agent receives requests from its user, it goes to each merchant site's searchable index, submitting a form with requested product attributes. After receiving the resulting page, it extracts inner logical data that matches the learned product description, sorting the results, and displaying a summary to its user. The searching agent's learning capability helps it extract product descriptions from a vendor's site accurately. It does not need to be hand-tailored for each online store at run time, and the scalability to different product domains can be achieved easily.

Negotiation is another important stage for various agent actors to conduct business activities. The major task is to determine concrete terms for a transaction and reaching a mutually-acceptable agreement. Some examples can be found within the Kasbah system proposed by Chaves and Maes (1996), the AuctionBot system suggested by Wurman et al. (1998), and the Tete-a-Tete system developed by Maes et al. (1999). All of these allow a consumer-owned buyer agent and a merchant-owned seller agent to negotiate across a range of terms.

In order to help application designers develop a generalised trading-agent with negotiation capability more conveniently, Karacapilidis and Moraitis (2001) introduced a module-based agent construction architecture, where three modules were presented: a communication module, a co-ordination module, and a decision-making module. The communication module is responsible for assisting an agent to interact with its trading partners, therefore, it is the same for both buyer and seller agents. Its major task is receiving, filtering, and transferring various messages. The co-ordination module is designed to deal with co-operation protocols concerning various types of interaction. For a buyer agent, the co-ordination module maintains a purchase database, which keeps records about customer preferences, purchase categories and
the online sellers. For a seller agent, a seller database containing the information about a product's specification, as well as a list of potential customers, is built up accordingly. In the decision-making module, a buyer agent keeps a library of offer synthesis strategies and an offer synthesis graph, which would be activated upon receiving a specific offer from the seller. Concrete synthesis and comparative evaluation processes are performed with the help of an inference mechanism. On the seller agent side, a similar inference mechanism with a library of offer-building strategies is included, with the aim of persuading its potential customers to accept its promoting product. This module-based structure organises various agent negotiation functions into a self-contained entity, which simplifies associated system design when negotiation-oriented implementation is essential.

3.2.2 Using mobile agents for e-market development

The above discussions illustrate how software agents can participate in various e-market-trading activities. Agent-oriented properties focus on communicability, reactivity and adaptability. In this subsection, agent mobility will be considered. From a historical perspective, the feasibility of employing mobile agents for e-market applications can be directly observed from many mobile agent platforms construction. In Chapter 2, three mobile agent platforms were discussed: Telescript, Agent Tcl, and Aglets. Interestingly, all of these presented an associated e-market structure. Figure 3.2 shows an e-shopping centre structure in Telescript:
The original aim of this architecture is to explain a collection of mobile agent-oriented concepts. But it indeed sketches a basic mobile agent-mediated e-market structure. A mobile agent representing its user travels to a remote shopping centre, entering a specific e-shop and meeting with its trading partner(s). Each shopping centre can host many electronic stores, with each store providing a specific service such as directory, ticket, or flower delivery. Telescript technology enables a mobile agent to travel from one shop to another searching for its interested products and preferred partners. The shopping centre organiser (also as an execution environment supporter) is responsible for facilitating agent-to-agent communications as well as agent-based authentication and authorisation. Unfortunately, no more system management facilities are reported in this suggested architecture.

A system manager and a marketplace manager are explicitly introduced into another mobile agent-based e-market framework: TabiCan in Aglets. In this multi-agent system, four kinds of agents are identified: consumer agent, shop agent, marketplace manager, and system manager, as shown in Figure 3.3:
The TabiCan system is supposed to host multiple agents working at the same market. Management work falls on its two managers: a system manager and a marketplace manager. The system manager is used to manage various system resources such as CPU, file system, and displaying facilities, while the marketplace manager is in charge of various internal affairs. The major function is to facilitate the interaction between shop agents and consumer agents, as well as associated message transformation and resources-accessing tasks. A type database is explicitly introduced, which is used for storing all message types for agent communication. Therefore, agents with different internal semantics can communicate freely with each other. In addition, updated behaviours can be injected into those delegating agents at run time as long as they observe the same interaction protocol. Inter-collaboration between the marketplace manager and the system manager is also supported. Another interesting feature of the suggested framework is that a market advertiser agent is explicitly implemented, which is expected to invite more trading agents to join in its virtual market. Unfortunately, this kind of advertising manner only works well when the volume of potential customers is small.
In order to better organise an open and larger e-market where numerous service providers and consumers are available, a yellow-pages service seems to be more feasible. A mobile agent-supported searching structure based on the Agent Tcl technology belongs to this type, as shown in Figure 3.4:

![Diagram of mobile agent-supported searching structure](Gray, 1997)

A consumer despatches a mobile agent into the Internet with a description of its preferred shopping product. The agent contacts the yellow-pages service to identify which vendor agents provide that product, and retrieve their network addresses. In order to obtain further information from each potential vendor, the agent faces two choices: migrating to the vendors' sites by itself, or generating a number of child agents and sending them to each merchant site for individual discussion. Technically, these two solutions follow two different working patterns: master-slave pattern, and supervisor-worker pattern. As Gray (1997) argued, the pattern selection depends on network reliability and vendor availability. Unlike the fully-functional virtual market suggested in TabiCan, the vendor site within this structure is much simpler. The only requirement is providing an easily-accessible interface for all visitors, facilitating their negotiation and finally helping them move to other vendors' sites.
3.2.3 A generic architecture for mobile agent-based e-market

From the above investigation, it can be noted that the basic function of a mobile agent-mediated e-market is the provision of a meeting environment for the visiting agents and facilitating their migration and communication activities. Another observable common feature is that each mobile agent has a home site. The agent starts its journey from its home and visits remote market places sequentially or selectively (which may be maintained by a general system manager, a vendor or even another customer). Based on these observations, a generic architecture for a mobile agent-based e-market can be broadly outlined in Figure 3.5 (Yang & Nguyen, 2003b).

![Figure 3.5: A generic architecture for mobile agent-mediated e-market](image-url)

Four elements are included in this structure: Home, Consumer, Vendor and Public E-Market.

- A **Home** belongs to either a consumer or a vendor. The principal duty of the agent controller on this place is to create a mobile agent, dispatch it to the network, and deal with various returned messages.

- A **Consumer** element represents consumers in the physical world. A consumer agent with knowledge about its purchasing preferences resides at this element, and is ready to receive visiting vendors.
• A *Vendor* element represents vendors. A vendor agent occupying the vendor element must understand its offered services and be ready to deal with incoming consumers. Foreign agent authentication and agent-to-agent communication will be facilitated in both Consumer and Vendor elements.

• A *Public E-Market* is a multi-functional element where two types of agents are involved: agents for trading (e.g., consumer agent, vendor agent, and third party agent), and agents for management (e.g., e-market organiser agent and system maintainer agent). A set of functions to facilitate multi-agent interaction and collaboration, for example, directory service, information database, and negotiation room has to be provided.

Indeed, the directory service is enlightened by the yellow-pages service in Agent Tcl, the information database is much like the type-database in Aglets' TabiCan, and the negotiation room offers the same function as that within Telescript's e-shopping centre. In fact, the above architecture links three different kinds of mobile agent-based e-markets. Mobile agents can work either as a business trader or as a system maintainer within this large and dynamic virtual market network, which is supposed to be platform-independent.

In subsection 2.3.5, the applicability of employing mobile agent technology for network-oriented systems construction has been generally discussed. Most of the identified advantages of mobile agent computing can be directly applied for e-market establishment, because an e-market is simply a special kind of network-supported application, with the aim of serving various market participants (e.g., vendors, clients, and market organisers). A brief summary of the potential and impediment of using mobile agents for e-market development will be discussed below:
• More open market with widespread participation

Although there are many existing techniques available to support current e-market construction such as CGI (Common Gateway Interface), Java Servlets, JSP (Java Server Page) and many others, most of these follow the same computing paradigm -- that is, the client-server model. The client sends a request to the server, while the server tries to process each coming request and send the results back, although these techniques may have differences in request-processing approaches. For example, with CGI, each request is treated by a heavyweight operating system process, whereas with Java Servlets, a new request is handled by a lightweight Java thread for efficiency. From the know-how/resources viewpoint, an e-market established on the server side holds both the know-how and resources. An e-market organiser is at the same time a system maintainer, which must be fully-functional. By contrast, a mobile agent-based e-market is more likely established on a peer-to-peer network, which is to a great extent eliminating the spatial and temporal boundaries between consumers and suppliers. On the one hand, a mobile shopper agent holding self-contained negotiation strategy may move to the vending markets holding resources for information retrieving. On the other hand, a mobile vendor agent with resources may seek those client sites holding user-defined shopping preference for remote marketing. Therefore, a relatively closed market that is traditionally maintained by a single system provider is giving way to a more open and dynamic market network supported by all related business participants, as illustrated in Figure 3.5. Both the service provider and consumer are encouraged to set up their own websites, with the help of the agent platforms' natural support. They may either stay at home waiting for incoming visitors or migrate to a public market to meet their business partners.
• Autonomous performance with reduced human involvement

In most e-market applications with the client-server architecture, only command-based requests can be fulfilled. Therefore, a human shopper wishing to commit business activities at a remote marketplace has to involve in many concrete business processes directly. For example, customers want to buy a book from an online bookstore (e.g., the amazon.com), they have to engage in most business operations in person from logging into the system to checking out, and from product selection to service comparison. Similarly, if human users take part in an online auction market, e.g., the AuctionBot presented by Wurman et al (1998), they have to monitor the auction process step by step in order to make a timely and competitive decision. Mobile agent technology changes this scenario significantly because mobile agents do not need to rely on a communication channel to maintain their remote operations. They can carry a set of pre-defined strategies to a remote market and conduct business transaction by themselves. For example, a mobile shopping agent equipped with shopping strategies can perform product comparison automatically, while a mobile auction agent can even learn from and react to its competitor's auction behaviours intelligently. Human users are only required to send such an agent to the remote market on their behalf and waiting for the final results at home. Direct involvement on most online decision-making processes is thus minimised.

• Tolerance of disconnected operation with unreliable network condition

A public network is often characterised by lower network bandwidth and intermittently reliable connections. Therefore, the stability and reliability of a virtual market established on such an open network would be inevitably affected, no matter what communication protocol is chosen (i.e., http or ftp). With client-server
computing, the client and server communicate over a network through a handshake paradigm: each request/response is a complete round trip on the network. If the communication connection is lost after the client sends out a request, the lost request has to be resent, sometimes even the whole business process has to be restarted, which is especially annoying when a client is conducting online payment operations. Mobile agent technology provides a software solution to this problem before the network conditions can be fully improved in a physical way, because a mobile agent is independent of the network connection once it reaches a remote destination, no further communication connection is needed to support its local business transactions. In fact, an explicit network connection is only needed when the agent decides to leave for its next destination or return home. If such a connection is closed when it tries to move or the home user is temporarily unreachable, the agent can simply wait at the current marketplace until the connection has been re-established or the home user re-connected to the network.

• Improved information exchange and service flow structures

Product and service delivery is an important function for an e-market construction. Service providers are required to forward their commodities to potential customers in a communication-efficient way for profit generation and service enhancement. From a communication pattern viewpoint, two major service delivery patterns are identifiable: pull-based delivery and push-based delivery. For the former, customers access a virtual market from different locations and brought their purchases home. Some commodities such as music CDs and books are suitable for this type of delivery (Shardanand & Maes, 1995). For the latter, service providers push their commodities from the server side to the client side. Such a delivery manner is initially proposed in
response to communication asymmetry exhibited by many applications where communication from the clients to the server is more restricted than the other way around, so it may make more sense to push service from the servers without waiting for the clients to pull it. For example, Netscape sometimes uses this manner to update their online softwares. Interestingly, a mobile agent-mediated e-market application combines these two types of service delivery methods dynamically. On the one side, an online customer may dispatch a shopping agent to a group of remote e-markets for offer collection. On the other side, an online vendor may dispatch a salesman agent to visit its potential customers in sequence to market those goods with only a short shelf-life. An agent-supported online market is thereby formed, with improved information exchange and service flow structures.

Although there is a wealth of evidence to prove the suitability of applying mobile agent technology to e-market construction, one fact must be acknowledged: mobile agent-mediated e-business applications are still at the experimental stage. It seems some practical problems still need addressing before such applications can be commercialised in the near future. At least two major issues can be identified at this stage: open platform standardisation and secure execution environment.

- Open agent platforms standardisation

Agent-supported e-marketplaces might be designed and maintained by different business actors, therefore, the agent and agent platform standardisation task becomes truly important. At first, agent and agent system names need to be standardised, so different e-market participants can recognise each other conveniently over the network. In addition, an agent transformation and authentication infrastructure
requires standardising to facilitate the safe migration of mobile agents among different marketplaces. Within each market, the standardising task would focus on agent management facilities. At first, a market operator is supposed to offer services (such as accessing to the backend databases, and locating each other's trading partners) to its visitors in a standardised style. In addition, a syntax and semantics transformation mechanism needs to be agreed upon in order to help those self-interested agents designed by different groups understand and communicate with each other straightforwardly. Two groups are currently active in the agent standardisation community: FIPA (Foundation for Intelligent Physical Agents) (1997) and MASIF-OMG (Mobile Agent System Interoperability Facility-Object Management Group) (Milojicic et al, 1998). The former seeks to work out standards for heterogeneous agent systems' interoperability, the latter concentrates on the mobile agent field for promoting mobile agent technology being widely deployed for commercial uses.

- Secure agent execution environment

As Corradi et al (1999) realised, the lack of a comprehensive security framework is a major obstacle for the deployment of mobile agents for e-businesses, this problem is still to be adequately overcome. Broadly speaking, security issues for mobile agent-mediated e-market applications can be viewed in two aspects: e-market system protection and mobile agent protection. The major task for the market system protection is the safeguards of various system resources (e.g., the file/database system and display facilities) from being attacked by malicious codes. As presented in subsection 2.3.5, most mobile agent-based hosts are not yet free of security holes, although many system designers try to minimise security threats in various ways. For mobile agent protection, the major function is to prevent the mishandling by dishonest
market operators of data and information to a travelling agent's memory. Unfortunately, this is still an open issue in the mobile agent security area.

3.3 Internet-supported e-business

The Internet not only assists various market participants to buy and sell products via electronic means, but also provides greater potential to many other areas within business operations. As Turban *et al* (2000) observed, knowledge flows, digital product distribution, and quality of service improvement can all benefit from the interactivity of this emerging approach. Thus, in a broader sense e-business can be described as: "…all about time cycle, speed, globalisation, enhanced productivity, reaching new customers and sharing knowledge across institutions for competitive advantage" (Barnes & Hunt, 2001: 45).

3.3.1 Basic e-business models

As discussed in the previous section, a significant potential benefit of employing mobile agents for e-business applications is their enriched information exchange and service flow structures. Therefore, in order to better understand how mobile agent technology can be used to help an online business draw revenue through these improved service flow structures, another specific area has to be discussed -- that is, business models. A rather loose description of business models is "the method of doing business by which a company can sustain itself, it spells out how a company makes money by specifying where it is positioned in the value chain" (Rappa, 2000). A more complete description can be found in Timmers (1998), as follows.
A business model:
- describes an architecture for product, service and information flows, including a
description of the various business actors and their roles;
- identifies the potential benefits for the various business actors;
- specifies the sources of revenue.

In other words, a schematic method of analysing a business model will include not
only its business actors and information flows, but also its benefits and revenue-
generation methods. A collection of basic e-business models has been suggested in
literature (Bambury, 1998; Rappa, 2000; Lawrence et al, 2000). Timmers (1999)
classified some commercial and experimental business models into eleven categories
(as listed in Table 3.1), and most of these are still fully-operational on the network. A
brief summary is given below:

<table>
<thead>
<tr>
<th>Models</th>
<th>Examples (As at 10 December 2003)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-shops</td>
<td>Fleurop at <a href="https://www.fleurop.com">https://www.fleurop.com</a></td>
</tr>
<tr>
<td></td>
<td>Travelocity at <a href="http://www.travelocity.com">http://www.travelocity.com</a></td>
</tr>
<tr>
<td>E-malls</td>
<td>Electronic Mall Bodensee at <a href="http://www.emb.ch">http://www.emb.ch</a></td>
</tr>
<tr>
<td></td>
<td>Techsavvy at <a href="http://www.techsavvy.com">http://www.techsavvy.com</a></td>
</tr>
<tr>
<td></td>
<td>Informa at <a href="http://www.informa.com">http://www.informa.com</a></td>
</tr>
<tr>
<td>Third-party marketplace</td>
<td>Tradezone at <a href="http://www.tradezone.com/trdzone.htm">http://www.tradezone.com/trdzone.htm</a></td>
</tr>
<tr>
<td></td>
<td>LocalEurope at <a href="http://www.localeurope.com">http://www.localeurope.com</a></td>
</tr>
<tr>
<td>Virtual communities</td>
<td>Amazon at <a href="http://www.amazon.com">http://www.amazon.com</a></td>
</tr>
<tr>
<td></td>
<td>Firefly at <a href="http://www.fireflybooks.com">http://www.fireflybooks.com</a></td>
</tr>
<tr>
<td>Collaboration platforms</td>
<td>GENIAL at <a href="http://cic.cstb.fr/ILC/ecproje/genial/home.htm">http://cic.cstb.fr/ILC/ecproje/genial/home.htm</a></td>
</tr>
<tr>
<td></td>
<td>Procat-GEN at <a href="http://www.c-lab.de/gen/projects.html">http://www.c-lab.de/gen/projects.html</a></td>
</tr>
<tr>
<td>Value-chain service providers and Value-</td>
<td>FedEx at <a href="http://www.fedex.com">http://www.fedex.com</a></td>
</tr>
<tr>
<td>chain integrators</td>
<td>UPS at <a href="http://www.ups.com">http://www.ups.com</a></td>
</tr>
<tr>
<td></td>
<td>Verisign at <a href="http://www.verisign.com">http://www.verisign.com</a></td>
</tr>
</tbody>
</table>

Table 3.1: Basic e-business models and corresponding examples
(Based on Timmers, 1999)
**E-shops model**

*Description:* It helps e-shops owners to establish their web presence.

*Benefits for actors:* Owners reduce their advertising and promotion costs. Customers enjoy wider choices, better information and more convenient comparisons.

*Revenue generation:* Increased online sales with reduced marketing costs for the owner.

**E-procurement model**

*Description:* It helps a business setting up online contact with its suppliers.

*Benefits for actors:* The business enjoys broader choices, improved services and reduced procurement costs. Suppliers open up more tendering channels and opportunities. Both parties enjoy time and cost savings and convenience.

*Revenue generation:* Reduced tendering and procurement costs for the online business.

**E-malls model**

*Description:* It helps a group of e-shops to form a shopping mall using a unique brand name with the aid of linking services and other supporting technologies.

*Benefits for actors:* All the e-shops under the same e-mall umbrella enjoy simplicity and convenience for presentation on the Internet. Customers enjoy additional convenience for easy access and payments to different e-shops with enhanced trust and confidence.

*Revenue generation:* Membership fees, transaction fees and advertisement revenue for the e-mall operators.
E-auctions model

Description: It is an electronic implementation of traditional auctions within cyberspace.

Benefits for actors: Based on the technology platform set up by an auction provider, suppliers save time and money on advertising and service promotion, while buyers enjoy global sourcing with increased efficiency.

Revenue generation: Platform providing fees, transaction fees and advertisement revenue for the e-auction provider.

Third-party marketplace model

Description: It provides an additional channel for e-businesses to enhance their online presence with the help of a common interface and a number of integrated services.

Benefits for actors: The shop owner leaves the Internet marketing to a third party and enjoys various value-added services and improved brand awareness.

Revenue generation: Membership fees and service fees for the service provider.

Virtual community model

Description: A group of online members organise into a virtual community with enhanced attractiveness and opportunities.

Benefits for actors: Most values come from the linkage of community members, other services (such as customer profiles and feedback) will be shared by all community members.

Revenue generation: Membership fees and advertisement revenue for the community organiser.
Collaboration platforms model

*Description:* A platform manager provides a collaborative environment for online enterprises to co-operate with their business partners.

*Benefits for actors:* Enterprises benefit from reduced co-operation cost, improved workflow management and enhanced product innovation.

*Revenue generation:* Membership fees and service fees for the platform manager.

Value-chain service providers model and Value-chain integrators model

*Description:* Both models focus on specific services related to the value-chain. The former enhances only a number of specific functions within the value-chain (i.e., online payment or logistics support), whereas the latter seeks to integrate multiple steps of the value-chain into a profitable flow.

*Benefits for actors:* The service providers improve customer services and exploit more business opportunities, while the integrators generate additional values through exploiting new information flows in multiple functions' integration.

*Revenue generation:* Consultancy fees and transaction fees for the service providers and value-chain integrators.

Information brokerage model and Trust providers model

*Description:* An information broker brings a range of new information services to its customers to meet their business requirements in online operations such as information searches, customer profiling, and investment advice. An online trust provider works as a trust third party within cyberspace offering specialised services, such as electronic notaries and certificate processing, to its customers.
Benefits for actors: Customers enjoy integrated services with broader choices, while businesses enjoy increased opportunities with valuable advice.

Revenue generation: Consultancy fees, subscription fees, usage fees and advertisement revenue for the information broker and trust provider.

At least one phenomenon can be noted from the above summary, some models provide only single or simple functions, while others offer complex or integrated services. Therefore, Timmers (1999) classified these models on a graph (as illustrated in Figure 3.6). The dimension of 'degree of innovation' (the x-axis) ranges from the simplest form that provides only an online implementation of the traditional trading, to an increasingly innovative form that cannot be accomplished in a traditional way. The dimension of 'functional integration' (the y-axis) ranges from the model merely offering single functions such as goods selling and marketing, to the model providing multiple functions that seeks to integrate multiple steps of the value-chain.

![Figure 3.6: Classification of eleven basic e-business models](Timmers, 1999)
The e-shops model is at the starting point of both dimensions, while the value-chain integrator model is at the other extreme. The business models in between illustrate different degrees of functional integration and technical innovation. Taking trust services as an example, it is not re-implementing similar functions provided in the physical world, but generating more added values through new technologies, for example, encryption technique and secure communication facilities.

From this figure, two further trends can be identified in the e-business models' development: increased integration of information flows, and more specialised network services (Timmers, 1999). For the former, an online company is required to create additional values through introducing more related services on its own business site. For example, online payment support has been added into many e-shops services. For the latter, an online business is desired to establish its public image as an expert in its specific area, thus increasing customer loyalty and market competitiveness. Compared with the first trend, specialised services are more heavily dependent upon technical support. For example, for an online trust service company, the use of a network-supported encryption technique makes more sense than introducing an additional payment service, in order to provide its customers with trust guarantees.

3.3.2 Transplanted and native e-business models

From the above discussion, it can be seen that different models have different degree of innovation. As Rappa argued (2000), a possible reason for the varying degree of models' innovation is that some e-business models simply extend traditional business models into cyberspace, while others are native to the Internet environment. Rappa therefore presented another taxonomy of e-business models: transplanted models and native models (see Table 3.2 and 3.3).
### Table 3.2: Transplanted e-business models with their sub-categories and examples

(Based on Rappa, 2000)

#### Advertising model

*Description:* It is an extension of the traditional broadcasting model. The broadcaster provides both advertisements and value-added services to the public.

*Benefits for actors:* It helps businesses build their images efficiently and assists customers to be well informed of new products and services.

*Revenue generation:* Advertisement revenue for the broadcaster.

#### Subscription model

*Description:* It is an extension of the traditional subscription model. Products and services will be delivered to potential customers based on their subscription.
Benefits for actors: Businesses can design their customer-oriented promotions more efficiently. Subscribers may receive their desired products or services more conveniently.

Revenue generation: Subscription fees for the service provider.

Utility model

Description: It is a variant form of the traditional subscription model. Products or services will be delivered and paid for based on metering usage.

Benefits for actors: It helps businesses deliver their products or services more flexibly. Customers only need to pay for services as they use.

Revenue generation: Usage fees for the business operators.

Merchant model

Description: It is an extension of the traditional merchant model. Wholesalers or retailers list their goods or services online to attract more potential clients.

Benefits for actors: Merchants improve their customer services with reduced advertising costs. Customers enjoy easy access to their favourite vendors.

Revenue generation: Merchant revenue comes from a high volume of traffic and more potential buyers.

From the above discussion, it is clear that each of these models has a traditional variation in the physical world. For example, the advertising model is based on the traditional broadcasting model, and the utility model takes advantage of the traditional subscription model. They extend or re-implement these traditional models in the virtual world by way of information technology and communication mediums. By
contrast, some other models and their sub-categories have evolved within the Internet environment. In other words, these models exist solely within the Internet context, thus are native to it, see Table 3.3 for detail:

<table>
<thead>
<tr>
<th>Native models</th>
<th>Sub-categories and examples (As at 15 December 2003)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affiliate model</td>
<td>Barnes &amp; Noble at <a href="http://www.barnesandnoble.com">http://www.barnesandnoble.com</a></td>
</tr>
</tbody>
</table>
| Informediary model | Advertising networks: DoubleClick at [http://www.doubleclick.com](http://www.doubleclick.com)  
Audience measurement services: Nielsen/NetRatings at [http://www.netratings.com](http://www.netratings.com)  
Incentive marketing: Coolsavings at [http://www.coolsavings.com](http://www.coolsavings.com)  
Metamediary: Edmunds at [http://www.edmunds.com](http://www.edmunds.com) |
| Brokerage model | Marketplace exchange: Orbitz at [http://www.orbitz.com](http://www.orbitz.com)  
Buy/sell fulfilment: CarsDirect at [http://www.carsdirect.com](http://www.carsdirect.com)  
Demand collection system: Priceline at [http://www.priceline.com](http://www.priceline.com)  
Auction broker: eBay at [http://www.ebay.com](http://www.ebay.com)  
Distributor: Questlink at [http://www.questlink.com](http://www.questlink.com)  
Search agent: mySimon at [http://www.mysimon.com](http://www.mysimon.com)  
Virtual mall: Amazon at [http://www.amazon.com](http://www.amazon.com) |
| Community model | Open source: Red Hat at [http://www.redhat.com](http://www.redhat.com)  
Knowledge networks: AllExperts at [http://www.allexperts.com](http://www.allexperts.com) |
| Manufacturer model | Brand integrated content: Bmwfilms at [http://www.bmwfilms.com](http://www.bmwfilms.com) |

Table 3.3: Native e-business models with their sub-categories and examples  
(Based on Rappa, 2000)  

**Affiliate model**

*Description:* It tries to affiliate website owners with a larger organisation. Potential customers will be redirected to their favourite online stores for shopping.

*Benefits for actors:* Affiliated partner sites receive financial incentives for participation. Customers enjoy broader choices with reduced searching costs.

*Revenue generation:* The affiliates receive revenue if the program generates sales.
**Informediary (Information intermediaries) model**

*Description:* Information intermediaries provide information about consumers' buying behaviours to the sellers, at the same time providing information about producers to the consumers.

*Benefits for actors:* Sellers use such information to adjust their marketing strategies, consumers to decide what to buy and from whom.

*Revenue generation:* Information fees are the major revenue for information intermediaries.

**Brokerage model**

*Description:* Brokers bring sellers and buyers together and facilitate business transactions and other supporting services.

*Benefits for actors:* It saves time and energy for both sellers and buyers in seeking business opportunities and conducting businesses online.

*Revenue generation:* Transaction fees are the major revenue for brokers.

**Community model**

*Description:* It brings together people with similar interests in a specific area.

*Benefits for actors:* People in the same community may share their common-interested information easily and receive professional services directly.

*Revenue generation:* The community organiser collects subscription fees or donations.

**Manufacturer model**

*Description:* It allows manufacturers to set up distribution channels to their customers more rapidly.
Benefits for actors: Manufacturers receive reduced costs for distribution and better understanding for their customers, and customers enjoy improved services.

Revenue generation: Revenue for the manufacturer comes from reduced advertising fees and distribution expenses.

Although many other e-business models or even different taxonomy can be found in literature (Bambury, 1998; Nguyen & Stewart, 2001), none of these is claimed to be exhaustive. One possible reason is that e-business models are not static, but constantly evolving, because emerging technologies often give rise to new e-business models, and creative combinations of various existing models are often proved helpful for online companies to realise their business strategies.

On the one hand, an e-business can combine these basic models into an aggregate form. For example, the advertising model has been integrated into most business websites for market promotion and revenue generation. On the other hand, an e-business can select multiple presence -- that is, adopting multiple basic models at the same time to meet its individual requirements. For example, Amazon.com employs the merchant model in nature while building its virtual community, integrating traditional book distribution services and launching business-to-customer auctions as well. An obvious advantage of multiple presence is improved customer loyalty and better business brand awareness.

A more generalised marketing scheme (namely the 3.5.7 model) has been suggested by Lawrence et al (2000), and is aimed at helping an online business achieve its commercial success via the network through three steps, five strategies and seven tactics. For more information on marketing, please refer to Lawrence et al (2000), which is outside the scope of this thesis.
3.3.3 Business models employing agent technology

Categorising e-business models according to their dependence on the Internet demonstrates the importance of this ever-important platform within the e-business models' development. However, the Internet alone is not sufficient for online businesses to create profits, additional technology is required to help make use of this public platform to achieve their individual strategies. Facing a huge volume of information on the Internet, as Timmers and Gasos (2002) argued, a desired technology is expected to deal with various non-structured information in a business-actor-oriented style. Agent technology is a good candidate. Based on Timmers and Gasos' study (2002), business models can benefit from agent technology in three different senses: individual-agents, collaborative-agents, and society-of-agents.

The *individual-agents* model is very simple and easy to implement because there is no need to worry about interactions with other agent actors. Usually, individual agents have specific objectives, and can be arranged in an *ad hoc* fashion. It is suitable for providing personalised services, for example, an information agent automatically retrieving online information according to its owner's preference, or a marketing agent serving its customers on an individual basis. The *collaborative-agents* model is much more complex, where a group of agents works together towards a common goal. Therefore, there must be a controlling mechanism, which has a global view of the system. First, interaction protocols and information-exchange semantics in the process of value chain integration or supply chain management need to be facilitated. Second, an agent-engineering technique is required for large-scale collaborative agent systems. The primary aim is to co-ordinate multiple agents to work in a productive way to generate a combined system behaviour, expected to be greater than the sum of its individual member's functions. As Timmers and Gasos (2002) argued, many
unaddressed issues remain within this model, such as self-organisation, self-regulation, and social dynamics. Compared with the two models cited above, *society-of-agents* provides the most dynamic working environment for multi-agent co-operation. Agents from different developers or providers can communicate and co-operate with each other to achieve their respective objectives and strategies. It is at the highest level in terms of flexibility and complexity. However, global standards for open agent platforms, system stability, and agents security are major problems that need to be solved before this model can be fully implemented in real applications.

Following the above classification, Timmers and Gasos (2002) identified a number of concrete examples that employed these agent models to help various online businesses enhance their network functions, as listed in Table 3.4:

<table>
<thead>
<tr>
<th>Models</th>
<th>Some examples in European Union's (EU) project (As at 18 December 2003)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-auctions model</td>
<td>CASBA at <a href="http://www.casba-market.org">http://www.casba-market.org</a></td>
</tr>
<tr>
<td>Third party market model</td>
<td>ABROSE at <a href="http://www.fokus.gmd.de/research/cc/ecco/climate/core-projects/abrose.html">http://www.fokus.gmd.de/research/cc/ecco/climate/core-projects/abrose.html</a></td>
</tr>
<tr>
<td>Value-chain integration model</td>
<td>LIAISE at <a href="http://pi.ijs.si/~liaise">http://pi.ijs.si/~liaise</a></td>
</tr>
</tbody>
</table>

Table 3.4: E-business models and examples employing agent technology  
(Based on Timmers & Gasos, 2002)

**E-shops and E-procurement models**

E-shops and e-procurement are two very simple models in term of function integration (see Figure 3.6). These are where individual agents can demonstrate their potential. They may facilitate the enhancement of customer services by providing personal digital assistance to customers for product selection and decision-making, such as in EU’s project COGITO. They may also assist in developing businesses'
promotion strategies, through analysing customer shopping behaviours and constructing one-to-one marketing tools, such as in EU’s other project NECTAR.

**E-auctions model**

Collaborative agents can be employed in this model, whereby selling and buying agents equipped with different bidding strategies may automate transaction processes on behalf of their owners. Administrator agents and auctioneer agents are employed to help the selling agents decide upon the sale price of goods and assist the buying agents with completion of product and service comparisons. In addition to simple price-based auctions, the multi-variable-oriented auction becomes a future direction wherein many other factors (such as delivery time and service contracts) will be considered. EU’s project CASBA is such an example.

**Third party market model**

Collaborative agents working within this model include a supplier agent, a buyer agent, a communication agent and a transaction agent. Supplier agents and buyers agents perform negotiations and transactions based on their specific knowledge and strategy, with the assistance of communication agents. Transaction agents provide services ranging from efficient catalogue searches and secure payment facilities to marketing and risk management, and even volume trading supports. Such agent-mediated business model has been successfully employed within EU's project ABROSE, where a service provider seeks to facilitate a secure and reliable environment for supplier and buyer agents to carry out business activities.
**Value-chain integration model**

Supplier agents, buyer agents, and a mediator agent participate in this model. The mediator agent interacts with buyer agents asking for their specific requirements and designs an initial solution based on each buyer's request. If the solution involves multiple suppliers within the value-chain, the mediator will contact all suppliers to make sure each solution can be satisfied optimally. Multiple steps within the value-chain are integrated, with the aim of changing the information flow and providing value-added services. This model is very useful in industrial automation systems for process integration. EU’s project LIAISE is a good example of this model.

3.3.4 Agent technology for the development of information economy

In the models cited above, the business process is merely viewed from a single-company perspective. In order to consider the whole organisation of the information economy at large, two other relations have to be considered: a multi-enterprise relationship and a market-mediated relationship. Timmers and Gasos (2002) termed these two relationship-patterns as value network and dynamic market. A value network is an enterprise-oriented relationship that focuses on value-chain integration and information flow exploration, whereas dynamic market refers to the enterprise-to-market relationship, which tries to enhance flexibility for an online business to achieve its strategic objectives (see Table 3.5).

<table>
<thead>
<tr>
<th>Focus</th>
<th>Time scales</th>
<th>Mutual commitment</th>
<th>Investment per relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value network</td>
<td>Medium-Long</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Internal relationships among a limited number of companies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic market</td>
<td>Short-Medium</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>External relationships among a very large number of parities</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Characteristics of value networks and dynamic markets
(Timmers & Gasos, 2002)
The internal relationships within the value network involve a limited number of companies. From the value-chain generation viewpoint, a subset of network participants is involved, and these may change frequently. The external relationships within the dynamic markets involve a large number of companies. The added value can be maximised through selecting the most appropriate business partners offering the best opportunity and flexibility. Actually, the value network and dynamic market are not isolated, but related to each other via rearranging the value-chain maximising process. A company seeking maximised value often involves both patterns, which means the combination of two patterns is diversified. The relationship between business models and these two development patterns can be illustrated in Figure 3.7:

![Figure 3.7: Business models with value networks and dynamic markets (Timmers & Gasos, 2002)](image)

An online business may choose different approaches to set up relationships with other market participants. Moving from vertical integration to value network to dynamic market is straightforward, because the group taking part in the value-creation process is naturally becoming larger. Based on different strategic objectives of the business, other forms of integration also make sense. For example, some online businesses
merge into the dynamic markets first through setting up dynamic relations with other
market parties, then internalising business relations in the direction of value network
by means of vertical or horizontal integration. Indeed, it is quite possible for an online
business to be involved in both value networks and dynamic markets at the same time.
That means, for different parts of its business processes, the business collaborates
with its partners in different fashions.

The value-chain integrator model and third party market model belong to the
value-network configuration, that is where agent technology, especially collaborative
agents, could make contributions. Collaborative value-network agents are expected to
detect demand patterns, assist supply-demand negotiations and arrange distribution
schemes. The major challenge of employing agent technique to value network
implementation is the alignment of business processes and business semantics across
the enterprises. And such complexity increases as the value network goes deeper. In
the case of dynamic market pattern, a number of companies often form a certain kind
of short-term relationship in an *ad hoc* style. That is where a society-of-agents could
provide a promising solution, because agents within the society are supposed to be
deployed by different developers and capable of interacting and negotiating within a
dynamic environment. A variety of open issues such as agent interoperability and
common business semantics needs to be addressed before society-of-agents can be
employed freely within real business operations.

### 3.4 Mobile agent-mediated e-business models

After investigating the relations between business models and agent technologies in
general, it is not hard to give mobile agent-mediated e-business models a suitable
position, as a mobile agent is simply a special kind of software agent with mobility. A
natural observation is that when a mobile agent is travelling around the network and carrying out business activities, it is expected to meet many other market participants en route. Therefore, mobile agents are more likely to contribute e-business models in collaborative-agents or society-of-agents fashion. Recalling the generic architecture for mobile agent-mediated e-market suggested in subsection 3.2.2, a mobile trader agent may either drop in a sequence of single-functional places for shopping or marketing, or meet with its business partners in a public marketplace to conduct one-to-one, one-to-many or many-to-many negotiations. Giving consideration to the number of agent actors and their different interaction manners, five associated models can be identified: Single-Buyer/Single-Seller (SBSS) model, Single-Buyer/Multiple-Sellers (SBMS) model, Single-Seller/Multiple-Buyers (SSMB) model, Multiple-Buyers/Multiple-Sellers (MBMS) model, and Mobile Broker (MB) model. A detailed discussion of these will be given in the following subsections (Yang & Nguyen 2003b).

3.4.1 Single-buyer/single-seller model

_Description_: It is the simplest model. All other complex models can be simplified to this form. Seller agents and buyer agents meet at a virtual marketplace for selling and buying on behalf of their users. The one-to-one relationship reflects a large proportion of the negotiation carried out in our daily life.

_Benefits for actors_: Buyers enjoy wider choices and easy access to their favourite vendors, while vendors get more opportunities to meet their potential customers.

_Revenue generation_: A higher volume of traffic and potential service fees for the virtual market operator.
MIT Media Lab's Kasbah market (Chavez & Maes, 1996) is such an example. It provides a virtual environment for visiting agents to sell or buy goods. Buying agents use a set of pre-defined criteria to guide their buying behaviours in the market, such as favourite item, desired price, the highest acceptable price, and the date to buy, while selling agents post their item lists on an advertisement board and wait for potential buyers. The Kasbah market operator is in charge of matching up different interests among the agents and facilitating their negotiations. If the negotiation is successful, the buying and selling agents will inform their owners for finalising the deal. A system prototype has been set up to demonstrate its feasibility. The system model is illustrated in Figure 3.8:

![Figure 3.8: Single-buyer/single-seller model](image)

To some extent, the SBSS model is much like the Third Party Market (TPM) model presented in subsection 3.3.3. However, due to some specific properties of its mobile agent participants, the marketplace presented in this model is expected to perform more agent-platform-oriented functions. In addition to helping multiple trader agents locate each other and facilitating their interaction, the virtual market operator also needs to authenticate each coming agent and assist their migration to their next destinations. Facilitating multi-agent communication is also more complex than that within the TPM model because different business actors (agents) deployed by different developers may use different communication semantics. However, in the Kasbah market, agent-to-agent communication is facilitated only by use of a message board, which is technically simple but insufficient when considering more functional
supports for agent communication within many large-scale applications. Another
evident drawback is that both the selling and buying agents within the Kasbah are
simply one-hop agents, they meet at a single server and cannot migrate from one
market to another. From the agent mobility viewpoint, this limits the system's
flexibility and scalability.

3.4.2 Single-buyer/multiple-sellers model

_Description_: It is a native form for mobile agent-mediated e-business. A human buyer
sets his/her preference for an item he/she wants to buy in advance, and dispatches a
mobile shopping agent to the network. The shopping agent is expected to visit a
number of sellers' sites sequentially or selectively, comparing quotes for the item
according to the human buyer's criteria, finally returning with the best offer.

_Benefits for actors_: Buyers enjoy much flexibility to choose their favourite vendors,
while sellers' marketing costs are reduced. The most attractive force driving a buyer
agent to migrate may be the potentially best offer through online comparisons.

_Revenue generation_: More potential visiting buyers for the seller agent.

As a native form to employ mobile agent technology for e-business activity, the
SBMS model has been successfully employed within many experimental systems
(Mandry et al, 1999; Nguyen et al, 2001). However, mobile agent protection is a well-
recognised problem that hinders this model from being implemented in reality.
Mandry et al (1999) took this model as a basis, and identified eight transaction steps
for a mobile agent to carry out its activities more safely, as shown in Figure 3.9:
Eight steps involved in each transaction are:

Step 1. A buyer initialises a mobile shopping agent from an agent provider;
Step 2. The buyer sends the agent to an agent server;
Step 3. The agent server hides the home address of the agent and dispatches it to the first market server;
Step 4. The agent migrates from one market to another;
Step 5. The agent gets the best offer after information collection and analysis;
Step 6. The agent gets back to that favourite market for negotiation;
Step 7. The agent returns the agent server;
Step 8. The agent server sends it back to its owner.

In the proposed system, an agent server is employed to hide some private information of the agent from the visited hosts, thus the vulnerability of a mobile agent is significantly reduced. They also suggested dispatching a travelling agent with query parts only, and to perform negotiation at the agent server side for stronger agent-
integrity protection. However, from the performance perspective, such an arrangement limits the flexibility of employing mobile agents within a real e-market.

3.4.3 Single-seller/multiple-buyers model

*Description:* It is another model native to mobile agent-mediated e-business. A mobile sales-agent is dispatched from a supplier site to visit its potential buyers, like a travelling salesperson with information about its promoting product and a flexible decision-making power, which is supposed to get maximum gross return to its supplier while considering its customers' purchasing behaviours along the way.

*Benefits for actors:* Sellers gain much flexibility to promote their products remotely, while buyers' searching costs are significantly reduced, because the mobile seller agent extends its marketing platform to all its potential customers over the network. This arrangement is quite suitable for marketing those time-limited goods where the demand curve is not stable, but quick decision-making capability is desirable.

*Revenue generation:* More potential buyers with reduced marketing and advertisement costs for the seller agent.

From the technical viewpoint, there is little difference between mobile seller agents and mobile buyer agents. The basic working pattern behind the scene is that an agent representing its owner starts its journey from its home site to visit a sequence of remote platforms. However, from the business model viewpoint, the pull-based SBMS model is different from the push-based SSMB model. The former is customer-oriented, while the latter is supplier-initialised, which is more suitable for dealing with those time-limited goods with changing values. Dasgupta *et al* (1998) termed a system established on this model as 'a supplier-driven e-market' (see Figure 3.10).
The use of push-based technology for information distribution has been well-studied (Su & Tassiulas, 1997), and the employment of mobile agents for heterogenous information retrieval is not new either (Kotz et al, 1997). The suggested SSMB model seeks to integrate advantages from both technologies under the e-business framework. The data carried by the mobile seller agent is not only the information about the products it is promoting, but also its supplier's sales strategy. With the above system, the optimal quantity for a mobile seller agent selling to a customer en route can be calculated using a maximum-returns algorithm that can be adjusted dynamically based on its current available stock and previous customers' purchasing behaviours.

3.4.4 Multiple-buyer/multiple-sellers model

Description: It is a natural combination of the SBMS model and the SSMB model, and can be viewed as a generalised form of agent-mediated business activities within a virtual market. Multiple buyer agents representing their users travel around the network to search for their preferred products and services, while multiple seller
agents representing their vendors go into the network to promote their commodities. Their meeting places might be a seller's site, a buyer's site, or a public e-market.

**Benefits for actors:** Extended transaction platforms and broader business opportunities for both the buyer and seller agents.

**Revenue generation:** Increased number of traders with potential service fees for the market operator.

The MAGMA (Minnesota AGent Marketplace Architecture) system developed by Tsvetovaty et al (1997) employed this model. Both buyer agents and seller agents within this system are called trader agents. They conduct business activities such as buying, selling, and negotiation with the aid of a common communication facility -- the relay server. All connections and messages among trader agents will be conducted via this server, which is expected to support multi-thread programming and multi-socket connections. It maintains an internal routing table with entries for all trader agents active in its working space, message routing will be carried out on a first-come-first-served manner, as illustrated in Figure 3.11:

![Figure 3.11: Multiple-buyers/multiple-sellers model (Tsvetovaty et al, 1999)](image)

Technically, this relay server sets up an insulation layer between the hosted agents and the underlying physical network. Two primary advantages are therefore derived.
One is added security, because encryption techniques can easily be implemented in this server to check message integrity and data confidentiality. The other is improved scalability, which allows those language-independent agents connected to this system to communicate easily with their partners only if they conform to the same MAGMA messaging semantics. Compared with the TPM model presented in subsection 3.3.3, most business actors participating in this model are mobile agents. Therefore, the suggested e-marketplace has to work as a mobile agent platform as well, which is supposed to perform more mobile agent-specific tasks, such as visiting agent authentication and mobile agent migration supports. In this sense, it is much like the SBSS model; however, it is not a simple accumulation of the SBSS model, but sets a higher demand on communication components. The relay server presented in the MAGMA system provides partial solutions to such requirements.

3.4.5 Mobile broker model

*Description:* It is native to the mobile agent-mediated e-business environment. A mobile broker agent travels continuously between multiple vendors and buyers. From vendors' sites, it gathers the information about their promoted products and provides them with potential buyers' lists. It also collects the demands from buyers' sites, and informs them of the services provided by the vendors. The number of customers involved in this model can be either large or small. Services provided by the broker can range from simplified buy-and-sell fulfilment to more advanced services, with added values.

*Benefits for actors:* Vendors expand business opportunities to more potential customers through a broader marketing channel. Buyers reduce their searching costs to gather desired information about certain products and services.
Revenue generation: Revenues of the mobile broker come from potential service fees, membership fees and transaction fees.

As another important member in the mobile agent family, the mobile broker agent does not belong to any specific vendor or buyer on the network, but works for both of these in a dynamic way. Figure 3.12 illustrates the basic working mechanism of this model:

Figure 3.12: Mobile broker model

Compared with the brokerage model discussed in subsection 3.3.2, the mobile broker model provides similar services: bringing vendors and buyers together, and facilitating their business operations. However, as the name indicates, the traditional brokerage model often takes place within a centralised environment where vendors and buyers get together from different places, a stationary broker helps them locate each other and conduct business activities thereafter. Many online auction applications such as Michigan Internet AuctionBot (Wurman et al, 1998) and Fishmarket (Rodriguez-Aguilar et al, 1998) employed this model. By contrast, the mobile broker model is naturally taking place across the network, where both vendors and buyers can simply stay at home and wait for brokerage services. After all, frequent migration is not always a communication-efficient operation for individual
users. In addition to automating information collection and re-distribution processes, the mobile broker can also perform some other value-added services, such as demand-and-supply matching, multiple service comparison, and distributed constraint satisfaction. In this case, it exactly extends a centralised broker model to a broader networking environment. Unfortunately, this model has not been adopted in any commercialised system. Again, the open agent platform problem and security protection issues have hindered widespread acceptance of this model in reality. Without stronger support for its customers and potential profit generation, the mobile broker agent may simply degrade to a stationary broker agent, providing services locally.

3.5 Chapter summary

This chapter investigates the agent-mediated e-business field. A generic architecture of mobile agent-based e-market is proposed, combining three types of virtual marketplaces: a consumer place established by a consumer, a vendor place developed by a vendor, and a general-purpose marketplace maintained by a third party. The consumer and the vendor elements are simply assisting consumers and vendors to enhance their business presence on the network. The third party marketplace provides more generalised functionality. Participants in this market include not only consumer agents and seller agents that may be directly involved in concrete business activities, but also e-market agents and system agents that are expected to maintain a safe and convenient environment for business transactions. In addition, other value-added services such as directory service, information database, and negotiation rooms are also included for multi-agent interaction and collaboration.
As agents might involve in business operations in three aspects: individual agents, collaborative agents, and society-of-agents, the latter two are regarded as more suitable for mobile agent technology. Five associated models are suggested thereafter: Single-Buyer/Single-Seller model, Single-Buyer/Multiple-Sellers model, Single-Seller/Multiple-Buyers model, Multiple-Buyers/Multiple-Sellers model, and Mobile Broker model. The basic classification criteria are the number of agent participants and their different interaction manners. Both mobile and stationary actors are involved in these models. Potential profits may come from possible membership fees and transaction fees, as well as reduced marketing and advertising costs. From the value-chain development pattern viewpoint, the SBSS and MBMS models belong to the value-network configuration. They aim at value-chain integration through improving internal relationships among a limited number of market participants within a centralised place. The SBMS, SSMB and MB models belong to the dynamic-market pattern. They focus on external relationships among a very large number of participants, with the help of an extended business platform over the network. An online business may choose different models to enhance its internal and external relationships from SSBS to SBMS or from SSMB to MBMS, or other creative combination forms based on its own business logic.

The following chapter will focus upon mobile agent design patterns, which are essential for making use of agent-oriented technical advantage in concrete e-business design, and for benefiting system designers within this specific field.
Chapter 4

Mobile Agent Design Patterns for E-Business Applications

4.1 Chapter introduction

As self-contained entities, mobile agents have been successfully used to enhance an e-business's Internet functions and reduce online customers' searching costs. However, as Deugo and Weiss (1999) noted, most current agent-based systems are constructed in an *ad hoc* manner. Many problem-solving strategies behind each concrete system design are not comprehensively surveyed, leading to common solutions being invented and reinvented by different system developers to address the same problem repeatedly. In addition, a higher-level abstraction above the single agent-level implementation is still lacking, which seriously hinders the agent technology from being employed for developing large-scaled applications with industrial strength. The use of design patterns is regarded as an appropriate solution for this problem, which is always used to document those generalised problem-solving schemes in a specific
area so that they can be re-used in similar scenarios. This chapter seeks to suggest several design patterns specific to the mobile agent-mediated e-business domain. A set of general forces will be identified. Collaboration among various participants in each pattern will be sketched. Advantages and potentials of suggested patterns for agent-based system design will also be discussed.

4.2 Design patterns for software engineering

Concepts of design patterns have their roots in architect Alexander's initial work for urban planning. In one of his earlier publications *The Timeless Way of Building* (Alexander, 1979), Alexander presented 253 patterns covering three different areas: towns, buildings, and construction. The primary goal was recording those expert developers' experience within the construction-planning field to help those new practitioners shape their own living world efficiently and professionally. According to Alexander's (1979: 247-248) description, a design pattern could be characterised by four aspects:

- Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution;
- As an element in the world, each pattern is a relationship between a certain context, a certain system of force that occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves;
- As an element of language, a pattern is an instruction, which shows how the spatial configuration can be used, repeatedly, to resolve the given system of forces, wherever the context makes it relevant;
- The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must create it. It is
both a process and a thing: both a description of a thing which is alive, and a
description of the process which will generate that thing.

Although other definitions for design patterns can easily be found within the literature
(Gamma et al, 1994; Riehel & Zullighoven, 1996), Alexander's description has been
well recognised as a significant basis for pattern study, thanks to its
comprehensiveness and broader applicability. At least, the minimal format including
name, problem, context, and solution has been widely accepted as the 'Alexandrian
Form' for design patterns exploration and documentation (Buschmann et al, 1996).
Nowadays, concepts of design patterns have gone far beyond Alexander's original
work within the context of architecture and urban planning, and successfully
incorporated into many other disciplines such as psychology, linguistics and most
importantly, software engineering.

4.2.1 Fundamental components for software pattern design

Software engineers face similar problems during software construction: formatting
those recurrent solutions made by experienced designers, so that other developers can
use them skilfully to generate good design. Two research groups are well known in
the software pattern design community. One is 'Gang of Four' (GoF) and their seminal
book Design Patterns: Elements of Reusable Object-Oriented Software. (Gamma et
al, 1994), and the other is 'Gang of Five' (GoV) and their book Pattern-Oriented
Software Architecture: A System of Patterns (Buschmann et al, 1996). Both of these
gave their own understanding of design patterns for software engineering.
• From GoF (Gamma et al, 1994: 3-4)
A design pattern names, abstracts and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue. It describes when it applies, whether it can be applied in view of other design constraints, and the consequences and tradeoffs of its use.

GoF viewed design patterns as an innovative mechanism for recording Object Oriented (OO) design experiences, therefore, their definition is evidently expressed in terms of OO concepts. As Appleton (1997) noted, such a description needs to be slightly adjusted by removing the words 'object-oriented' and replacing 'class' with 'component' before being adopted as a general definition for software design patterns.

• From GoV (Buschmann et al, 1996: 8)
A pattern for software architecture describes a particular recurring design problem that arises in a specific design context, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate.

In contrast to GoF’s definition, GoV's description spells out patterns for software development more generally, which is for software architecture patterns, not merely for design patterns (as GoF expressed). Actually, GoF coined the term 'design pattern'
to deal with all patterns related to software structure, subsystem design and
programming implementation. However, GoV tried to categorise these into three
different levels: architectural patterns, design patterns, and idioms, and gave each a
separate definition as below (Buschmann et al, 1996: 12-14):

**Architectural pattern:** An architectural pattern expresses a fundamental
structural organisation schema for software systems. It provides a set of
predefined subsystems, specifies their responsibilities, and includes rules and
guidelines for organising the relationships between them.

**Design pattern:** A design pattern provides a scheme for refining the subsystems
or components of a software system, or the relationships between them. It
describes a commonly-recurring structure of communicating components that
solves a general design problem within a particular context.

**Idiom:** An idiom is a low-level pattern specific to a programming language. An
idiom describes how to implement particular aspects of components or the
relationships between them using the features of the given language.

Such classification is based on different levels of abstraction, from high-level schemes
concerning overall system structure through medium-level strategies on subsystem
component design to low-level techniques for language-specific implementation. This
division brings convenience to system developers because it relates different types of
patterns to concrete software development processes. The overall organisation of a
software system must be planned first, as it will affect the architecture of all its
subsystems. Smaller architectural units will be treated by medium-scale design
patterns. Their behaviours and inter-relationships are the major considerations at this
stage, associated patterns are supposed to be paradigm-independent. Language-
specific techniques are classified into idioms (also called coding patterns), which are similar to the programming patterns in Riehle and Zullighoven's classification (1996). Please refer to Riehle and Zullighoven (1996) for more information.

No matter how patterns are classified in concept, software patterns in general provide a common vocabulary for software developers to document their design experience in an easily-maintained and easily-accessible way. Therefore, in order to record an explored pattern with some basic components, a pattern template needs to be specified explicitly. Both GoF and GoV gave their own format for this purpose, as listed in Table 4.1. The components on the left column are called GoF format, those on the right can be viewed as GoV format.

<table>
<thead>
<tr>
<th>Design pattern components (GoF)</th>
<th>Design pattern components (GoV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern name and classification, Intent, Also known as, Motivation, Applicability, Structure, Participants, Collaborations, Implementation, Sample code, Consequences, Known uses, Related patterns.</td>
<td>Name, Also known as, Example, Context, Problem, Solution, Structure, Dynamics, Implementation, Example resolved, Variants, Known uses, Consequences, See also.</td>
</tr>
</tbody>
</table>

Table 4.1: Fundamental components for software pattern design

Although these two formats use different headings, fundamental elements are quite similar. At first, a design pattern needs a meaningful name for communication. The also known as item is required when a pattern has more than one recognisable name in literature. The intent component in GoF has the same meaning as the problem item in GoV, which is used to describe what problem that a pattern intends to address. A pattern's applicability is used to indicate certain context where the problem may recur and the pattern may apply. The motivation component gives a deeper discussion of a pattern's associated forces to demonstrate how these intricate constraints need to be balanced within a problem-solving context with certain tradeoffs. In the solution component, different presentation techniques will be employed. The static structure is
always presented based on Object Modelling Technique (OMT) (Rumbaugh et al., 1991), the *dynamic* behaviour is illustrated using interaction diagrams (Jacobson et al., 1992), while various *participants* and their *collaborations* have to be detailed in words. Other guidelines for concrete *implementation* including some hints, pitfalls and *variants* also need to be clarified, possibly with the aid of several fragments of *sample code*. Finally, the *consequence* and *example resolved* components illustrate the expected results as well as side effects of the suggested solution, while *known uses* and *related patterns* specify some real examples taken from existing systems in the literature.

4.2.2 Common problems for most agent applications developments

The basic reason agent technology is ideal for building large-scale applications is that it conceptually presents a form of software abstraction. Unfortunately, as Deugo and Weiss (1999) noted, most agent-based systems to date have been developed in a very specific way, which means every problem -- no matter general or special -- has to be dealt with specifically from the beginning. Such developmental approaches limit the potential of employing agent technology to construct those dynamic applications with industrial strength.

Interestingly, Deugo *et al* (1999) found a paragraph of comments in the call for papers on the workshop of *Mobile Agents in the Context of Competition and Cooperation* at the 3rd International Conference on Autonomous Agents'99 as follows: "…gaining more widespread acceptance and recognition as a useful abstraction and technology…" and "…we are uninterested in papers that describe yet another mobile agent system…". Such comments presented a common problem arising within the
current agent-based applications design. In detail, three aspects can be summarised as follows (Kendall et al, 1998):

- Lack of a common vocabulary for application developers to document their design experience, so it is difficult to explore design alternatives with a standardised approach, agents created by different groups may have obviously different properties;
- Lack of an effective method to identify and specify common abstractions above the single-agent level, which adds unnecessary complexities for agent applications development and sometimes leads directly to effort duplication;
- Lack of in-depth discussion of the problem-solving strategy behind each specific application, therefore it is hard to extend those generalised structures or reuse their designing knowledge to build more complex systems, which is believed to be essential for constructing industrial applications.

As noted by Deugo et al (1999), OO technology met similar difficulties when it emerged during the 1980s. Software patterns played an essential role in helping OO system developers to overcome those limitations. The initial 23 patterns suggested by GoF are termed simply as 'elements of reusable object-oriented software' (Gamma et al, 1994). Conceptually, objects and associated design patterns provide a greater level of abstraction than procedural or data abstraction, so they can be directly employed as building blocks for concrete software construction. Such phenomena prompted speculation as to whether OO patterns can also be used for agent system design. An obvious observation is that agent technology shares many general characteristics with the OO technology. An agent itself can be viewed as an active object with a set of agent-specific properties, and many mobile agent platforms, such as Aglets, Voyager
and Grasshopper, are all written in Java, which is also a pure OO language. Applying OO patterns to the agent world will be discussed in the following subsection. Other non-OO architecture styles will be reviewed in the subsection 4.3.3.

4.2.3 Applying object-oriented patterns to the agent world

Kendall et al (1997) studied how OO design patterns can be applied for multi-agent systems development. As they observed, the structure of agent patterns is similar to OO patterns. However, many differences still exist, which broadly come from the differences between agents and objects. At least from the goal-driven viewpoint, an active agent is expected to perform more actor-oriented functions than a passive object that simply executes some encoded behaviours responding to certain triggering events. In order to further explore the applicability of using traditional OO design patterns within the agent world, Kendall et al (1997) presented a generalised framework of a functional agent, as illustrated in Figure 4.1:

![Diagram of Applying OO design patterns within an agent-based framework](image)

Figure 4.1: Applying OO design patterns within an agent-based framework

(Kendall et al, 1997)

Multi-agent interaction and collaboration are two major areas where OO patterns can be employed. The Mediator pattern and Adapter pattern are two meaningful examples
(Do et al, 2003). The Mediator pattern defines a separate object to encapsulate how a group of objects interact with one another. A mediator keeps a communicating interface for all its colleague objects. Each colleague object is required only to contact its mediator when colleague-to-colleague interaction is needed. The Adapter pattern is used to convert a sender's request into another recognisable form for the receiver. Therefore, these two patterns can be used when multiple agents at the same platform with different dialects try to communicate with each other. For remote communication, the Proxy pattern is a useful example (Schelfthout et al, 2002). The proxy class controls access to a real object through a surrogate. A distinct interface helps the surrogate subject hide its real location from all its requestors. Such an arrangement is very useful when multiple agents working at different places try to collaborate with one another but do not know their partners' current addresses, which is a quite common scenario within mobile agent-based applications.

In addition to the above examples of applying OO patterns for agent interaction and collaboration, there are some other suitable patterns for agent reasoning and travelling. The Interpreter pattern and Strategy pattern provide two good examples (Meira et al, 2000; Weiss, 2001). Traditionally, the Interpreter pattern is used to define the interpretation of a language based on a set of grammatical and representational rules. For an agent that has its beliefs and plans, this pattern can be used straightforwardly for the automated reasoning process where corresponding behaviours are carried out when an agent's beliefs are satisfied. The Strategy pattern is used to encapsulate a family of algorithms and make them exchangeable. Therefore, through employing this pattern, an agent's plan in the plan library can be changed independently without the need to modify other associated functional components.
As for mobility, the situation is a little complicated. On the one side, this renders certain OO patterns, such as the Client-Server pattern, no longer suitable to the mobile agent world, because differences between client and server become less meaningful when an agent carrying either its know-how or required resources moves around the network. On the other side, some design patterns originally intended for distributed systems are still well fitted to agent applications such as the Broker pattern (Do et al, 2003). The broker seeks to manage a number of registered services with a set of n-to-n connection facilities. It might be directly used when a mobile agent wishes to keep access to certain network resources remotely during its migration process.

4.3 Agent-oriented and e-business-specific software patterns

The examples cited above are only small samples of the sizeable OO patterns that can be successfully mapped into the agent world. Technically, an active agent is different from a passive object not in its individual functionality, but in its aggregate advantage where all associated properties are integrated into a self-contained entity. In order to further study the applicability of using software patterns within agent-based system design, a group of patterns identified in this specific area will be examined below.

4.3.1 A layered pattern for a functional agent

From the general investigation presented in subsection 2.2.1, it is not difficult to see that a software agent can be characterised with a set of mandatory and secondary properties. Many of its secondary properties (such as communicability, adaptability, and mobility) are not isolated, but correlated to each other. Based on these observations, Kendall et al (1998) suggested using a layered pattern to describe a
functional agent, with each layer handling one specific aspect of agency, as shown in Figure 4.2:

![Figure 4.2: A layered pattern for a functional agent](Adapted from Kendall et al, 1998)

This layered pattern can be read in two directions: clockwise and anti-clockwise. From the clockwise direction, an agent receives a remote message from the mobility layer. The translation layer is responsible for reformatting the message into the semantics that the agent can understand. The collaboration layer is used to decide whether the incoming request needs to be further processed. If necessary, the actions layer then takes over the task. Automated reasoning will happen within the reasoning layer, and various updates will finally take effect within the agent's belief and sensory layers. From the anti-clockwise round, the sensory layer is in charge of sensing changes in the agent's working environment. Based on the sensor's input, the agent's beliefs will be updated. The reasoning layer decides how to process the beliefs and what to perform next. The actions layer is used to carry out concrete plans. If collaboration is required, the collaboration layer will be triggered. An outgoing message will be generated at the translation layer, and if necessary, a direct migration decision will be finalised at the mobility layer.

Between every two neighbouring layers in this pattern, as Kendall et al (1998) claimed, there is a kind of dependency-relationship. That is, each layer is dependent
only on its two neighbours. Concrete patterns within each layer are supposed to work for the same purpose, as summarised in Table 4.2:

<table>
<thead>
<tr>
<th>Seven specific layers</th>
<th>Concrete patterns within each layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>The sensory, beliefs and reasoning layers</td>
<td>Deliberative agent, reactive agent, opportunistic agent, interface agent.</td>
</tr>
<tr>
<td>The action layer</td>
<td>Intention, plan as command, plan and intention factory, prioritizer, future observer, adaptable active object.</td>
</tr>
<tr>
<td>The collaboration layer</td>
<td>Centralised collaboration and decentralised collaboration.</td>
</tr>
<tr>
<td>The mobility layer</td>
<td>The clone, the remote configurator, the broker, the migration thread factory.</td>
</tr>
</tbody>
</table>

Table 4.2: Concrete patterns within a layered-like functional agent

Patterns for the sensory, beliefs and reasoning layers are grouped together because they are all broadly used for determining how to respond to an outside stimulus or an incoming request. The *Deliberative Agent* pattern is applied when a symbolic representation of the environment can be identified so that the agent can select a knowledge-based solution to address associated problems. The *Reactive Agent* pattern is employed when there is no symbolic representation available for a given situation. Thus, a reactive agent has to use a set of situated rules to guide its stimulus-response behaviour. If there is a specified symbolic representation for the problem but not a knowledge-based solution available, the *Opportunistic Agent* pattern will be applied, where constraint propagation and satisfaction techniques are used to find the solution. Finally, the *Interface Agent* pattern is employed to meet each user's specific needs through monitoring their actions and identifying their user models over time.

The action layer is aimed at scheduling actions selected by the reasoning layer. The *Intention* pattern is used to instantiate a plan and execute it in a separate thread of control. The *Plan as Command* pattern tries to encapsulate a plan as a self-contained object with higher-level operations, while the *Plan and Intention Factory* pattern tries
to instantiate different plans at run time. If there are many forms of behaviours need to
be attached to an intention dynamically, the Prioritizer pattern will be employed for
priority handling. The Future Observer pattern can be used to address the
asynchronous communication problem between an intention thread and a concurrent
server thread. In order to manage an agent's actions that may occur within different
environments, the Adaptable Active Object pattern becomes useful by separating an
agent's method execution operation from its invocation functions.

The collaboration layer will be invoked when the agent needs external co-
operation. Two types of collaboration are identified naturally: centralised style and
decentralised style. The Facilitator pattern is developed for centralised collaboration,
providing a common gateway for agents' co-operation taking place within the same
society. The Proxy and Emergent Society patterns are suggested for decentralised
collaboration. The former facilitates such co-operation through providing a uniformed
interface to all of its delegate subjects. The latter treats collaboration among reactive
agents as a series of linked stimulus-response behaviours.

Within the mobility layer, the Clone pattern seeks to generate a cloned agent at a
remote machine and provide it with sensors and effectors for the new environment.
The Remote Configurator pattern is used to help a migrated agent complete its
configuration at the destination machine without changing its base classes. In addition
to the Broker pattern (which actually provides a virtual form of migration for an agent
to get remote resources), a Migration Thread Factory pattern is also suggested to help
an agent choose virtual or real migration at run time through a thread manager and a
handler creator.

In summary, the layered pattern organises an agent's various identified properties
in a logical way. An agent system developer is expected to re-use it skilfully for
functional agent construction. However, it does not comprehensively consider an agent's working environment, which is also essential for an agent's computational advantage fulfilment.

4.3.2 An architectural pattern for an agent system

Silva and Delgado (1998) suggested an architectural pattern for an agent system, which considered more general features in distributed applications, such as agent user, agent execution place, and even security management, as depicted in Figure 4.3. The primary aim of this pattern, as Silva and Delgado (1998) argued, is to help software designers construct an Agent-Based Application (ABA) soundly, which is supposed to be autonomous, heterogeneous, open, dynamic, robust and secure. Seven participants are included in this pattern: User, Client, Agent View, Agent, Concrete Agent, Security Manager, and Execution Place.

![Figure 4.3: Collaboration diagram for an agent system pattern](Silva & Delgado, 1998)

The User and Client components have different meanings. The former represents an agent's owner to facilitate an agent's identification and authentication, while the latter represents any other objects that have access to the agent instance. The Agent View component adopts a similar concept as that of the proxy pattern, which supports
transparent control to the agent, thus avoiding additional operations on remote classes management. The Agent component is described as visible and extensible. Visible means that several services such as persistence, communication, mobility, and naming service are all available to the outside objects. Extensible means that its 'callback' and 'helper' methods can be customised and further modified by concrete agent classes. The Concrete Agent component is therefore used to implement the agent's specific functionality. Finally, the Execution Place component specifies a suitable working environment for the agent's computation. The Security Manager is used to specify the security policies on the execution place. Clearly, different agent platforms might define different security policies.

As Silva and Delgado (1998) claimed, several existing agent platforms have employed this pattern in their systems' design, e.g., AgentSpace (Silva et al, 1998), Aglets (Lange & Oshima, 1998), and Telescript (White, 1994a). Although minor differences exist in their individual implementation, the suggested pattern does provide a broader abstraction for agent-based applications' development. It clearly specifies the relationship among agent, agent execution environment and client, and encapsulates more agent-oriented functions as well as object-oriented abstractions. The layered agent pattern presented previously is not in conflict with this pattern. Instead, it can be incorporated into the system pattern through the concrete agent component: the agent system pattern setting up a general structure while the layered pattern builds a multi-functional agent. Unfortunately, none of these patterns incorporates agent properties with any domain knowledge, which is expected to have its own abstract representation within a pattern's problem/solution space.
4.3.3 Some specific patterns for Internet shopping

As e-business turns out to be a prominent topic in recent years, e-business patterns also become more and more desirable (Fernandez et al, 2001). As Manolescu and Kunzle (2001) analysed, most currently available technologies such as CGI scripts, Java Server Pages, Servlets and their simple combinations cannot fully satisfy the e-business developers' higher-level requirements, where more expressive abstractions in structure are needed to support their system management and service integration. Design patterns may contribute to this purpose properly because they try to incorporate various problem-solving strategies and related domain knowledge into a set of well-designed components. Application developers are encouraged to reuse these components to achieve their business successes through a combination of their own business logic and the best presentation technology available on the network.

Taking Internet shopping as an example, design patterns have been employed within a variety of business processes e.g., searching, shopping, and management. Table 4.3 lists a group of patterns for Internet shopping collected from the literature:

<table>
<thead>
<tr>
<th>Patterns for Internet shopping</th>
<th>Selected patterns in each category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searching process</td>
<td>News, push communication, advising, opportunistic linking.</td>
</tr>
<tr>
<td>Shopping process</td>
<td>Shopping process, dynamic configuration, explicit process, easy undo.</td>
</tr>
<tr>
<td>Management process</td>
<td>Catalog, stock manager, a set of order/shipment patterns.</td>
</tr>
</tbody>
</table>

Table 4.3: Some selected patterns for Internet shopping

The News pattern (Lyardet & Rossi, 1998) tries to keep visitors informed of the latest information available on the website through fast visual feedback. Both the Push Communication pattern and the Advising pattern (Rossi et al, 2000) are used to simplify a user's searching process. The former maintains a user's profile for
advertising new products via push-based communication, the latter works as a shopping assistant to provide useful advice when a user visits a virtual store. The *Opportunistic Linking* pattern (Rossi *et al.*, 2000) is used to help a user navigate a website comfortably through facilitating more useful links and adaptable interfaces. In addition, a set of searching and selecting patterns has been given in Lyardet *et al.* (1999), which includes *Selectable Search Space*, *Selectable Search Engine*, *Selectable Keywords*, *Structured Answer*, and *Simple Search Interface* patterns. The primary aim of this pattern group is to improve a website's searching capability with the aid of a more powerful search engine. Enhanced reporting structure and reduced navigating costs are two major advantages.

A collection of patterns used for the shopping process can also be found in literature. The *Shopping Process* pattern (Fernandez *et al.*, 2001) introduces a set of well-defined shopping steps by way of a shopping cart component to record a user's shopping activity. The *Dynamic Configuration* pattern suggested by Lyardet and Rossi (1998) provides a similar function, but adding a validation process when the users put shopping items into their baskets. The *Explicit Process* pattern is employed to help a user understand the checkout and registering procedure through a series of perceivable feedbacks (Rossi *et al.*, 2000). And the *Easy Undo* pattern (Rossi *et al.*, 2000) provides a convenient undo function to improve a system's fault tolerance capability.

Finally, there are other patterns used for virtual store management. The *Catalogue* pattern (Fernandez *et al.*, 2001) is used to present a large variety of products information within a well-organised structure to improve its attractiveness. The *Stock Manager* pattern (Fernandez, 2000) places more attention on inventory management, seeking to keep track of the quantity of stocked items in different
locations in an easily-accessible manner. A set of *Order/Shipping* patterns describing the basic aspects related to ordering, shipping, and delivery services has also been suggested. For more information, please refer to Fernandez et al (2000).

Although the above patterns are presented by different research groups, it can be seen that many aspects of an Internet business's operation have been covered: helping users to find what they want, easing their shopping activities, and assisting business operators to maintain their websites. The system designers are expected to select them skilfully to implement their business presence over the network.

4.3.4 A family of patterns for agent-mediated e-business

Applying agent-oriented patterns to e-business system design is quite natural. As Weiss (2001) observed, when a virtual business deals with large volumes of data, resources or knowledge distributed over the network, the whole system might be structured as a society of autonomous entities that collaborate with one another. Therefore, based on various business roles that an agent might play, Weiss (2001) outlined a family of patterns specific to the agent-mediated e-business, see Figure 4.4:

![Figure 4.4: A family of patterns for agent-based e-business design](Weiss, 2001)
The *Agent Society* pattern is at the top level, which tries to address the problem of how to organise a group of collaborating agents within a concrete e-business system design. The *Agent as Delegate* pattern works for a single user, which is used to assign time-consuming tasks to a software assistant to reduce search costs and balance overloaded information. The *Agent as Mediator* pattern works for a user group, which is used to facilitate agent-mediated business transactions. The *Common Vocabulary* pattern defines a common exchange format in which multiple agent-members can communicate with each other. Collections of lower-level patterns are also presented: the *User Agent* pattern deals with interfaces between the user and the system to ensure quality of service, the *Buyer Agent* pattern defines an agent to work for a customer, while the *Seller Agent* pattern defines an agent for a vendor. The *User Profiling* pattern is used to capture a user's buying preference automatically, and the *Cataloguing* pattern tries to provide personalised services to an individual client. Finally, the *Notification* pattern seeks to keep a customer informed of the latest changes to certain events through remote information evaluation.

These patterns seek to capture some real problems within the agent-based e-business domain and incorporate a variety of well-formulated solutions. However, all of these patterns focus on static agents only, neither mobile agent-oriented properties nor mobile agent-based system management have been taken into account, which is regarded as essential for applying agent-based technology to Internet-supported systems. In fact, many other presented patterns in Lyardet *et al* (1999), Rossi *et al* (2000) and Fernandez *et al* (2001) have similar drawbacks.
4.4 Design patterns for mobile agent-mediated e-business

In order to overcome the limitations cited above, this section tries to suggest a set of patterns specific to the mobile agent-mediated e-business domain. Most mobile agent-related characteristics (e.g., mobility, communicability and adaptability) will be considered. Most business roles involved in e-business design will be described: a mobile shopping agent and its homeowner, a stationary market-organiser agent and its travelling visitors, etc. A collection of general forces regarded as the major factors to push and pull a complex problem to different solutions will be investigated.

4.4.1 A refined pattern template and a set of general forces

The pattern template provides a standardised way to document an explored pattern. However, there is currently no uniform template agreed upon within the pattern design society. Different templates with different components can be found in literature, such as the formats presented by GoF (1994) and GoV (1996), and the re-formatted templates suggested by Lange and Oshima (1998). This subsection employs a combined pattern template as illustrated in Table 4.4 (Yang & Nguyen, 2004b).

<table>
<thead>
<tr>
<th>Components</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intent</td>
<td>A statement to explain what problem is solved</td>
</tr>
<tr>
<td>Applicability</td>
<td>A description of where the pattern may apply</td>
</tr>
<tr>
<td>Forces</td>
<td>A group of factors that need to be balanced in the pattern design</td>
</tr>
<tr>
<td>Participants</td>
<td>The classes taking part in the explored pattern</td>
</tr>
<tr>
<td>Collaboration</td>
<td>The collaboration relationship among all participants in the pattern</td>
</tr>
<tr>
<td>Consequences</td>
<td>The expected results as well as side effects of the suggested pattern</td>
</tr>
<tr>
<td>Implementation</td>
<td>Pieces of source code to illustrate how to implement the pattern</td>
</tr>
<tr>
<td>Related patterns</td>
<td>References to patterns that share with the similar context</td>
</tr>
</tbody>
</table>

Table 4.4: A template for mobile agent-mediated e-business patterns
This template seeks to include both essential components in general software design and optional components suitable for mobile agent-based e-business applications development. A 'Forces' component is included in the above template, which is expected to be domain-dependent. As Coplien (2003) described, forces reveal the complexity of a problem, and are supposed to impact upon proposed patterns in either a positive or a negative way. A good pattern design needs to encapsulate the forces as much as possible, and describes trade-offs that must be made as well. For this reason, some general forces within the related study domain must be investigated before concrete design patterns can be explored.

In this subsection, a number of general forces specific to the mobile agent-mediated e-business pattern design will be identified, where some come from mobile agent specification and others from e-business system maintenance (Yang & Nguyen, 2004b).

- Market openness and service quality
The Internet makes artificial boundaries between suppliers and buyers in the physical world becoming indistinct. Representatives gathering from different locations can carry out business activities locally. However, with more and more agents meeting at a specific market, the market management problem becomes serious because it may lead directly to the quality of services being affected. On the one hand, it is desirable for a market operator to provide an open business environment for all visiting agents to search for their preferred products and services. On the other hand, it is necessary to help agents with similar interests locate each other's business partners for group trading and direct negotiation. In other words, the market operator is supposed to serve both general agent visitors and some specific agent trading groups.
• Free mobility and self-control capability

One of the significant characteristics of a mobile agent is its capability of visiting a set of remote hosts according to a pre-defined itinerary. Therefore, it is possible, and desirable, to objectify a mobile agent's itinerary module to facilitate reuse. In a mobile agent-based e-business application, the major work of these shopping delegates is navigating among a number of remote vendors' sites and collecting product information for their owners. Therefore, a mobile agent's pre-defined itinerary is always affected by the agent's business activity along the way. Incorporating a mobile agent's self-navigation capability with its business environment en route is a general solution for the mobile agent-based trading system design. The primary aim of such an arrangement is to improve a mobile agent's adaptability within complicated market conditions while maintaining the reusability of its modularised navigation plan.

• Remote delegation and effective management

Remote delegation is an important application of agent technology. A mobile agent may represent either a buyer or a vendor to travel around the network performing a pre-defined task. Potential advantages of this application can be summarised as lightweight travelling, goal-oriented driven and disconnected operation. Nevertheless, there is still considerable reluctance to adopt mobile agents for such a purpose. One major reason has been the security issues involved in allowing the relevant agent to visit remote hosts. A dishonest agent host may analyse a visiting agent's decision-making policy illegally and/or alter its carried information maliciously. Mobile agent protection against malicious hosts remains an open issue within the agent research community. Before this problem can be fully addressed, an agent owner has to enhance its control over its child agent's behaviours remotely. One possible solution is
to hide some vulnerable information from remote hosts and withdraw some decision-making functions from the travelling agent site to its home site. Admittedly, performance degradation is the major drawback of this arrangement.

- **Group delegation and inter-group collaboration**

There are two ways for a user to delegate a task to its agents: single delegation and group delegation. For the former, a single agent represents its user, travelling among a set of remote markets in sequence. Technically, it is a kind of sequential computing. For the latter, a group of agents is dispatched to a collection of destination-markets. They are expected to stay there and carry out pre-assigned tasks. Such a practice can be viewed as a kind of parallel computing. However, in a mobile agent-based e-business environment there are too many online markets available, and dispatching a single agent to visit all of them is extremely time-consuming, but generating the same number of mobile agents for all the markets is not efficient either. A common solution is to dynamically combine these two forms of delegation, where a user sends out a limited number of agents and lets them visit available online markets in parallel. Indeed, such an agent group forms an agent family that belongs to the same user and works on the same task. In this case, inter-group collaboration seems quite desirable. Taking the itinerary as an example, the inter-group collaboration needs to ensure the same destination is visited by one of the group members once only. Itinerary balancing is also necessary in order to improve the overall performance of the group.

Items presented above are only part of the observable forces within the mobile agent-based e-business domain. They demonstrate the complexity of applying mobile agent technology to the e-business field. A successful pattern design is expected to address
these problems in a considerate way, aiming at resolving these forces with as much flexibility as possible. In the following subsections, four associated patterns will be suggested: a market-organiser pattern, two comparison-shopping patterns, and an itinerary-balancing pattern. All of these will be described and analysed based on the pattern template listed in Table 4.4.

4.4.2 Market-organiser pattern

Intent
The market-organiser pattern provides a way to help seller agents and buyer agents with similar interests more conveniently establish local contacts on a specific marketplace.

Applicability
This pattern can be employed in the following cases:

- When a market operator wants to provide a uniform interface to all visiting customers to simplify their access to the available services;
- When a market operator agent wishes to manage its trading environment in an easily-controllable and easily-extensible manner;
- When a market operator tries to organise all visiting agents based on their interests in order to improve the efficiency of their local communication.

Forces

- Market openness and service quality

Participants

Figure 4.5 shows the structural relationships among the participants of this pattern.
- **marketGuide**: This object works as a market-guide providing a uniform interface to all visiting agents and presenting a general directory of all available trading groups. It is also responsible for storing the information associated with all active agents within a trading group, and keeping the records up-to-date.

- **visitingTrader**: This is a concrete mobile agent class, representing either a buyer or a seller visiting a specific market and expecting to meet business partners there.

- **Lookup**: This is a mobility listener used to equip each mobile trading agent for responding to various mobility events.

![Figure 4.5: Participants within the market-organiser pattern](image)

**Collaboration**

Collaboration among the participants within this pattern is shown in Figure 4.6:

![Figure 4.6: Collaboration within the market-organiser pattern](image)
Whenever a mobile seller or buyer agent arrives at a new marketplace, an 'onArrival' method in the 'Lookup' listener is triggered.

A 'marketGuide' object handles a 'Search' message from the newcomer, and searches for the backend database. After locating a proper trading group, it sends a reply with all available partners within that group.

A 'visitingTrader' object enters that group and begins its business activities with retrieved partners' proxies.

Whenever a mobile trader agent wishes to leave the market, it triggers an 'onDispatching' method in the 'Lookup' listener.

The 'tradingGroup' object erases the leaving agent's identity from the corresponding group after receiving an 'Exit' message, the corresponding database is then refreshed.

**Consequences**

The market-organiser pattern has the following benefits and drawbacks:

- This pattern can be employed at the market manager side to improve its service quality. The functionality of a traditional market guide is extended to help multiple seller and buyer agents within the same market search for each other's trading partner. This kind of group-trading environment is more like a chat-room with a specific topic. Participants within each room share similar interests, and their communication will not be subject to interference from outside conditions. The market operator takes over all the dirty work, such as information filtering and partners locating.

- The suggested pattern also has drawbacks. At first, the identifier of each trading group is difficult to standardise, especially for those temporary groups created at
run time. A menu-like multilevel location technique might be a possible solution, however, which in turn adds to the complexity of the market organiser's work. In addition, it is not easy to help an agent with more than one interest look for all of its trading partners at the same time. Proxy cloning is possible, but management work for the duplicated proxy becomes more complicated for the market operator.

**Implementation**

There are four basic classes within the market-organiser pattern. A detailed implementation can be found in Appendix A.1.

- The 'Lookup' class extends 'MobilityAdapter', which takes an agent's trading commodity vector from its constructor. Two specific methods -- 'onArrival' and 'onDispatching' -- are overridden. The former is triggered when a mobile agent arrives at a new market and starts searching for its partner. The latter is called when the agent tries to exit the trading group before leaving. The whole class is supposed to be plugged into an arbitrary trader agent in a ready-to-use fashion.

- The 'marketGuide' class registers itself with a well-known name on the network. It maintains a sellers-database and a buyers-database for each trading group and making them accessible to all of its visitors. For the 'Search' message from an incoming agent, it will send back a proxy list for sellers or buyers that are currently active within the group. For the 'Exit' message from a leaving agent, the backend databases will be refreshed accordingly.

- The 'visitingTrader' class is a base class for the mobile trading agent. It gets an instance of the 'Lookup' listener upon being created. When 'running' within a remote marketplace, the agent tries to retrieve its partner's proxies within a
specific group. When completing its given task and preparing for leaving, the agent de-registers from the market organiser's database.

**Related patterns**

This pattern is inspired by the *Meeting* pattern defined by Aridor and Lange (1998), where agents could establish local interactions within a specific context. However, the primary aim of the *Meeting* pattern is to synchronise behaviours of the agents coming from different origins, so the destination ID and partner ID are two principal arguments to help them locate each other. The market-organiser pattern does not assume a visiting agent belongs to any specific group in advance, but provides a uniform interface for all visiting agents. An incoming trader agent (seller or buyer) may choose a proper group to join in, and local communication will be easily established. To some extent, the 'Lookup' listener and the 'marketGuide' object in the market-organiser pattern encapsulate more basic functionality unique to a real virtual market than the meeting manager object suggested in the *Meeting* pattern. As a known use, this pattern has been successfully employed within a mobile agent-mediated e-shopping system suggested by White (1994b). A mobile shopping agent visits different shops within a large e-market to look for its preferred products. Each shop is designed for a specific commodity, all participants within the same shop can exchange information straightforwardly. In the proposed market-organiser pattern, detailed inter-group communication mechanism is ignored, while the trading group organisation functionality is clarified.
4.4.3 Agent-side comparison-shopping pattern

Intent

The agent-side comparison-shopping pattern defines a mobile agent-mediated e-shopping scheme where a mobile shopping agent is equipped with an objectified itinerary and a simplified self-navigation module.

Applicability

This pattern can be employed in the following cases:

- When an agent owner wishes to objectify a mobile agent's itinerary and define the agent travelling plan in an easily-manageable fashion;
- When an agent developer wants to enhance a travelling agent's self-navigation capability after it is dispatched into the network and carries out tasks remotely;
- When a system designer tries to separate an agent's travel plan from other task-delegation functions in order to improve the maintainability of the whole routing module.

Forces

- Free mobility and self-control capability

Participants

Figure 4.7 shows the structural relationships between the participants of this pattern.

- shoppingAgent: This is a concrete mobile agent class, which represents a shopping agent travelling around the Internet to collect relative information for its owner. It can decide whether to continue its journey based on the collected information and comparison result en route.
• **Routing**: This is an abstract class that provides a common interface for all travelling shopping agents with a set of abstract methods defining various routing functions.

• **myRouting**: This class implements the abstract methods defined in the abstract 'Routing' class, comparison-based routing strategies will be specified in order to meet diversified shopping requirements from the agent owner.

![Diagram of Participants within the agent-side comparison-shopping pattern](image)

**Figure 4.7: Participants within the agent-side comparison-shopping pattern**

**Collaboration**

Collaboration among the participants within this pattern is shown in Figure 4.8:

![Diagram of Collaboration within the agent-side comparison-shopping pattern](image)

**Figure 4.8: Collaboration within the agent-side comparison-shopping pattern**
• After its creation, a shopping agent invokes a 'goWhere' method in its routing class, which will 'move' the agent to its first destination -- remote market 'A'.

• When the shopping agent arrives at the target market, its 'onArrival' method is triggered, the 'compare' method in its routing module is then invoked, which will take a shopping item to query the local market operator for comparison purposes.

• If the collected offer is not satisfactory, a 'goWhere' method will be called again, which will retrieve the next available destination in its routing vector, and the 'move' method will help the agent migrate to the second market 'B'.

• The same scenario will occur within the market 'B'. If the comparison result at this market satisfies the criteria set by the agent owner, the agent will stop its journey and go back home immediately with the aid of the same routing methods 'goWhere' and 'move'.

Consequences

This pattern has the following benefits and drawbacks:

• The agent-side comparison-shopping pattern tries to capture the basic functionality of a mobile shopping agent within an Internet-based e-business environment. Variations of this pattern can be objectified and plugged into a travelling agent in a ready-to-use style. Based on the same destination set, an agent designer may specify different routing strategies to improve its flexibility. In addition, it demonstrates certain feasibility to link an agent's travelling plan with its delegation tasks. Thus an agent can adjust its routing behaviour based on its observations along the way.

• The suggested pattern has an obvious drawback. The mobile agent is equipped with both the itinerary information and the self-routing strategy in its memory.
Such an arrangement brings serious security risks. Because an agent is nearly subordinate to its working platform, a visited host might dishonestly prevent the agent from visiting its competitor's host, or maliciously alter associated information to cheat the agent owner. A possible solution might be using a server-side comparison-shopping pattern (discussed in the following subsection), which withdraws some decision-making functions from the agent-side to its owner's side.

**Implementation**

There are four basic classes in the agent-side comparison-shopping pattern. A detailed implementation can be found in Appendix A.2.

- The 'shoppingAgent' class is a simplified mobile agent class extending 'Aglet'. It initialises a routing module with its proxy and instantiates a mobility listener in its constructor. A 'compare' function is implemented in its 'onArrival' method.

- The abstract 'Routing' class implements 'Serializable' because it is supposed to be transferred with the shopping agent together. It has a constructor to set the information of the mobile agent's routing vector. A 'move' method will be implemented in this class, which will be used to dispatch the agent to its next destination or dispose of it remotely. A couple of abstract methods 'compare' and 'goWhere' will be defined explicitly in this class.

- The 'myRouting' class extends the generic 'Routing' class. It takes an agent's routing vector and a shopping vector as arguments in its constructor. Two abstract methods defined in the abstract 'Routing' class will be implemented here. A 'compare' method takes a shopping-item string to query a local market operator and perform the concrete comparison algorithm. A 'goWhere' method is used to
retrieve the next available host-address. Both methods are ready to be overridden by different system developers to enrich their individual strategies.

- The 'Creator' class is not shown in the collaboration diagram, but is needed to combine all the above elements together. Two vectors are initialised in this class: a routing vector includes a list of proposed destinations and a home address, while a shopping vector includes several shopping-related items. Both vectors will be handed to the shopping agent after its creation.

Related patterns
This pattern is inspired by the Itinerary pattern defined by Lange and Oshima (1998), where an objectified agent itinerary is used to help a mobile agent travelling between a set of destinations. However, as a static itinerary module without any routing functionality is seldom useful, extending this general pattern with certain self-navigation characteristics seems essential to enhance its operability. In the mobile agent-mediated e-business system design, a comparison-shopping function with self-routing capability can be found in many existing systems such as in Dasgupta et al (1999), Mandry et al (1999) and Nguyen et al (2001). With these applications, a mobile agent travels around the Internet collecting associated information about its preferred products, then continuing its journey to the next available marketplace. It has been viewed as a classic prototype for demonstrating the applicability of employing mobile agent technology into the e-business field. The suggested agent-side comparison-shopping pattern seeks to capture a common solution to this recurrent scenario. A comparison-shopping function and a conditioned-routing function are designed separately but retain some linkage, which improves both feasibility and reusability.
4.4.4 Server-side comparison-shopping pattern

**Intent**

The server-side comparison-shopping pattern defines another mobile agent-shopping scheme where a home shopper assigns the information-collection task to an itinerant agent but controls its travelling plan from home.

**Applicability**

This pattern can be employed in the following cases:

- When a home shopper wishes to dispatch a shopping agent to a series of marketplaces collecting information about certain products or services;
- When an agent owner wants to control a remote working agent's itinerary;
- When a mobile agent does not want to carry too much sensitive information along its journey.

**Forces**

- Remote delegation and effective management

**Participants**

Figure 4.9 shows the structural relationships among the participants of this pattern.

- **Collector**: This is an abstract mobile agent class providing a simplified skeleton for an information-collecting agent. It is responsible for keeping in touch with the homeowner and reporting the results at each remote host whenever a delegation task is completed.
- **myCollector**: This is a concrete mobile agent class to implement the abstract method defined in the abstract 'Collector' class. A local market operator will be contacted, concrete information-collection approaches will be implemented.
• Policy: This is a separate functional module implementing the concrete server-side
decision-making scheme. It is responsible for generating a mobile collector agent
and dispatching it into the network. Offers collected by the agent will be used for
comparison, and a suggested destination will be worked out to direct the agent's
following trip. When the collection task is completed, it will dispose of the agent
at a remote site.

![Diagram of participants in the server-side comparison-shopping pattern]

Figure 4.9: Participants within the server-side comparison-shopping pattern

**Collaboration**

Collaboration among the participants within this pattern is depicted in Figure 4.10:

![Diagram of collaboration within the server-side comparison-shopping pattern]

Figure 4.10: Collaboration within the server-side comparison-shopping pattern

• A collector agent equipped with all necessary information for shopping is
dispatched to its first destination -- remote market 'A'.
• Upon arriving at the remote market, the collector agent tries to 'query' the local market operator, collecting related information about certain products and sending the results home.

• The 'Policy' module on the server side is called, a 'Compare' method and a 'goWhere' method are employed to decide whether the collector agent needs to go on its trip, and where to go for the next stop.

• If the collected offer is not satisfactory, the collector agent will be 'moved' to the next market 'B'. The same task will be performed and querying results will be returned.

• The 'Compare' method within the 'Policy' module is invoked again, a decision is made to stop the collector agent's trip. The collector agent will be 'disposed' of remotely.

Consequences

This pattern has the following benefits and drawbacks:

• The server-side comparison-shopping pattern provides a generic structure for remote delegation. It hides two types of sensitive information from the remote hosts: well-designed itinerary and decision-making policy. Although the proposed pattern is called 'comparison shopping' and used for a home shopper, it is also feasible for an itinerant agent representing a supplier to visit remote client hosts if only the 'query' method is modified. This helps a homeowner perform strong control on its worker agent over the network. Normally, the delegated task is simple and the itinerant agent is light.

• The presented pattern also has some obvious drawbacks because of the strong control over an itinerant agent. At first, the expected flexibility of a free-roaming
agent is seriously hindered. A remote mobile agent has to depend on frequent two-way communication to acquire the information to continue its journey. Two other observable problems are the working efficiency and communication reliability when choosing this pattern for the remote delegation purpose, because the delegating agent relies too heavily on its home-owner's decision.

**Implementation**

There are four basic classes within the server-side comparison-shopping pattern. A detailed implementation can be found in Appendix A.3.

- The 'Collector' class is an abstract mobile agent class extending 'Aglet', which takes a reference to the home shopper's policy in its constructor, which will be used to communicate with the server-side decision-making module when a querying result is received.
- The 'myCollector' class extends the 'Collector' class. The only requirement is to implement the 'query' function defined in its parent class. A shopping-item string will be used to query the local market operator and results will be sent back.
- The 'Policy' class is an objectified decision-making module installed on the server side (in the form of a stationary agent). It creates a concrete collector agent and equipping it with a shopping item and a 'Policy' proxy. A 'Compare' method is implemented to compare the current offer with a pre-defined expected value. The comparison result will be used in a 'goWhere' method for selecting the next available destination stored in its itinerary vector. All 'dispatch' commands and the final 'dispose' command will be issued from this class.
- The 'homeShopper' object is not shown in the collaboration diagram, its function is much like the 'Creator' object within the client-side comparison-shopping
pattern, which is responsible for binding all the above elements together. It initialise a shopping vector with all necessary information for shopping such as a querying string, an expected price, a home address, and a set of proposed destinations. It hands this vector to the 'Policy' object and lets it deal with the remaining comparison-shopping task.

**Related patterns**

This pattern is inspired by the *Master-Slave* pattern defined by Lange and Oshima (1998), where a master agent can delegate a task to a slave agent and control its life cycle. However, when applying this *Master-Slave* pattern within the mobile agent-mediated e-business environment, a serious security concern arises because a travelling slave agent carries too much sensitive information in its memory. Of these pieces of information, the mobile agent's pre-defined itinerary and its decision-making policy are two major targets for a dishonest agent host or a malicious third party over the network. The proposed server-side comparison-shopping pattern successfully hides the vulnerable information from possible attackers. A collector agent follows the command from its owner step-by-step. The information collected en route will be sent back home immediately and used to compute its next destination. Therefore, most attacks launched by a malicious entity become invalid. A known use for this pattern can be found within a mobile agent shopping system suggested by Mandry *et al* (1999), where a private decision-making strategy is suggested for carrying out at the home site to satisfy the security requirement. Another obvious advantage of such an arrangement is that the proposed mobile collector agent becomes lighter and simpler after these pieces of information have been taken out of its memory.
4.4.5 Itinerary-balancing pattern

Intent

The itinerary-balancing pattern defines a route-optimisation scheme for a family of mobile agents belonging to the same user and working for the same task.

Applicability

This pattern can be used in the following cases:

- When a stationary agent wants to delegate its task to a group of mobile agents who are expected to perform the same task at a number of destinations;
- When a group of mobile agents working at different hosts wish to share itinerary information with one another;
- When a group of agents need a route-balancing mechanism to help them filter redundant addresses for accomplishing a given task within the group.

Forces

- Group delegation and inter-group collaboration

Participants

Figure 4.11 shows the structural relationships among the participants of this pattern.

- doubleCheck: This class works as a reusable mobility listener which can be used to plug into each agent body responding to various mobility events, especially when the agent arrives at a new destination or leaves for its next stop.
- Employer: This is a basic mobile agent class. A group of employer agents working for the same owner carries out the same task at different destinations. Each agent is able to migrate from one host machine to another, while the whole group works
in a parallel manner. Inter-group co-ordination is required to help group members accomplish the given task efficiently.

- **logBook**: This object runs on a well-known server with a unique identifier. It maintains an itinerary log for a group of agents that have the same job ID. Its major responsibility is helping each agent to optimise its itinerary based on the consideration about other group members’ itinerary plan.

![Diagram of the itinerary-balancing pattern](image)

**Figure 4.11: Participants within the itinerary-balancing pattern**

**Collaboration**

Collaboration among the participants within this pattern is depicted in Figure 4.12:

![Diagram of collaboration](image)

**Figure 4.12: Collaboration within the itinerary-balancing pattern**

- A group of mobile agents is created and dispatched to a set of remote destinations. Each of these is assigned the same job ID and equipped with a 'doubleCheck'
listener. When an agent arrives at a new destination, an 'onArrival' method within the listener is triggered.

- The 'doubleCheck' listener calls a ‘doUpdate’ method in a 'logBook', which is responsible for registering the agent's current location and corresponding job ID to a backend database.

- The 'Employer' agent carries out its task at the current host. Before migrating to its next destination, the agent calls the 'check' method, which receives a list of potential destinations suggested by that host. The 'logBook' will be contacted for inter-group itinerary balancing purposes.

- When a 'doBalance' method is triggered, the 'logBook' retrieves a number of itinerary plans reported from all agent members within the same group, a redundant destination will be deleted, a balanced destination will be returned, and finally, the corresponding database will be refreshed.

**Consequences**

The itinerary-balancing pattern has the following benefits and drawbacks:

- Benefits of the proposed pattern can be summarised in two aspects. From the task delegation perspective, it provides a generic structure to combine the advantages of single-sequential computing and group-parallel computing. Improved working efficiency is an obvious merit. From the inter-group collaboration perspective, a centralised 'logBook' co-ordinator is introduced to help optimise multiple agents' itineraries through redundancy filtering and information balancing. A family of agents can use this pattern to co-ordinate their travelling plan within the group, and without their owner's direct involvement.
The suggested pattern also has some limitations. At first, the co-ordination activity is heavily reliant upon a centralised 'logBook', which can easily become a bottleneck for the overall system performance, because frequent communication presents a serious drawback to system reliability. In addition, the applicability of employing such a pattern on other inter-group collaboration tasks remains an uncertain issue. Due to the diversity of delegation tasks, it is difficult for the central co-ordinator to provide a standardised solution in all circumstances.

Implementation

There are three basic classes within the itinerary-balancing pattern. A detailed implementation can be found in Appendix A.4.

- The 'doubleCheck' class extends 'MobilityAdapter', which takes references to the 'logBook' proxy and the agent proxy from its constructor. The 'onArrival' method is overridden, which will be triggered when an employer agent arrives at a new destination. The 'check' method also needs to be implemented, which is required to retrieve a list of itinerary plans from the current host and send these to the remote 'logBook' for optimisation.

- The 'Employer' class is a base class for one of the agent members. It acquires a unique job ID in its constructor. A mobility listener 'doubleCheck' is initialised and plugged into the agent's memory. When 'running' at a remote platform, it carries out its business activity. Before it is ready to leave, the agent triggers the 'check' method in its listener.

- The 'logBook' class registers itself with a well-known address and makes its proxy available to the public. It establishes a separate database (in Hashtable) for each agent group, job ID is the key, 'history' record and 'future' plan from all group
members are recorded in its two inner vectors. Two specific methods have to be implemented properly: 'doUpdate' and 'doBalance'. The 'doUpdate' method only updates the history record, while the 'doBalance' method checks both history data and future itinerary and returns a balanced result.

**Related patterns**

This pattern is inspired by the *Blackboard* pattern presented by Deugo et al (2001), where a 'blackboard' component is provided for a collection of agents to monitor each other's behaviours and mutual progresses. However, a blackboard's function is limited to message notification, which seems passive. Additionally, implementing a blackboard pattern in a distributed environment is not an easy task because it lacks a proper management mechanism. In a mobile agent-mediated e-business situation, a group of mobile agents needs a more functional arrangement to co-ordinate each other's visiting plan. On the one hand, a free-travelling agent may encounter many copies of advertising agents at different hosts that may guide the agent to visit the same origin. On the other hand, the number of candidate destinations retrieved at each visited site is naturally unbalanced. Therefore, the 'logBook' is just designed as a centralised co-ordinator to perform both the redundant-itinerary filtering and the overall-plan balancing work.

### 4.5 Chapter summary

This chapter presents a collection of mobile agent-mediated e-business design patterns. To overcome the limitations of most current approaches, which simply apply OO techniques to agent-based e-business pattern design, this chapter suggests viewing the mobile agent-mediated e-business domain as an organic whole. A set of general
forces unique to this specific area is identified. One of these comes from an e-business system requirement: open market and service quality, which regards how to maintain an online business environment soundly. Three other forces come from agent-oriented properties: free mobility and self-control capability, remote delegation and effective management, and group delegation and inter-group collaboration. These indicate important issues about how to make use of mobile agents' computational advantages within associated e-business design, at the same time avoiding potential weaknesses.

Four design patterns are proposed in this chapter, which are related to virtual market management, agent-based shopping, and agent-to-agent collaboration respectively. The group-trading pattern helps a market operator set up a group-trading environment for seller and buyer agents with similar interests to establish local contacts. The agent-side and server-side comparison-shopping patterns are related to an agent's mobility. The former objectifies a mobile agent's itinerary and clarifies its self-navigational function. The latter defines a comparison-shopping scheme where a home shopper agent controls its delegating agent's itinerary from home. Finally, the itinerary-balancing pattern seeks to optimise a group of agents' itinerary through a centralised organiser. The consequence analysis and discussion of some known uses demonstrate the benefits and drawbacks of suggested patterns within concrete system design. This pattern list might be continually extended in the future as more requirements and trade-offs involved in this area become better understood.

The coming chapter will focus on another of the mobile agent's properties: adaptability. Various decision-making strategies for mobile agents within e-business applications will be studied. The applicability of employing an online algorithm for agent-based task scheduling and decision-making will also be investigated.
Chapter 5

Competitive Analysis of Online Agent-Mediated E-Business

5.1 Chapter introduction

The decision-making strategy plays an important role in mobile agent system design. One possible reason is that when a mobile agent travels around the Internet working for its owner, it is expected to deal independently with all types of uncertainties along the way. Addressing problems with imprecise and approximate information is regarded as an essential capability for mobile agents working within a real e-business environment. In contrast to many existing approaches, which apply pre-defined soft computing techniques or business rules to deal with various e-business situations, this chapter introduces an online competitive algorithm to achieve flexibility. A Travelling Agent For Travel Agency (TAFTA) problem is identified, and a Travel Agency Booking (TAB) algorithm is derived. In this problem-solving strategy, an online player is capable of making a timely decision without any assumption about future
requests or request patterns. The suggested online algorithm works well compared with the associated optimal offline algorithm against the same request sequences. Sets of simulations and algorithm comparisons are also carried out to demonstrate the feasibility and effectiveness of the proposed solution in both theoretical and practical e-business conditions.

5.2 Computing techniques for solving problems under uncertainty

Imprecise and approximate information might be frequently encountered within a real e-business environment. For example, a customer's degree of satisfaction about a product is always expressed in an approximate manner, an agent in an auction has difficulty acquiring secure knowledge of its partners' bidding strategy, while a company wishing to invest money in a business has to face a lot of uncertainties about the future of the market. Therefore, equipping an agent with adaptability to address these problems intelligently becomes both desirable and essential.

5.2.1 Soft computing for agent-mediated e-business

Traditional approaches to address these problems under uncertainty are so-called Soft Computing (SC) technology, a consortium name for a group of relevant techniques. The basic principle of SC, as Fortuna et al. (2001) argued, is "combining use of various computation techniques to achieve a higher tolerance level towards imprecision and approximation, and thereby new software/hardware products can be had at lower cost to be integrated in the real world". Probabilistic Reasoning (Hajek, 1993), Fuzzy Logic (Zadeh, 1975), Neural Network (Zeidenberg, 1990), and Genetic
Algorithm (Buckles & Petry, 1992) are all principal members of the SC family. Apart from Probabilistic Reasoning which is related to a two-valued logic to deal with randomness or stochastic uncertainty, three other members can behave as a many-valued logic to deal with fuzziness or linguistic uncertainty. Therefore, they are more suitable to be employed by an agent to improve its performance in an e-business area.

**Fuzzy logic for agent-mediated e-negotiation**

The Fuzzy Logic (FL) technique seeks to deal with the concept of partial truth, where the truth-value is between completely true and completely false. Different degrees of truth are represented by different degrees of membership in a fuzzy set. A collection of membership functions and fuzzy rules forms the basis for a fuzzy expert system to reason about data. The general inference process can be summarised as fuzzification, inference, composition, and defuzzification (optional). As Zadeh (1965) noted, the whole process can be viewed as a basic methodology to generalise a specific theory from a discrete form to a continuous fuzzy form. Some examples that employ FL for agent-based e-business include fuzzy e-negotiation agents suggested by Kowalczyk (2002), fuzzy e-auction agents proposed by He et al (2003), and fuzzy-logic based business-forecasting agents presented by Shen et al (2002).

Technically, most negotiation processes can be modelled as a constraint satisfaction problem (Mackworth, 1990), where a final agreement is reached when all parties' pre-defined constraints have been satisfied. Kowalczyk (2002) suggested introducing fuzzy set theory into this specific area that allows one to express different degrees of satisfaction towards different solutions. A fuzzy constraint-based negotiation process for autonomous trading can be illustrated in Figure 5.1:
Figure 5.1: A fuzzy constraint-based negotiation process between two parties
(Kowalczyk, 2002)

$C^a(x)$ and $C^b(x)$ denote the fuzzy set of individual interests of parties $a$ and $b$, while the conjunctive combination $C(x) = C^a(x) \land C^b(x)$ prescribes the common interests area. The objective of negotiation is to find a solution $x^*$ within the intersection area with the maximum joint degree of constraint satisfaction. To this end, e-negotiation agents incorporate some principles of utility theory. Under the multi-attribute utility theory, a value $v$ can be defined as a function of values of all its considering issues, $v(x_1,...,x_n) = \sum_{i=1}^{n} w_i v_i(x_i)$, $\sum_{i=1}^{n} w_i = 1$, $v_i(x_i) \in [0,1]$, where $x_i$ is the $i$-th issue needs to be considered, $v_i$ is the utility function of the $i$-th issue, and $w_i$ is the associated issue's weighting factor. If viewing the utility function as a fuzzy utility constraint -- that is, $C^p(x) \equiv v(x) \in [0,1]$, a final solution can be reached when each party's satisfaction level is equal to or greater than a prescribed threshold, i.e., $C^p(x^*) \geq C^{p^*} \in [0,1]$, where $x^*$ is a received offer, $C^p(x^*)$ is the evaluation of that offer, and $C^{p^*}$ is an acceptability threshold. Fuzzy utility function makes the constraint definition and evaluation processes within a multi-attribute-based negotiation more accurate and flexible.
Multi-agent market modelling by neural network

Neural Network (NN) is another important member of the SC family. A collection of neurons can be organised into layers or clusters, with each processing unit having a small amount of local memory. The inter-neuron connection strengths (or synaptic weights) are used to store knowledge, which can be acquired through a variety of learning processes (Haykin, 1994; Bishop, 1995). As Sarle (1997) observed, the NN technique is especially useful for solving those problems where hard and fast rules are not easily applied, but function approximation and imprecision tolerance capability are desired, and there are enough data and computing resources for training. At least, feedforward NN could approximate almost any finite-dimensional vector function on a compact set to arbitrary precision. Figure 5.2 shows an NN-based Foreign eXchange (FX) market model suggested by Grothmann (2002):

Figure 5.2: A four-layer feedforward NN for FX-market modelling (Grothmann, 2002)

The primary aim of an agent within a FX-market is trying to maximise its expected profits by properly anticipating changes of the FX-rate. The objective function can be expressed as

$$\frac{1}{T} \sum_{t} \hat{\pi}_{t+1} \cdot a_{t} \rightarrow \max \ln \left( \frac{p_{t+1}}{p_{t}} \right),$$

where $\hat{\pi}_{t+1} = E \left( \ln \left( \frac{p_{t+1}}{p_{t}} \right) \right)$. In this function, $a_{t}$ is the trading decision made by the agent, $\hat{\pi}_{t+1}$ is the anticipated changes of the FX-rate.
rate, \( t = 1 \ldots T \) is a set of trading periods, and \( p_t \) is the expected profits at the time slot \( t \). With this objective function in place, a price dynamics model for a single FX-market can be easily established, as shown in Figure 5.2 (left). It is a feedforward NN with four layers. The input layer is used to capture the dynamics of each financial time series. A number of agent neurons reside at the 'agent' layer, with each representing a decision-making unit to carry out three major tasks: information filtering, market evaluation, and acting (see Figure 5.2 [right]). Behaviours of each neuron are guided by the above objective function, which takes the FX-rate changes \( \pi_{t+1} \) as a target value. The sparse matrix \( A \) is used to generate a set of heterogeneous behaviours towards the same market situation. Fixed matrix \( B \) is employed to calculate the market excess that might be directly used for producing response. After being trained by a collection of real examples, this multi-agent-based NN structure is supposed to output an approximately desired evaluation result within an ever-changing FX-market.

Genetic algorithm for agent-supported portfolio selection

Genetic Algorithm (GA) was devised as a type of evolutionary computation (Holland, 1992). Many concepts of GA operators come from the biological field, such as population, fitness, reproduction, recombination, and mutation. There is always a population of structures, with each individual assigned a measure of fitness to indicate its competence within the environment. Recombination and reproduction help exploit the available fitness individuals, while the mutation operator seeks to achieve diversity in the evolutionary course. Multi-dimensional searching and optimisation is a major application suitable for this algorithm, where an array of character strings
encodes values of different parameters being optimised. A general implementation process of GA can be demonstrated as below:

```
t = 0; // start with an initial time
initpopulation P (t); // initialise a usually random population of individuals
evaluate P (t); // evaluate fitness of all initial individuals of population
while not done do // test for termination criterion
    t = t + 1; // increase the time counter
    P' = selectparents P (t); // select a sub-population for offspring production
    recombine P' (t); // recombine the “genes” of selected parents
    mutate P' (t); // perturb the mated population stochastically
    evaluate P' (t); // evaluate its new fitness
    P = survive P, P' (t); // select the survivors from actual fitness
od
```

Figure 5.3: A general implementation process for genetic algorithm
(Sarle, 1997)

Three essential steps are included in this implementation cycle: evaluating the fitness of all initial individuals in the population, generating a new population through crossover, recombination and mutation, and finally replacing the old population with a new one. Shapcott (1992) suggested an investment agent for portfolio selection making use of GA and Quadratic Programming (QP) techniques. Given that all the shares in a market index form a set, GA is essential in selecting a subset of shares that best matches the average performance of the whole market. Then, given a subset of shares, QP is used to calculate the deviation of each chosen portfolio from the whole index, with the aim of minimising the tracking error. Therefore, the proportion of capital to be invested in each company is a particular subset of shares with the highest fitness values. Based on the algorithm comparison, Shapcott (1992) claimed that a simple random-search strategy could only achieve a reasonable performance, GA with isolated subpopulations performed better, whereas GA introducing migration performed the best, because it enhanced its resistance to premature convergence. In other words, GA does benefit from its sophistication over the simple random algorithm. Bigger searching space coverage is a significant advantage.
5.2.2 Hybridisation of various soft computing techniques

From the above discussion, it can be seen that every technique has its own strengths within its specific application domain. Fuzzy Logic is good at dealing with approximate reasoning and information granularity, Neural Network has a very strong learning and adaptation capability, while Genetic Algorithm tries to employ a stochastic searching approach to achieve an optimal or near optimal solution. A detailed comparison about these three techniques can be found in Goonatilake and Khebbal (1995). A brief summary is given in Table 5.1. Properties being assessed include automated knowledge acquisition, coping with brittleness, and explanation, where symbol (*) indicates the assessment level, the more the stronger.

<table>
<thead>
<tr>
<th>Soft computing techniques</th>
<th>Assessed properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Automated knowledge acquisition</td>
</tr>
<tr>
<td>Fuzzy logic</td>
<td>*</td>
</tr>
<tr>
<td>Neural network</td>
<td>*****</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>*****</td>
</tr>
</tbody>
</table>

Table 5.1: Property assessment of three soft computing techniques

(Zhang, 2001)

Automated knowledge acquisition is an essential capability for interpreting knowledge within a specific domain. FL cannot properly deal with changes in its decision-making environment, and thus has poor knowledge acquisition capability. Both NN and GA perform very well in this perspective: they can learn domain knowledge online or offline, which may help them detect and adapt their behaviour for facing various changes in their working situations. The concept of 'brittleness' indicates that some systems are operational only within a very narrow domain, with limited conditions (Sell, 1985). For SC techniques, the property of coping with brittleness is regarded as the capability to deal with incomplete or inconsistent
information. FL makes use of the fuzzy set to represent knowledge with diffused concepts' boundaries. Inexact or partially correct concepts could be processed properly with an approximate reasoning mechanism. NN also has some kind of fault tolerance capabilities, because the knowledge representation and reasoning in NN may be highly distributed. As a consequence, imprecise data may result in performance degradation, but the whole system would still function gracefully. GA improves its non-brittle capability through maintaining a population of solutions with rules describing a wide range of competing hypotheses. The rules' selection depends on a statistical aggregation of the correctness from its past performance, which is very similar to the statistical reasoning property of NN. Finally, the explanation facility tries to convince users of the soundness of the reasoning procedure. FL and GA provide a rational explanation capability. The former achieves this through a collection of fuzzy rules with a simple 'IF-THEN' format, while the latter performs this through setting up a chain of rule-based reasoning models. Unfortunately, NN meets certain difficulties in producing an adequate explanation, the major reason being that the domain knowledge has been encoded as weights distributing over the entire networking structure.

Based on the above comparison, it is reasonable to conclude that it is not practical to employ a single SC technique to tackle all problems regarding uncertainty. When facing a real complex problem, hybrid use of these techniques is more appropriate. At least, combining FL with NN to form a FNN (Fuzzy Neural Network) and combining FL with GA have been proven successful (Kasabov et al, 1998; Welstead, 1994). A number of multi-agent systems based on hybrid SC techniques have been presented in the literature (Hilario et al, 1994; Khosla & Dillon, 1997). However, as Zhang et al (2001) argued, most of these approaches suffered from an
obvious drawback. As each agent is fixed to a single SC technique upon being created, it is difficult for an agent designer to equip the agent with more intelligence at run time. In order to overcome this limitation, Zhang et al (2001) suggested an agent-based SC society, as depicted in Figure 5.4:

Three kinds of agents are offered: a serving agent, a collection of SC agents, and a group of problem-solving agents. Problem-solving agents work at the front end to receive a user's requirements and produce solutions. SC agents, with each having a specific SC technique, work at the back end. They do not belong to any single problem-solving agent, but are accessible to all of them via a service broker (or the serving agent). Zhang et al (2001) examined this SC society in a financial investment context. At first, a problem-solving agent is asked to evaluate an investor's risk tolerance capability. A SC agent with FL technique will be chosen for this purpose because fuzzy rules are suitable for expressing relations between financial risks and various evaluation criteria, such as the investor's annual income and investment amount. The second task is making predictions on the interest rate in the market. Two kinds of solutions are available: one is employing feedforward NN, the other is using FL with GA. The serving agent is authorised to select one of these, based on whether the history data need to be considered or not. Finally, FL-based Ordered Weighted
Averaging (OWA) algorithm (Yager, 1992) is used to aggregate the portfolios returned from different evaluation models in order to calculate the most profitable portfolio selection. Through the SC society, a problem-solving agent is capable of using all available techniques without adding much complexity in its own memory. Additionally, possible changes on society members would not affect the system's general functionality. Thus, the system's shareability, flexibility and robustness are significantly improved.

5.2.3 Online problems and competitive algorithm

Although various SC techniques and their simple combinations address problems under uncertainty through different ways, they share one common characteristic, namely, they always assume the algorithm designer has secure knowledge of all future events or event sequences. Taking the NN technique as an example, such knowledge comes from its training examples. In order to help a NN acquire necessary intelligence, one has to provide it with enough training data, including all possible request patterns. Although its initial aim for NN construction is trying to get some generalised knowledge from these training data and produce roughly correct results for new cases, it has to be acknowledged that NN can not magically create information which is not included in its training set. By contrast, there is another kind of uncertainty problem beyond the capability of these SC techniques, but still frequently encountered in the computing and business applications -- that is, the online problem.

According to Borodin and El-Yaniv's (1998: 25) description: "online problems are optimization problems in which the input is received in an online manner and in which the output must be produced online". More precisely, if a request sequence is
denoted as $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$, and each request is denoted as $\sigma(i)$, $1 \leq i \leq m$, when a decision-maker tries to serve the request $\sigma(i)$, it does not have any knowledge about requests $\sigma(i')$, $i < i' \leq m$, and cannot assume any distribution pattern about the future requests either.

A well-studied example is the online paging problem, where there is a two-level memory system consisting of a fast memory and a slow memory. The fast memory is relatively small, but pages in the fast memory can be accessed very quickly without any cost. The slow memory is larger, but when the pages in the slow memory need to be accessed, they first have to be moved to the fast memory, which will involve a cost. When a requested page is not in the fast page (or a page fault occurs), the system needs to evict an old page currently in the fast memory to make room for the coming requested page. Which page has to be moved out of the fast memory is an online problem. Although theoretically any page might be taken out of fast memory, to avoid evicting a page that might be requested in the near future is desirable due to performance concerns. Facing the paging problem, a number of online algorithms have been suggested including LRU (Least Recently Used), FIFO (First In First Out), LIFO (Last In First Out), LFU (Least Frequently Used), and others. Studying these online algorithms might be helpful to clarify a set of basic concepts in the online computing field. So, a further explanation about these strategies will be given below:

**LRU**: when a page fault occurs, evicts the page whose most recent request is the oldest.

**FIFO**: this strategy moves out the page that stays at the fast memory the longest.

**LIFO**: when eviction is necessary, moves out the most recently page.

**LFU**: this strategy takes out of the page that has been accessed the least compared with other pages in the fast memory.
To analyse online algorithms quantitatively, it is necessary to define an optimal offline algorithm. The offline decision-maker is supposed to have complete knowledge of all requested pages \( \sigma = \sigma(1), \sigma(2), \ldots, \sigma(m) \), thus, it is capable of forming an optimal plan to avoid all unnecessary moving costs. Intuitively, when serving the current request, an optimal plan should move the page that has the longest forward distance in the future requests, so the system could minimise the times required to swap an evicted page back to the fast memory. Such algorithm is usually called LFD (Longest Forward Distance) (Belady, 1966).

The performance of an online competitive algorithm is defined on the basis of its corresponding optimal offline algorithm, which was introduced by Sleator and Tarjan (1985) in their study of "Snoopy Caching" problem. More formally, if for any possible sequence of request \( \sigma \), let \( C_A(\sigma) \) denote the cost of an online algorithm \( A \), \( C_{OPT}(\sigma) \) the cost of the optimal offline algorithm \( OPT \), \( A \) is called \( \alpha \)-competitive if relation (5.1) holds:

\[
C_A(\sigma) \leq \alpha \cdot C_{OPT}(\sigma) + \beta
\]

where \( \alpha \) is called the competitive ratio, \( \beta \) is an additive constant for initialisation. When \( \beta \) is equal to zero, algorithm \( A \) is called strictly \( \alpha \)-competitive. With the help of the above concepts, it is not difficult to prove that for the online paging problem, LRU and FIFO are \( k \)-competitive algorithms, where \( k \) is the size of the fast memory, while LIFO and LFU are not competitive at all.

Taking LRU as an example, it is necessary to prove that for any possible request sequence \( \sigma \), it has \( C_{LRU}(\sigma) \leq k \cdot C_{OPT}(\sigma) \). To this end, an input request sequence is divided into several phases, with each phase having exactly \( k \) page faults. Based on LRU's definition, LRU makes \( k \) page faults in all, whereas the optimal offline
algorithm has to make at least one page fault for the same phase (even if it gets the full knowledge of the entire request sequence). Therefore, one can say that the cost of LRU algorithm is at most $k$-times larger than that of the optimal offline algorithm. A detailed proof for the competitiveness of LRU algorithm can be found in Albers (1996), where two additional cases during each phase are further identified: page fault incurred twice on the same page, or on $k$ different pages. Technically, FIFO, LIFO and LFU algorithms can be proved in a similar way (Sleator & Tarjan, 1985).

5.3 Competitive analysis within business and mobile agent domains

Although the mathematical proof indicates that both the LRU and FIFO algorithms can achieve the same optimal competitive ratio $k$, it is a well-known fact that LRU performs much better than FIFO in practice. And experimental results also demonstrate that the empirical competitive ratio of LRU is not an increasing function of the size of the fast memory, but nearly a small constant (Fiat et al, 1993). One possible reason for such discrepancy between theory and practice, as Borodin and El-Yaniv (1998) recognised, is that the competitive analysis always takes into account all possible bad inputs, which in reality may not happen frequently. However, when dealing with those deterministic algorithms where their behaviours can be predicted completely from the inputs, the current adversary model seems to be natural, which is more like a two-player game. An online player runs an online algorithm on the request sequence generated by an offline adversary. The offline adversary gets to know the online player's strategy in advance (and thus is capable of creating the worst possible inputs), with the aim of maximising its rival's cost and at the same time making the task inexpensive to itself. However, two intuitive questions arise: whether the offline
adversary could have so much power, and whether the online player could choose a randomised strategy to minimise cost differences with its offline rival.

5.3.1 Selecting different adversary models

In order to answer these questions, Ben-David et al (1994) introduced three kinds of adversary models: OBL (OBLivious), ADON (ADaptive ONline), and ADOF (ADaptive OFfline). The primary aim is to classify the power of an adversary into different levels, thus improving the competitiveness of some online algorithms against different adversaries, and minimising the discrepancy between theoretical and empirical results.

**ADOF**: This is the most powerful adversary. The ADOF adversary knows the current action taken by the online player, and is capable of choosing the worst possible input as the next request to maximise the online player's cost. Moreover, the cost it has to pay itself is the optimal offline cost against the same inputs.

**ADON**: This kind of adversary shares the same adaptability with ADOF in selecting the next request, based on its knowledge of its online rival. However, it has to pay the online cost by itself. It is less powerful than ADOF, because if the online player chose a randomised algorithm, the ADON adversary would meet some difficulty to maximise the online player's cost and optimise its own at the same time.

**OBL**: This kind of adversary is a standard form for analysing online randomised algorithms. The OBL adversary is required to generate a request sequence in advance. An online player pays online costs while the OBL adversary is able to pay optimal offline costs. In other words, the OBL adversary is not allowed to create the worst possible request sequences for an arbitrary online algorithm.
With respect to these three kinds of adversaries, the competitive ratio of a randomised online algorithm can be re-defined, with the help of a new measurement of expected-cost (Borodin & El-Yaniv, 1998). Let \( E[C_A(\sigma)] \) and \( E[C_{ADV}(\sigma)] \) denote the expected costs incurred by a randomised online algorithm \( A \) and an offline adversary (ADOF, ADON or OBL), the algorithm \( A \) is called \( \alpha \)-competitive if the equation 
\[
\beta \sigma_A \leq \alpha \cdot E[C_{ADV}(\sigma)] 
\]
holds for all possible request sequences \( \sigma \), where \( \beta \) is a constant. Albers (1996) presented two other essential theorems on the inner relationship among these three adversaries. First, "if there is a \( c \)-competitive randomised algorithm against ADON adversary and there is a \( d \)-competitive algorithm against OBL adversary, then there must have a \( (c \cdot d) \)-competitive algorithm against ADOF adversary". Second, "if there is a \( c \)-competitive randomised online algorithm against ADOF adversary, there also has a \( c \)-competitive deterministic online algorithm". That means a randomised strategy cannot improve its competitiveness facing the ADOF adversary.

Following the above theory, Fiat et al (1993) proposed a randomised Marking algorithm for the previous online paging problem. The selected adversary model is the OBL.

**Marking**: All pages in the fast memory are unmarked at the beginning, becoming marked only when first requested. If the requested page is not in the fast memory, the evicted page will be chosen randomly from all the unmarked pages. If all pages are marked and a page fault occurs, all marks will be cleaned, and a new phase started.
The Marking algorithm is proved to be $2H_k$-competitive against an OBL adversary, where $H_k$ is the $k$-th Harmonic number, $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$, which is roughly $\ln k$. For analysing the performance of this randomised algorithm, a request sequence is divided into several phases, with each phase having $k$ distinct pages ready to be requested, so that all pages in the $k$-sized fast memory would be marked at the end of each phase. This can prove that the expected page faults incurred by Marking are at most $cH_k$, while those made by an optimal offline algorithm during the same phase are at least $\frac{c}{2}$. A final conclusion can be made after incorporating these two statements. It can be seen that the selection of the OBL adversary model improves the competitive ratio of a randomised online algorithm significantly, from $k$ to $2\ln k$.

5.3.2 Adopting general proof techniques

Two general techniques are frequently employed within competitive analysis: one is Tarjan's (1985) potential function, the other is Yao's (1997) minimax principle. The former is used to analyse the performance of a deterministic online algorithm, while the latter is often used for developing lower bounds on the competitive ratio achieved by a randomised online algorithm against an OBL adversary.

For the potential function method, an average cost for a sequence of requests is taken into account. For any possible request sequence $\sigma = \sigma(1), \ldots, \sigma(m)$, let $C_A(t)$ and $C_{opt}(t)$ denote the average cost incurred by an online algorithm $A$ and an optimal offline algorithm $OPT$ for serving any individual request $\sigma(t)$ $1 \leq t \leq m$, a potential function $\Phi$ can be expressed as $C_A(t) + \Phi(t) - \Phi(t-1) \leq c \cdot C_{opt}(t)$, where $\Phi(t)$ is the value of the potential function after satisfying the current request $\sigma(t)$. Summing up
the performance for serving each individual request $\sigma(t) \, 1 \leq t \leq m$, an online algorithm $A$ is called $c$-competitive if the equation $\sum_{t=1}^{m} C_A(t) + \Phi(m) - \Phi(0) \leq c \cdot \sum_{t=1}^{m} C_{OPT}(t)$ holds. Therefore, the main task for this approach is constructing a potential function $\Phi$ and proving the above equation is true for all individual requests. Albers (1996) gave a proof for the competitiveness of the online LRU algorithm based on this technique. The potential function is developed as $\Phi = \sum_{p \in S} \Phi(p)$, where $S$ is the set difference between $S_{LRU}$ (the set of pages in LRU’s fast memory) and $S_{OPT}$ (the set of pages in OPT’s fast memory), and $w(p)$ is the weighting factor of page $p$ in $S_{LRU}$ with a range of $[1,k]$. It can be proved that the cost of the potential function increases by at most $k$ whenever OPT has a page fault, and if LRU meets a page fault, the cost of the potential function decreases by at least 1. Hence, the equation $C_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k \cdot C_{OPT}(t)$ holds for all possible requests, and LRU is $k$-competitive.

Another general technique for competitive analysis is Yao's (1997) minimax principle. It indicates that the best competitive ratio of a randomised online algorithm against the OBL adversary is the same as the best competitive ratio of a deterministic algorithm under the worst-case request distribution. More formally, Yao's minimax principle can be expressed as $\inf_{R} c_{R} = \sup_{P} \inf_{A} c_{A}^{P}$, where $c_{R}$ is the best competitive ratio of a randomised online algorithm $R$, $c_{A}^{P}$ is the best competitive ratio of a deterministic algorithm $A$ under a probability distribution $P$ for selecting a request sequence. Therefore, constructing a probability distribution $P$ and developing a lower bound on $\inf_{A} c_{A}^{P}$ for a deterministic algorithm $A$ is the main task for computing the lower bound on the competitive ratio of a randomised online algorithm. Albers (1996)
applied this strategy to prove that $H_k$ is the lower bound for any randomised online paging algorithm. If partitioning each request sequence into several phases, with each having $k+1$ distinct pages, these $k+1$ pages would form a set $S$. For every request $\sigma(t), t > 1$ the page is chosen randomly from $S \setminus \{\sigma(t-1)\}$. It can be proven that the expected cost for any randomised online paging algorithm is $1/k$, because it is the probability of a page in the fast memory being hit. Also it can be proven that the expected length of each phase is $kH_k$, which makes use of an existing result in the complete graph theory (see Fiat et al, 1993). A final conclusion can be reached after combining the above two statements.

5.3.3 Addressing online problems within the business field

Employing competitive analysis for economics and financial games has attracted much attention. One important reason is that many problems within the business field are inherently online. For example, a market operator is unable to make too many assumptions about future market conditions, yet has to make a timely decision based on its current observation. Based on El-Yaniv's (1998) identification, at least three kinds of problems are suitable for competitive solutions: leasing and replacement, randomised searching and one-way trading, and portfolio selection and two-way trading.

Leasing and replacement problems

In the online leasing problem, the following scenario is considered: a player who wants to use certain equipment has to face two choices each day: renting the equipment at $x$ dollars or buying it at $kx$ $(k > 1)$ dollars. Clearly, buying would be a better idea if he/she decides to use the equipment for a long time, otherwise, renting
would be better. The online leasing problem can be expressed as when to stop renting and choose buying with the aim of minimising overall spending cost on that equipment. As an interesting variant, Irani and Ramanathan (1998) studied a car-renting problem where price of the car varies from day to day (within a reasonable range). If $c$ denotes the cost of renting, and the price of buying is fluctuating within $[1, M]$, Irani and Ramanathan (1998) suggested the customer bought the car when either the price of buying on the $i$-th day dropped below $P = (1 + \sqrt{4M + 3})/2$, or the cumulative amount cost spent on renting was over $R = P - 1 - c$. For a complete proof, please refer to Irani and Ramanathan (1998). In addition to this generalised deterministic algorithm, a randomised algorithm is also studied in the literature, where a cumulative distribution function and a probability distribution function are explicitly introduced (Phillips & Westbrook, 1999). Additionally, an interest-rate-involved algorithm is also proposed by El-Yaniv et al (1999) where an interest rate factor is considered, making the associated financial model more practical.

The above leasing problem can be viewed as a rudimentary form of a more generalised replacement problem. For the replacement problem, a player is engaged in an activity associated with two items: a replacement cost and a flow rate. New activities might replace the current one with a different rate. If the player switches from one activity to another, he/she has to pay the replacement cost. The problem is when to replace the current activity with a new one for achieving minimised total costs. If viewing a small replacement cost as rental and a higher cost as purchase price, it can be seen that the leasing problem is a special case of the replacement problem when the player chooses the buying activity. More variations of the replacement problem can be found, such as the menu-cost problem (Sheshinski &
Weiss, 1993), the mortgage-refinancing problem (El-Yaniv & Karp, 1996), and the equipment replacement problem (Azar et al, 1999).

**Randomised searching and one-way trading problems**

The randomised-searching problem describes another interesting scenario: an online player is searching for the highest price in a sequence of offers, at each period of time he/she obtains a price quotation with a sampling cost. The player has to decide whether to accept the current offer or to continue asking for more samples. The game is over after the player accepts an offer, the aim being the highest total return for the capital layout. One-way trading is an important application for this randomised-searching problem. In this application, a trader is trying to exchange one holding currency (i.e. dollars) to another currency (i.e. yen). At each period of time the trader might accept the current exchange rate or wait for a better quotation in the future. A small sum of transaction fees can be viewed as the sampling cost in the randomised-searching problem.

A RPP (Reservation Price Policy) algorithm has been given by Borodin and El-Yaniv (1998). It assumes the price (or exchange rate) is fluctuant among an interval $[m,M]$, where the fluctuation rate is defined as $\phi = M / m$. The RPP strategy accepts the first price greater than $p^* = \sqrt{Mm}$, because an intuitive observation is that the reservation price $p^*$ should help equate the offline-to-online return ratio no matter whether or not the maximum price can be encountered. Based on this simple RPP strategy, a randomised EXPO (EXponential) threshold algorithm is also suggested (El-Yaniv, 1998). Assuming the global fluctuation rate $\phi = 2^i$ for $i = 0,1,...,k - 1$, this algorithm chooses the deterministic RPP strategy ($RPP_i$) and reservation price ($m2^i$) with probability $(1/k)$. This strategy improved the associated competitive ratio from
In order to get a further optimal performance, El-Yaniv et al (2001) presented a threat-based algorithm, which was based on the assumption that an adversary would drop the exchange rate to its minimum at any moment in the game and maintain it for the rest of the time. Therefore, it accepts the current rate only when it is the highest seen so far. And at each time of conversion it changes just enough currency to ensure that a competitive ratio \( c^* \) for any deterministic trading algorithm can be achieved. The provable competitive ratio for this approach is \( c^*_\infty(m, M) \), where \( m \) and \( M \) are the minimum and maximum values of the exchange rate respectively.

Many variants for this problem exist, such as searching with or without recall, sampling with or without discount, and searching for the highest or the lowest price. One may refer to Borodin and El-Yaniv (1998) for more information.

**Portfolio selection and two-way trading problems**

In contrast to the above two online problems, the portfolio selection problem has gone slightly beyond the traditional competitive analysis approach and employed some basic methodology from the mathematical finance field. It considers an investor who has certain initial wealth and wishes to invest securities in a market. He/she has to decide how to rearrange the current investment at the beginning of each trading time in order to maximise the total wealth at the end of the game. A special case, where there are only two securities in the market and one of them is cash, is called the two-way trading problem. Traditional solutions include the buy-and-hold strategy where some portfolio was selected and held for quite a long-time, and the constant-rebalance strategy where securities were bought and sold more frequently (Borodin et al, 2000). Unfortunately, both of these algorithms meet a serious performance problem -- they may lose money, that is, generate a negative profit.
As Chou et al (1995) argued, one possible reason is that there are no constraints imposed on the offline adversary when it selects the worst-case market sequence, and in reality some extremely bad input sequences may not occur. Therefore, Chou et al (1995) introduced a statistical adversary (Raghavan, 1991) to generate market sequences where some additional statistical parameters have to be obeyed. An online money-making algorithm against this statistical adversary is offered. Two constraints applied in the price sequence: \( n \) and \( \phi \), where \( n \) is the sequence length, and \( \phi \) is the associated optimal offline return. The latter can be expressed as 
\[
\phi = \prod_{i=1}^{n} \max\{1, x_i\},
\]
where \( x_i \) is the relative price for any specific securities. With the help of these two constraints, an online player can adjust its behaviour based on its knowledge of the wealth return accrued by the optimal offline adversary, simply with one-day delay. An improved competitive ratio is \( \max\{n-1, \phi\} \). In addition, a family of \( \mu \)-weighted algorithms has also been given (Borodin et al, 2000), where a new measurement using maximin or minimax regret rather than the traditional competitive ratio was employed to examine the competitiveness of a portfolio selection algorithm.

### 5.3.4 Employing competitive algorithm for mobile agent applications

The initial idea of employing a competitive algorithm for mobile agent-mediated e-business comes from two intuitive observations. First, a mobile agent decision-maker is often facing the problem such as where and when to go to accomplish a given task, while minimising computation and communication cost. This is exactly the aim for a competitive algorithm to achieve -- minimised online cost compared with the optimal offline cost, against all possible request sequences. Second, viewing e-business as an online version of conducting traditional business activity within cyberspace, mobile
agents are expected to perceive various market conditions autonomously, making decisions based on their knowledge and adjusting their strategies accordingly. Competitive analysis seems to be a good candidate for improving a travelling agent's adaptability within such a changing environment. This subsection will study the applicability of employing competitive algorithms for mobile agent-based task scheduling. Mobile agent-mediated decision-making will be discussed in the next section.

Consider a mobile agent-mediated after-sale service system. When a client is unsatisfied with the use of a certain product, he/she may simply contact a local representative of the service provider for making a complaint. After receiving the call, the service provider must dispatch an online technical team to serve the customer's request, as illustrated in Figure 5.5. For simplicity, there are only four clients and two technical teams depicted in the proposed system. In reality there could be more, but it is reasonable to assume that the number of online technical teams is fewer than the number of clients. The service provider in the proposed system is required to schedule two technical teams among four potential clients online.

Figure 5.5: A mobile agent-mediated after-sale service system
A reasonable strategy is to arrange the nearest technical team to serve each request. Unfortunately, such an arrangement is not always efficient. Consider the following scenario: Technical teams 1 and 2 initially reside at Host 1 and 2 respectively, Client 3 who resides in Host 3 asks for the service for the first time. Technical team 1 is dispatched because the distance between Host 1 and 3 is shorter than that between Host 2 and 3. After satisfying the current request, Team 1 stays at Host 3, waiting for future command. Unfortunately, the second request comes from Client 1 (in Host 1), so Team 1 has to be dispatched back to Host 1 because it is still the nearest available team. The worst-case scenario might be the request sequence $\sigma = host(3), host(1), host(3), host(1),...$; Team 1 is sent back and forth between Host 1 and 3, while Team 2 sits idle at Host 2 all the time. Clearly, if the service provider received the entire request sequence in advance, it might dispatch Team 2 to Host 3 the first time. Although it seems inefficient for the first time, it saves the following costs for subsequent task fulfilment. However, the real problem lies in that the service provider does not know the future requests before making a decision to satisfy the current one, nor does any assumption about the possible distribution of requests seem convincing.

Interestingly, a similar problem can be found within the competitive analysis field -- the $k$-server problem. It considers a graph with $M$ vertices and $k$ servers. A request from any vertex in the graph must be satisfied by immediately placing a server there. Moving a server to a vertex involves a serving cost (which is related to the distance between relevant vertices), the primary aim being to minimise the total cost for satisfying all possible request sequences. For a special case where $k = 2$, a Residue (RES) algorithm has been suggested by Manasse et al (1990), as summarised below.

The RES algorithm always keeps two mobile servers on two different vertices and tries to compare a set of relevant residues before making a decision. Assuming
there are $n$ vertices in the graph, currently Servers 1 and 2 occupy Vertices 1 and 2 respectively, as Vertex 1 is the last requested vertex, an offline algorithm has to cover Vertex 1, thus $(n-1)$ non-infinite residues must be available. Let $R_{ii}$ denote the residue that compares RES to an offline algorithm covering Vertices 1 and $i$, an initial set of residues can be calculated as $R_{ii} = d_{12} + 2d_{2i}$, where $d_{12}$ and $d_{2i}$ denote the distance between Vertex 1 and 2, and between Vertex 2 and $i$ respectively. To satisfy a current request at Vertex $i$, RES dispatches the server from Vertex 1 if $\min_{k} \{ R_{kk} + 2d_{ki} \} \geq d_{12} + 2d_{ii}$, otherwise the server from Vertex 2 will be dispatched, where $k$ is the remaining vertices except for the last requested one. So, after servicing the current request from Vertex $i$, Vertex $i$ becomes the new Vertex 1, denoting as 1'. If the server from Vertex 1 were moved, another static Server 2 would be denoted as 2'. The updated residues are:

$$R'_{1,j} = \begin{cases} 
\min_{k} \{ R_{kk} + 2d_{ki} \} - d_{ii} & \text{if } j = 1 \\
R_{1j} + 2d_{ii} & \text{otherwise}
\end{cases}$$

If the server from Vertex 2 were moved, another static Server 1 would be denoted as 2'. The updated residues are:

$$R'_{2,j} = \begin{cases} 
\min_{k} \{ R_{kk} + 2d_{ki} \} - d_{2i} & \text{if } j = 2' \\
R_{2j} + 2d_{ii} - d_{2i} & \text{otherwise}
\end{cases}$$

Where $j$ in above two equations denotes the remaining vertices, except for the current requested one. Such a process will be repeated whenever a following request comes by simply replacing Vertices 1 and 2 with the updated Vertices 1' and 2'. A complete proof of this algorithm can be found in Manasse et al (1990).

Incorporating the RES algorithm into the above agent system is quite natural. Two technical teams represent the two moving servers, four clients are the four
vertices. The service provider employs the RES algorithm to deploy the technical teams. A mathematical proof given by Manasse et al. (1990) indicates the competitive ratio of the RES algorithm is 2, which means the cost of RES is at most twice the cost incurred by an optimal offline algorithm. In order to examine the performance of the proposed system in an online environment, a simulation experiment is conducted. IBM's Aglets is chosen as the testing platform, as it is easy to run multiple agent contexts at different ports on the same computer. Distances among different contexts are defined as serving cost, and request sequences are generated randomly. Simulation results of 100 trials are given in Table 5.2:

<table>
<thead>
<tr>
<th>Number of trials</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>1.833</td>
<td>1.881</td>
<td>1.951</td>
<td>1.940</td>
<td>1.947</td>
</tr>
<tr>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>1.963</td>
<td>1.957</td>
<td>1.969</td>
<td>1.967</td>
<td>1.970</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Incorporating RSE algorithm for proposed after-sale service system

It can be noted that the competitive ratio of the RES algorithm is fluctuant but not beyond a lower bound 2, which is actually the best theoretical result for the online $k$-server problem when $k = 2$. If the number of online technical teams increases to, say, $2 < k < (n-1)$, the RES algorithm is no longer optimal. Instead, a work function algorithm suggested by Koutsoupia and Papadimitriou (1995) is the best solution so far, which is proved to be $(2k-1)$-competitive. If the number of online technical teams reaches $(n-1)$, a Balance algorithm can be employed (Fiat & Woeginger, 1998). For the online $k$-server problem when $k = n - 1$, the Balance algorithm can achieve the best competitive ratio of $(n-1)$. Therefore, a central scheduler in the proposed agent system is capable of choosing different strategies under different conditions to improve its adaptability and flexibility.
5.4 Online decision-making for mobile agent-mediated e-business

In most task-scheduling applications, it is often assumed that multiple agents working within the same system are naturally co-operative. However, for most business-actor-oriented applications, such an assumption seems impractical, because mobile agents representing different owners may have different interests. Customers seek to purchase their preferred products with minimum expenditure, whereas vendors try to clear their inventory with maximum gross return. Therefore, the decision-making task in a mobile agent-mediated e-business environment is more challenging.

5.4.1 Market-oriented mobile agent planning

As business rules have provided us a sound solution to deal with similar complex issues in our daily life, it is natural to speculate as to whether these business rules or regulations can be directly employed within the mobile agent-based e-business domain. The answer is yes. The MAgNET system presented by Dasgupta et al (1998), the utility-driven market suggested by Bredin et al (1998a, 1999a), and the Kasbah market proposed by Chavez and Maes (1996) are all well-designed examples.

The MAgNET system is a classic instance (Dasgupta et al, 1998), which has been analysed in subsection 3.4.3 as a SSMB model for agent-based e-business. Here a deeper view will be given into its decision-making algorithm. A sales agent visits a collection of potential buyers on behalf of its owner. Such an arrangement is claimed to be suitable for marketing those commodities with a short shelf life, therefore, the sales agent has to clear all stock before it expires. Before dispatching a sales agent, the supplier provides it with an expected price for the product and current stock
available for sale. After visiting each client, the marketing agent calculates an optimum price presenting to the next client, based on its current sales information and a pre-defined demand-and-supply curve $p = m_d q + c_d$, where $p$ is the calculated price, $q$ is the quality that the next client is expected to buy, $m_d$ and $c_d$ are associated parameters obtained from prior sales. If the provided price was too high, the amount purchased by the next client would be less than expected, then the agent would have to lower the price in order to generate more sales by successive buyers. Otherwise, it could raise the price for a possible higher return. As Dasgupta et al (1998) argued, the derived profit maximising algorithm works well in most simulation scenarios. However, they also acknowledged that the suggested demand-and-supply curve is assumed to be linear and continuous, which to some extent is naive. Higher-order and discontinuous curves would be more accurate for describing clients' behaviours within a real market, but these also bring computational complexities. In fact, many other agent-mediated e-business applications also meet similar limitations (Chavez & Maes, 1996; Bredin et al, 1998a).

A possible reason for such a drawback is that most business rules are merely a summary of regularities for ordinary business activities. Therefore, when applying these rules to a real marketing situation, it always assumes the future request pattern conforms to the same summarised regularity. If meeting with distribution irregularity or encountering certain isolated events, it is difficult to expect the agent player to perform a satisfactory result.

5.4.2 The travelling agent for travel agency problem

In order to overcome the limitations detailed above, this subsection suggests employing a competitive algorithm to address the online decision-making problem.
An online Travelling Agent For Travel Agency (TAFTA) problem is identified, which considers a scenario where a travelling agent works for a travel agency to market its holiday plan to a group of network clients (as illustrated in Figure 5.6). Three working stages will be summarised below:

- **Early-bird registration collection**
  This is the first stage of the whole working process. A travelling agent working for a travel agency is dispatched to the network to visit potential customers. At each destination it demonstrates a pre-designed holiday plan with an early-bird registration rate $C_{f1}$ to its client. If the client is satisfied with this offer and would like to take it, the travelling agent will help to finalise the associated registration formality, and then continue its navigation to its next destination. This process would not stop until the early-bird rate is no longer valid.

- **Standard registration collection**
  At this stage, the travelling agent travels around the network performing the same task as it did during the first stage. However, since the early-bird rate is no longer valid, it would present a standard rate $C_{f2}$ to its potential customer. $C_{f2}$ is expected to be
higher than $C_{f1}$. If the customer wishes to take the standard offer, the travelling agent will enter his/her name into the touring party, and then go to visit its next network customer. This stage would continue until the end of the whole plan.

- **Online decision-making**

This stage actually takes place between the above two phases. The travel agency has to pay the third-party service provider (such as hotels or airplane companies). For these costs, two different rates might exist correspondingly: an early-bird rate $C_e$, and a standard rate $C_s$. Therefore, at this stage the travel agency needs to try to reserve some places at the lower cost $C_e$. If the number of clients registering at the first stage is $d$, while the agent estimates more clients will join in and the total number could reach $D$, reserving $D$ places with the lower cost $C_e$ might be a good strategy, because it will return additional profits $(D-d)(C_s-C_e)$ at the end. However, if the travelling agent reserved $D$ places based on such an assumption, but no clients registered during the second stage at all, unclaimed $(D-d)$ places would have to be released to the service provider, and unfortunately a fine (say, $C_c$) for each place would be payable. Therefore, the online TAFTA problem can be expressed as follows (Yang & Nguyen 2004a):

*Given the number of clients registering at the first stage $d$ and the possible maximum number at both stages $D$, and all potential costs ($C_{f1}$ and $C_{f2}$ are the early-bird and standard registration rate for the customer; $C_e$ and $C_s$ are corresponding rate for the travel agency; $C_c$ is the fine for releasing each unclaimed place), what is the best number of places that should be reserved by the travelling agent in order to maximise profits for the travel agency?*
5.4.3 Competitive analysis and proposed online solution

In literature, a similar problem -- online Snacks problem, has been studied by Ma et al (2002). In this problem, the boss of an eatery does not know how many customers will visit his/her shop each day, but has to prepare an optimal number of snacks in the morning as producing more snacks in an urgent manner incurs extra costs, although throwing away surplus snacks after each day is also a waste. A competitive Extreme Numbers Harmonic (ENH) algorithm is suggested, the best choice the eatery boss should make is \( d = Mm(p + q + 1)/Mp + m(q - 1) \), where \( m \) and \( M \) are the possible minimum and maximum number of customers, \( p = c_1/e \), \( q = c_2/e \), \( e \) is the basic cost for producing a share of snacks, \( c_1 \) the cost for those snacks that have to be thrown away, \( c_2 \) the cost for urgent supply. The achievable competitive ratio is \( \alpha = 1 + p(q - 1)(M - m)/Mp + m(q - 1) \). Although the competitive ENH algorithm is proven correct in theory, at least four drawbacks can be observed (Yang et al 2002a, 2002b). Firstly, the measurement for the competitive algorithm is chosen as the cost for producing a share of snacks. However, such cost is to some extent dependent upon the number of clients (or requests), an ever-changing production cost (or input) may lead directly to an incorrect decision on the snacks preparation at the end. Secondly, honestly speaking, the boss of an eatery may be more interested in the net profits than the online costs for his/her business. In some cases -- but not always -- profit maximisation and cost minimisation belong to the same problem, this depends on how gross return is generated. Thirdly, considering clients’ requests vary each day, the fluctuation rate may be more meaningful than the lower/upper bounds for the requests. Unfortunately, the relationship between the competitive ratio and the fluctuation rate is not addressed within their study. Finally, no simulation is provided
to prove the ENH algorithm's feasibility within either a theoretical or a practical environment.

Based on the above observations, at least four major improvements have been made in this refined TAFTA problem model. Firstly, the registration rate (or online cost) is stable at each stage, which satisfies an essential requirement in competitive analysis where an online solution must not be reliant upon its possible request sequences. In addition, the identified TAFTA problem is a generalised profit maximisation problem, which enjoys a broader solution coverage than that of the online Snacks problem. Technically, the latter could be viewed simply as a special case, where differences in profit generation are ignored. Thirdly, the relationship between competitive ratio and fluctuation rate of requests is explored. Due to the fact that the fluctuation rate may be estimable in a particular problem, the suggested algorithm's operability is therefore improved significantly. At last, an algorithm comparison is made, considering various candidate algorithms in both a theoretical and a practical situation to demonstrate the applicability of the proposed solution (Yang & Nguyen 2004a).

Within a cost minimisation problem, a competitive algorithm focused on minimising the worst-case ratio of the online cost to the optimal offline cost (as given in relation [5.1]). For the profit maximisation problem, the competitive ratio can be re-defined analogously (El-Yaniv, 1998). If for any possible request sequences, let \( P_A(\sigma) \) denote the profits of an online algorithm \( A \), and \( P_{OPT}(\sigma) \) the profits of an optimal offline algorithm \( OPT \), \( A \) is called \( \alpha \)-competitive if relation (5.2) holds:

\[
\alpha \cdot P_A(\sigma) \geq P_{OPT}(\sigma) + \beta
\]  
(5.2)
where $\alpha$ is the competitive ratio, $\beta$ is an additive constant for initialisation. When $\beta$ is equal to zero, the above equation can be re-formatted as $\frac{P_{OPT}(\sigma)}{P_A(\sigma)} \leq \alpha$. Algorithm $A$ is efficient if the competitive ratio $\alpha$ is small.

In order to conduct a competitive analysis for the given problem and work out an online solution, it is necessary to consider how profits are generated for both an online algorithm and an optimal offline algorithm. If the number of clients at the first stage is $d$ and the total number of clients at both stages is $x$, $x \geq d$, it is not difficult to see the optimal offline algorithm would reserve exact $x$ places at the decision-making stage, the final profit for this specific request is $P_{OPT} = C_{f1}d + C_{f2}(x-d) - C_e x$. If the entire request sequence is denoted as $\sigma = (x_1, x_2, \ldots, x_n)$, where $x_i$ is the actual request number on the $i$-th round, the optimal offline profits can be expressed as follows:

$$P_{OPT}(\sigma) = \sum_{i=1}^{n} [C_{f1}d + C_{f2}(x_i - d) - C_e x_i]$$

where $C_{f1}$ and $C_{f2}$ are the early-bird and standard registration rate offered to the customer, $C_e$ is the early-bird reservation fee payable to the third party provider.

Facing the same request, an online player does not know the exact number of clients $x$, so it has to estimate a possible number $m$ and reserve $m$ places at the decision-making stage. Because the actual number of clients $x$ might be less than or greater than $m$, calculating profits for the online player is a little complicated. At first, suppose $x \leq m$ meaning the places reserved by the agent are more than the total number of clients, so $(m - x)$ places have to be released, and the fine paid $C_e (m - x)$.

In that case, the profits for the online player can be expressed as $C_{f1}d + C_{f2}(x-d) - [C_e x + C_e (m-x)]$. Otherwise, if the places reserved at the decision-making stage number fewer than the total number of clients, $(x - m)$ places
have to be paid at the standard cost to the third party provider. The profits for the online player can be calculated as \( C_{f_1}d + C_{f_2}(x - d) - [C_{x}x + (C_x - C_e)(x - m)] \).

Combining the above two cases, it is not difficult to derive the profits of an online algorithm \( A \) for the same request sequence, as follows:

\[
P_A(\sigma) = \sum_{i=1}^{n} \left[ C_{f_1}d + C_{f_2}(x_i - d) \right] - \sum_{i=1}^{n} C_{x}x_i - \sum_{i=1}^{n} C_{e}(m_i - x_i) - \sum_{i=1}^{n} (C_x - C_e)(x_i - m_i)
\]

(5.4)

where \( C_x \) is the standard reservation rate for the travel agency, and \( C_e \) is the fine for returning each unclaimed place.

Therefore, the profit maximisation problem for the competitive algorithm \( A \) is trying to calculate how many places \( m \) should be reserved in order to achieve the minimised competitive \( \alpha \), which can be expressed as:

\[
\frac{P_{OPT}(\sigma)}{P_A(\sigma)}.
\]

In order to carry out an online solution facing all possible request sequences, it is necessary to consider two possible worst-case scenarios that an offline adversary might choose to minimise the profits of the online player, and maximising its competitive ratio. Two natural worst-case scenarios can be observed: \( \sigma_D = (D, D, \ldots, D) \) and \( \sigma_d = (d, d, \ldots, d) \), where \( D \) and \( d \) are respectively the maximum and minimum number of clients registered in both stages.

Let \( P_A(\sigma_D) \) and \( P_{OPT}(\sigma_D) \) denote the respective online and optimal offline profits when \( \sigma_D \) occurs, according to Equations (5.3) and (5.4), the competitive ratio can be calculated as follows:

\[
\alpha = \frac{P_{OPT}(\sigma_D)}{P_A(\sigma_D)}
\]

\[
= \frac{\sum_{i=1}^{n} [C_{f_1}d + C_{f_2}(x_i - d) - C_{x}x_i]}{\sum_{i=1}^{n} \left\{ C_{f_1}d + C_{f_2}(x_i - d) \right\} - [C_{x}x_i + (C_x - C_e)(x_i - m_i)]}
\]
After simplifying the above expression, it has:

\[
\alpha = \frac{(C_{f_1}d + C_{f_2}(D - d) - C_e D) \cdot n}{(C_{f_1}d + C_{f_2}(D - d)) - [C_e + (C_e - C_e)(D - m)] \cdot n}
\]

\[
= \frac{(C_{f_1}d + C_{f_2}D - C_{f_2}d - C_e D)}{C_e D - C_e m - C_e D + C_e m}
\]

\[
= 1 + \frac{(C_{f_1}d + C_{f_2}D - C_{f_2}d - C_e D) - (C_e D - C_e m - C_e D + C_e m)}{(C_{f_1}d + C_{f_2}D - C_{f_2}d - C_e D) - (C_e D - C_e m - C_e D + C_e m)}
\]

(5.5)

A similar reasoning process can be used in the case of \( \sigma_d \). Let \( P_A(\sigma_d) \) and \( P_{OPT}(\sigma_d) \) denote the online and optimal offline profits, the corresponding competitive ratio can be expressed in Equation (5.6):

\[
\alpha = \frac{P_{OPT}(\sigma_d)}{P_A(\sigma_d)}
\]

\[
= \sum_{i=1}^{m} \frac{(C_{f_1}d - C_e x_i)}{\sum_{i=1}^{m} [C_{f_1}d - C_e x_i - C_e (m_i - x_i)]}
\]

After simplifying the above expression, it has:

\[
\alpha = \frac{[C_{f_1}d - C_e d] \cdot n}{[(C_{f_1}d - C_e d) - (C_e m - C_e d)] \cdot n}
\]

\[
= 1 + \frac{C_e m - C_e d}{(C_{f_1}d - C_e d) - (C_e m - C_e d)}
\]

(5.6)

Because \( \sigma_D \) and \( \sigma_d \) are the two possible worst cases for the given problem, it seems that \( m \) can be chosen to help balance the maximum bounds from these two cases.

From Equations (5.5) and (5.6), it has:

\[
\frac{C_e m - C_e d}{(C_{f_1}d - C_e d) - (C_e m - C_e d)} = \frac{C_e D - C_e m - C_e D + C_e m}{(C_{f_1}d + C_{f_2}D - C_{f_2}d - C_e D) - (C_e D - C_e m - C_e D + C_e m)}
\]

and

\[
m = \frac{[(C_{f_2} - C_e)C_e + (C_{f_1} - C_e)(C_e - C_e)]Dd - [(C_{f_2} - C_{f_1})C_e]d^2}{[(C_{f_2} - C_e)C_e]D + [(C_{f_1} - C_e)(C_e - C_e) - (C_{f_2} - C_{f_1})C_e]d}
\]
In order to simplify the above expression, let \( C_{f2} - C_e = R, \ C_{f1} - C_e = S, \ C_s - C_e = Q, \ R - S = T \) and \( C_e = P \), the optimal number \( m \) that an online player should choose for the TAFTA problem can be easily derived. Theorem 1 summarises the result from the above analysis, and is thereafter termed as Travel Agency Booking (TAB) algorithm.

**Theorem 1:** For the given TAFTA problem, the optimal reserving number for the online agent to choose is:

\[
m = \frac{(RP + SQ)d - TPd^2}{RPD + (SQ - TP)d}
\]

where \( d \) is the number of clients registered at the first stage, \( D \) the maximum number of clients joining in both stages, and \( R = C_{f2} - C_e, \ P = C_e, \ S = C_{f1} - C_e, \ Q = C_s - C_e, \ T = C_{f2} - C_{f1} \).

5.4.4 Competitive ratio and its lower bound

After deriving the optimal reserving number \( m \), the corresponding competitive ratio for the TAB algorithm can be calculated by substituting the result in Equation (5.7) into Equation (5.5) or (5.6). The resulting competitive ratio can be summarised as below:

**Theorem 2:** For the competitive TAB algorithm given in Theorem 1, the competitive ratio is:

\[
\alpha = 1 + \frac{PQ(D - d)}{(RP - PQ)D + (SQ - TP + PQ)d}
\]

or it can be said that TAB is a \left( 1 + \frac{PQ(D - d)}{(RP - PQ)D + (SQ - TP + PQ)d} \right) -competitive algorithm.
This means the profits of an optimal offline algorithm are at most $\alpha$ times those of the online TAB algorithm, in other words, the latter are at least $1/\alpha$ those of the former for the same requests. In fact, not all online competitive algorithms are optimal, so in order to demonstrate the proposed TAB algorithm is an optimal one, it has to prove that 
\[ \left(1 + \frac{PQ(D-d)}{(RP-PQ)D + (SQ-TP+PQ)d}\right) \] is the lower bound for the achievable competitive ratio.

**Proof.** This problem can be translated as follows: 'if an online player chose a different reserving number from the value of $m$ in Equation (5.7), namely, $m' \neq \frac{(RP + SQ)Dd - TPd^2}{RPD + (SQ - TP)d}$, the competitive ratio would get worse'. A two-step proof follows.

Firstly, assuming $m' < m$, it needs to be proven that if the offline adversary chose request sequence $\sigma_D$, the suggested competitive ratio cannot be achieved. According to Formula (5.5), it has:
\[
\alpha' = 1 + \frac{(C_s - C_e)(D - m')}{[(C_{f1} - C_e)D - (C_{f2} - C_{f1})d] - [(C_s - C_e)(D - m')]} = 1 + \frac{Q(D - m')}{(RD - Td) - Q(D - m')} \tag{5.9}
\]
From this, it can be seen that if $m' < m$ then $\alpha' > \alpha$.

Secondly, assuming $m' > m$, it needs to be proven that if the offline adversary chose the request sequence $\sigma_d$, the competitive ratio in Equation (5.8) cannot be achieved either. According to Formula (5.6), it has:
\[
\alpha' = 1 + \frac{C_e(m' - C_e d)}{(C_{f1}d - C_e d) - (C_e m - C_e d)} = 1 + \frac{P(m' - d)}{Sd + Pd - Pm} \tag{5.10}
\]
From the above expression it is observed that if \( m' > m \), then \( \alpha' > \alpha \). Therefore, the proof is completed. The whole analysis can be concluded in Theorem 3 as follows:

**Theorem 3:** In the given TAFTA problem, \( 1 + \frac{PQ(D - d)}{(RP - QP)D + (SQ - TP + QP)d} \) is the lower bound of the achievable competitive ratio.

Further analysis of the algorithm given in Theorem 1 and 2 can generate a series of meaningful corollaries that will be helpful in understanding how the proposed solution can be simplified in particular cases. Corollaries 1 and 2 focus on special values of registration rates \( C_{f1} \) and \( C_{f2} \), while Corollary 3 presents a specific solution related to the fluctuation rate. Proofs of these corollaries are similar to what was expressed for Theorem 2, as they are merely special cases.

**Corollary 1:** In the online TAFTA problem, if \( C_{f2} = C_{f1} \), that is, \( T = 0 \) or \( R = S \), the optimal reservation number for the online agent to choose will be:

\[
m = \frac{(P + Q)Dd}{PD + Qd}
\]

(5.11)

and the corresponding competitive ratio will be:

\[
\alpha = 1 + \frac{PQ(D - d)}{(RP - PQ)d + (SQ + PQ)d}
\]

(5.12)

The significance of assuming \( C_{f2} = C_{f1} \) is that the travel agency presents the same offer to the potential clients at both early-bird and standard registration stages, although it has to pay different rates to the third-party service provider for booking. In fact, \( C_{f1} \) and \( C_{f2} \) represent different profit-generation methods for the travel agency, so when \( C_{f2} = C_{f1} \), such difference is eliminated, and the profit maximisation problem
turns out to be a simplified cost minimisation problem. As mentioned in the previous section, the online Snacks problem initialised by Ma et al. (2002) simply reflected this special case, where $C_{f2} = C_{f1}$.

**Corollary 2:** In the online TAFTA problem, if $C_{f2} - C_{f1} = C_s - C_e$, that is, $T = Q$, the optimal reservation number for the online agent to choose will be:

$$m = \left[ \frac{(RPD - PQd) + SQD}{(RPD - PQd) - SQd} \right] \cdot d$$  \hspace{1cm} (5.13)

and the corresponding competitive ratio will be:

$$\alpha = 1 + \frac{PQ(D - d)}{(RP - PQ)D + SQd}$$  \hspace{1cm} (5.14)

Recall that $C_e$ and $C_{f1}$ represent the early-bird reservation rates for the customer and the travel agency, while $C_s$ and $C_{f2}$ are the standard rates for each of them. The significance of this corollary is that the travel agency completely shifts off the difference on these two rates introduced by the third-party service provider to its clients. Therefore, with this corollary in place, the derived solution given in Theorem 1 would be suitable for more generalised cases, where $C_{f2} - C_{f1} \neq C_s - C_e$.

**Corollary 3:** If the fluctuation rate is denoted as $\phi = D/d$, for the given TAFTA problem, the optimal reservation number for the online agent to choose will depend on $\phi$ when other parameters remain constant.

$$m = \frac{(RP + SQ)\phi - TP}{RP\phi + (SQ - TP)} \cdot d$$  \hspace{1cm} (5.15)
and the corresponding competitive ratio will depend on $\phi$ too. When $\phi$ gets smaller, $\alpha$ gets better.

$$\alpha = 1 + \frac{PQ(\phi - 1)}{(RP - PQ)\phi + (SQ + PQ)}$$  \hspace{1cm}(5.16)

From this corollary, it can be noted that the number of clients registering at the first stage $d$ is known before the travelling agent has to make a decision, and all other parameters except the fluctuation rate $\phi$ is constant for the TAFTA problem. Thus, the optimal reservation number and corresponding competitive ratio will depend on $\phi$ only. If the fluctuation rate were estimable in certain applications, it would become much easier for the travelling agent to make a dynamic and optimal decision through its online observation.

### 5.4.5 Experimental results and algorithm comparison

In order to evaluate the effectiveness of the proposed TAB algorithm, a series of simulation experiments are conducted which can be classified into two types. The first set deals with theoretical issues, and the second with practical considerations. Within a theoretical environment, an ADOF adversary is employed, which is the strongest one among all kinds of adversaries (namely, ADON, ADOF and OBL) (Ben-David et al., 1994). Because there is no constraint applied on its request sequence selection, the ADOF adversary could select the worst possible inputs to maximise its rival's competitive ratios based on its knowledge of their individual strategies. However, within a practical environment, the offline adversary is forced to generate request sequences randomly, and all candidate algorithms will face the same sequences equally. Three other algorithms are explicitly introduced for comparison. These are
the average algorithm, the follower algorithm, and the random algorithm. Each will compete with the ADOF adversary one by one within a theoretical environment, but deal with the same randomised requests within the practical environment. A detailed explanation of these three algorithms will be given below:

- **Average algorithm**
  For the given TAFTA problem, the reservation number \( m' \) that an average algorithm will choose each time is the arithmetic mean of \( d \) and \( D \), namely, \( m' = \frac{(d + D)}{2} \), where \( d \) is the number of clients registering at the first stage, and \( D \) is the possible maximum in total.

- **Follower algorithm**
  The follower algorithm chooses the same number as what happened in its previous time, namely, \( m'_i = x_{i-1} \).

- **Random algorithm**
  The random algorithm chooses \( m' \) ranging from \( d \) to \( D \) randomly each time.

In order to quantitatively evaluate the performance of each suggested algorithm, it is necessary to specifically define all parameters of the identified TAFTA problem. For now, there is no real data available from exactly the same scenario, therefore, a group of simulation data has been selected with the aim of approaching the real data as realistically as possible. The selected simulation data are as follows: the number of clients registering at the first stage, \( d = 50 \), the maximum number of clients registered at both stages, \( D = 80 \). The early-bird and standard registration rates quoted to the
customer, $C_{f_1} = 1600, C_{f_2} = 1800$. The early-bird and standard reservation rates for the travel agency to pay its third party service provider, $C_e = 1200, C_s = 1500$. The fine for releasing each unclaimed place $C_c = 100$. Performance of each selected algorithm against the ADOF adversary is given as follows:

- **Online TAB algorithm**
  
  Based on Theorems 1 and 2 and the above simulation parameters, the optimal reservation number that an online TAB algorithm would choose is $m = 68$. In order to maximise the competitive ratio of the online player, the ADOF adversary selects the request sequence $\sigma_d = d, d, \ldots, d$ or $\sigma_B = D, D, \ldots, D$. As proved in the previous subsection, the TAB algorithm could achieve an optimal competitive ratio $\alpha = 1.105$.

- **Average algorithm**

  Given the same simulation parameters, the average algorithm chooses $m' = 65$ as the optimal reservation number. Facing the average strategy, the ADOF adversary chooses $\sigma_B = (D, D, \ldots, D)$ as the inputs, as it leads to a worse competitive ratio $\alpha = 1.134$.

- **Follower algorithm**

  The follower algorithm chooses an average number for the first time and subsequently follows the number that appeared in the previous time. To overcome this algorithm, the ADOF adversary introduces an adaptive strategy. It chooses $x = D$ at the first time and choose $x = d$ at the second time, and then $\sigma = D, d, D, d, \ldots$. The competitive ratio $\alpha$ will then get worse. Table 5.3 shows the simulation results for 100 trials:
Table 5.3: Simulation results of the follower algorithm facing ADOF adversary

- Random algorithm

Facing the random algorithm (which chooses a random number each time), the ADOF adversary will employ a trickier strategy. If \( m' > m = 68 \), it will choose \( x = d \), otherwise, it will choose \( x = D \). The competitive ratio \( \alpha \) will also get worse. Table 5.4 shows the simulation results after 100 trials:

<table>
<thead>
<tr>
<th>Number of trials</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1.237</td>
<td>1.248</td>
<td>1.253</td>
<td>1.255</td>
<td>1.256</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>1.257</td>
<td>1.257</td>
<td>1.258</td>
<td>1.258</td>
<td>1.258</td>
</tr>
</tbody>
</table>

Table 5.4: Simulation results of the random algorithm facing ADOF adversary

From the above simulation, it can be noted that the ADOF enjoys a distinct advantage in that it might choose the request sequence after the online player has made a decision, and might be adapted to each individual player. Although it is customary in the literature to conduct competitive analysis on this basis, it can be argued that the comparison is biased in favour of the offline adversary. Therefore, it can be reasonable to ask whether these candidate algorithms may in fact perform better within a real e-business environment. Keeping all the parameters chosen the same as given above, within a practical environment the request sequences are generated randomly, and all candidate algorithms would be treated equally. Corresponding simulation results of 100 trials are given in Table 5.5 and Figure 5.7:
<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Online TAB</th>
<th>Average</th>
<th>Random</th>
<th>Follower</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 trials</td>
<td>1.089</td>
<td>1.069</td>
<td>1.076</td>
<td>1.104</td>
</tr>
<tr>
<td>20</td>
<td>1.052</td>
<td>1.056</td>
<td>1.095</td>
<td>1.092</td>
</tr>
<tr>
<td>30</td>
<td>1.050</td>
<td>1.055</td>
<td>1.093</td>
<td>1.083</td>
</tr>
<tr>
<td>40</td>
<td>1.048</td>
<td>1.058</td>
<td>1.090</td>
<td>1.085</td>
</tr>
<tr>
<td>50</td>
<td>1.046</td>
<td>1.053</td>
<td>1.090</td>
<td>1.087</td>
</tr>
<tr>
<td>60</td>
<td>1.043</td>
<td>1.049</td>
<td>1.089</td>
<td>1.088</td>
</tr>
<tr>
<td>70</td>
<td>1.047</td>
<td>1.054</td>
<td>1.081</td>
<td>1.084</td>
</tr>
<tr>
<td>80</td>
<td>1.045</td>
<td>1.052</td>
<td>1.081</td>
<td>1.085</td>
</tr>
<tr>
<td>90</td>
<td>1.044</td>
<td>1.052</td>
<td>1.077</td>
<td>1.085</td>
</tr>
<tr>
<td>100</td>
<td>1.045</td>
<td>1.053</td>
<td>1.078</td>
<td>1.084</td>
</tr>
</tbody>
</table>

Table 5.5: Simulation results of four algorithms within a practical environment

![Figure 5.7: Performance comparison of algorithms within a practical environment](image)

From Table 5.5 and Figure 5.7, it can be seen that the performance of all candidate algorithms becomes better than that within the theoretical environment. The online TAB algorithm works better than the other three algorithms, except for the 10-trial mark. The follower's performance is similar to the random algorithm's result. This happened because the request sequence is generated randomly, therefore, the follower simply adopts the random number that occurred in its previous time.

The aim of the last simulation is to test the conclusion given in Corollary 3, where the relationship between the optimal reservation number $m$ and the fluctuation rate $\phi = D/d$ is examined. The number of clients $d$ is observable before the online
player makes a decision, and the fluctuation rate may be estimated from either experience or historical data. Given knowledge about $d$ and $\phi$, the operability of the suggested algorithm would be improved. In the following simulation, the value of $d$ fluctuates within a range $d \in [30,50]$, and the fluctuation rate $\phi = D/d = 1.6$ is fixed.

Simulation results are shown in Table 5.6 and Figure 5.8:

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Online TAB</th>
<th>Average</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 trials</td>
<td>1.043</td>
<td>1.031</td>
<td>1.123</td>
</tr>
<tr>
<td>20</td>
<td>1.070</td>
<td>1.079</td>
<td>1.084</td>
</tr>
<tr>
<td>30</td>
<td>1.052</td>
<td>1.063</td>
<td>1.093</td>
</tr>
<tr>
<td>40</td>
<td>1.054</td>
<td>1.064</td>
<td>1.087</td>
</tr>
<tr>
<td>50</td>
<td>1.050</td>
<td>1.060</td>
<td>1.086</td>
</tr>
<tr>
<td>60</td>
<td>1.047</td>
<td>1.056</td>
<td>1.090</td>
</tr>
<tr>
<td>70</td>
<td>1.046</td>
<td>1.055</td>
<td>1.092</td>
</tr>
<tr>
<td>80</td>
<td>1.045</td>
<td>1.055</td>
<td>1.090</td>
</tr>
<tr>
<td>90</td>
<td>1.046</td>
<td>1.054</td>
<td>1.088</td>
</tr>
<tr>
<td>100</td>
<td>1.045</td>
<td>1.054</td>
<td>1.085</td>
</tr>
</tbody>
</table>

Table 5.6: Simulation results of three algorithms when $\phi$ is fixed

![Performance comparison of three algorithms when $\phi$ is fixed](image)

Figure 5.8: Performance comparison of three algorithms when $\phi$ is fixed

In the above simulation, request sequences are generated randomly. The follower algorithm is no longer included, as it might be out of range and frequently become meaningless. Performances of the algorithms within this practical environment are
generally better than that within the theoretical environment. Overall, the online TAB algorithm works better than other two candidates, especially within a long-term run.

5.4.6 Applicability and limitations for the practice

In order to indicate the applicability and limitations of this ongoing research, it is necessary to give further discussion about the practice behind the scene. Technically, the identified online TAFTA problem and online Snacks problem are not isolated cases, but demonstrate a class of similar issues with a set of common features in reality, which can be summarised as follows:

- The commodity in consideration normally has a short shelf-life, and associated business activity will take place periodically. For example, in the online TAFTA problem, the commodity is the holiday plan, which is only valid for a short period of time, with a deadline for each touring party. In the online Snacks problem, the commodity is the snacks, which cannot generate profit after each day.

- For each period of business transaction, the customer requests may vary within a range, but the exact request quantity is not accessible beforehand. For example, customer requests in both given problems fluctuate among a minimum and maximum limit, which forms a lower and upper bound for each request sequence.

- An online decision-maker is required to prepare a number of service units before each transaction for either cost-saving purposes (e.g., in the TAFTA problem) or service-quality-improving considerations (e.g., in the Snacks problem). However, no matter whether the initial selection is greater than or less than the customers' real requests, an increased online cost will occur, which may come from different service rates (e.g., in the TAFTA problem) or unavoidable wastes (e.g., in the Snacks problem), or both.
Minimised online cost or maximised service profit is the principal objective for an online decision-maker to achieve in case of all possible requests.

From the above summary, it can be said that the different service rates present a basic reason for the changeability of the online costs and essentiality of a competitive solution. Such a scenario is not rare, for example, if someone wishes to distribute some documents at an exhibition, he/she would like to prepare some copies of the document in advance. Although there might be a photocopying machine available at the exhibition hall, this would be much more expensive than producing them at home. If that advertising material was for sale and valid only for a short period of time, this document-preparation problem would have a kind of online fashion. A careless decision-maker may simply choose the average number of the possible requests as a solution. Based on the experiment conducted in previous sections, it must be acknowledged that the performance of the average algorithm is not too bad. However, when considering the business activity that would happen periodically, a well-designed competitive algorithm would be more appropriate. Taking the online TAB algorithm as an example, the optimal number an online player should choose is not the arithmetic mean of the potential minimum and maximum number of requests, but taking into account various related parameters comprehensively as its balancing point.

From the conducted simulations, a number of limitations of the suggested algorithm can also be noticed. At first, facing the randomised requests, the competitive ratio of the average algorithm or the random algorithm is sometimes better than the online TAB algorithm. For every 100 trials, the highest record observable shows that $\alpha_{\text{average}} < \alpha_{\text{TAB}} = 9$ and $\alpha_{\text{random}} < \alpha_{\text{TAB}} = 8$. One possible reason for this phenomenon is that the request sequence is chosen randomly, which might be
equally approaching any number selected by the candidate algorithms. But the competitive TAB algorithm is still a better choice facing all possible request sequences within a long-term run.

The second issue that needs to be discussed here is whether the 'practical' environment can prove an algorithm's applicability against all possible situations. Actually, it is controversial and certainly debatable as to how to model 'reality' and which one best models 'reality' (Ben-David & Borodin, 1994; Irani & Ramanathan, 1998). Because even within a 'practical' environment where all requests are generated randomly, there is still a hidden assumption -- that is, requests within the given range are equally selected. That is why all performance curves of candidate algorithms become smooth and almost stable at the end. However, before any agreement on this controversial problem can be reached or any new measurement proved to be more accurate, the ADOF adversary-based competitive analysis remains a meaningful approach to measuring the proposed online algorithms.

Another observable drawback so far is that the relationship between $C_{f1}$, $C_{f2}$ and $d$ is not studied, namely, whether introducing different profit generation rates could attract more potential clients to join and bring more total return for the agency. It is unlikely to be a problem that can be addressed by competitive algorithms, but studying such a relationship is absolutely necessary when considering more complex business models within the real world.

5.5 Chapter summary

Adaptability is essential for a mobile agent working within a virtual business situation. Two common techniques are frequently employed for this purpose: one is the SC technology which is suitable to address problems with imprecision
information, and the other is the traditional business rule which is good at dealing with complex issues within the physical world. Although these techniques employ different methods to obtain and store knowledge, they still lack the flexibility to dynamically adjust a mobile agent's behaviour within an online environment, where future request patterns are not accessible, however, they do affect the performance of a non-rigorous problem-solving strategy in a serious way.

This chapter proposes to use competitive algorithms to overcome these limitations. Two kinds of applications are considered suitable for online solutions: one is agent-based task scheduling, the other is agent-mediated decision-making. For the former, a RES-supported task-scheduling strategy is examined which aims to deploy multiple agents working within a system with minimised communication costs. For the latter, an online TAFTA problem with an enhanced TAB algorithm is identified, which proves to improve an agent's adaptability, based on a guided decision-making scheme in term of an optimised competitive ratio.

However, all related problem-solving techniques have their own merits and drawbacks. It is difficult to attain satisfactory results when trying to employ a single technology to address all problems under uncertainty, which is still true for online algorithm implementation. Therefore, hybrid use of various methods (online and offline, soft-computing and hard-computing, business rules and technical constraints), and making use of various techniques' mutual complementarity to improve associated systems' robustness is not only practical but also desirable.

In the coming chapter, the main focus will be on the agent security issues, and risks associated with an agent being attacked by malicious hosts will be discussed in detail.
Chapter 6

Mobile Agent Protection in an Internet-Based E-Business Environment

6.1 Chapter introduction

In previous chapters, a number of mobile agent-based systems have been selected in order to demonstrate the feasibility of suggested models, patterns and algorithms. However, as most of the systems or prototypes are still in the experimental stage, it is difficult to find any successful example that employs mobile agents in commercialised applications. One possible reason is the security concern. On the one hand, an agent host is a network server, which is expected to continuously provide services to all visiting agents. Without proper protection mechanisms, it is an easy target for various malicious entities over the network, thereby affecting the whole system's stability and reliability. On the other hand, a mobile agent itself is vulnerable when it leaves its home site and works at a remote and possibly malicious platform. Without proper security arrangements, it is difficult to convince people employing
mobile agents to conduct transactions on their behalf. In this chapter potential threats and malicious attacks within a mobile agent-mediated e-business environment will be identified, promising solutions and successful approaches for agent and agent host protection will be investigated, and finally a collection of agent protection structures and associated protocols will be suggested.

6.2 Security issues within mobile agent systems

Mobile agent protection versus malicious attacks is an endless game. While you can design various regulations and techniques to protect agents, your opponents (malicious designers) will always try to crack your protection and outsmart you. Therefore, before agent protection technologies are investigated, it is necessary to identify various potential threats that may occur within this specific area.

6.2.1 Potential threats within a mobile agent-mediated e-shopping system

There are some other reviews available in literature regarding security threats and malicious attacks within this field (Jansen & Karygiannis, 1999; Ng, 2000). However, few of these put the analysis in a mobile agent-mediated e-business context. This subsection seeks to analyse the same question with the help of a simulated mobile agent shopping system to improve its relevance, as given in Figure 6.1. In this proposed system, a mobile shopping agent starts its journey from its home to visit two virtual markets in sequence: one is honest, and the other is malicious. Another host within the system represents a bank or financial institution (which is not for agent shopping, but needed when the agent owner wishes to pay online). A malicious third
party attacker is also included, which represents any other malicious entity active on the network.

In total, there are five kinds of security threats within the simulated system: (1) malicious host attacks host(s), (2) malicious agent attacks host(s), (3) malicious agent attacks agent(s), (4) malicious host attacks agent(s), and (5) malicious third party attacks agent-and-host. With the first two threats, the agent host is the major target, because it is supposed to facilitate multiple mobile agents’ execution. If the host were under attack, every agent working in this platform would be affected. Moreover, all confidential information stored within this host would become unsafe too. The mobile agent is another vulnerable target: potential threats might come from either a malicious agent or a malicious host. Both of these may seek the private information carried by the agent, because some of the data must be accessible to the underlying platform when the agent tries to resume operations there. The last type of threat is from a malicious third party. This kind of attacker does not need to invade an agent host directly, as its target is mainly the agent-to-host or host-to-host communication. For example, a travelling agent may need to contact its home site for consultation, or

Figure 6.1: Five kinds of potential threats within a mobile agent-based system
an agent host may need to communicate with the bank for online payment. Contents of the messages are just the attacker's targets.

6.2.2 Taxonomy of attacks towards sensitive information

The security threats mentioned above have been roughly analysed from the viewpoint of the victims. This subsection seeks to examine the same problem from the viewpoint of the attackers. Two distinct categories can be classified in general: passive attacks and active attacks (Lange & Oshima, 1998; Jansen & Karygiannis, 1999). The major criterion for such a classification is whether or not the attacker would modify the target information. Most active attackers may alter the confidential information aggressively, whereas passive attackers do not change the captured data. A brief summary of various attacks is given in Table 6.1. A further discussion will follow.

<table>
<thead>
<tr>
<th>Attack type</th>
<th>Sub-category</th>
<th>Potential attacker(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive attacks</td>
<td>Eavesdropping</td>
<td>A third party</td>
</tr>
<tr>
<td></td>
<td>Traffic analysis</td>
<td>A third party</td>
</tr>
<tr>
<td>Active attacks</td>
<td>Masquerade</td>
<td>An agent or an agent host</td>
</tr>
<tr>
<td></td>
<td>Illegal access</td>
<td>An agent or an agent host</td>
</tr>
<tr>
<td></td>
<td>Alteration</td>
<td>An agent, an agent host or a third party</td>
</tr>
<tr>
<td></td>
<td>Repudiation</td>
<td>An agent or an agent host</td>
</tr>
<tr>
<td></td>
<td>Copy and replay</td>
<td>A third party or an agent host</td>
</tr>
<tr>
<td></td>
<td>Denial of service</td>
<td>An agent host</td>
</tr>
</tbody>
</table>

Table 6.1: A brief summary of potential attacks within a mobile agent system
(Based on Jansen & Karygiannis, 1999)

**Eavesdropping**

An eavesdropper tries to capture and analyse the communication message between an agent and a host. Any plain text transmitted over the network may be directly under such attack. For example, if the shopping agent sent a message back home to report its collected offers in plain text, the eavesdropper can capture this message and read it without any trouble. Clearly, the best solution to prevent this happening is the use of a
secure network for communication and employing cryptography technique to encrypt the message. However, message encryption takes time, thus mobile agents may not bother to apply cryptography techniques for unimportant messaging. This is why eavesdropping attacks are always successful, although technically simple.

**Traffic analysis**

Cryptography techniques are often employed when the message is confidential, for example, when an agent owner wishes to communicate with a bank to arrange its online payment. In this case, a simple eavesdropping attack will not be valid, because the eavesdropper can only receive a string of ciphered text. However, even if the message were encrypted, another kind of passive attack called traffic analysis is still effective. A perpetrator needs only to analyse the communication pattern between a sender and a receiver, as the frequency of the message-flow can always reveal some useful information about the communication participants. For example, a higher message-flow frequency indicates the host is very active on the network. Therefore, attacking such a host may cause more damage for an attacker than attacking an inactive one.

**Masquerade**

In contrast to the two passive attacks detailed above, the masquerade is a truly active one. In this attack, the malicious entity pretends to be a different entity in order to perform illegal actions. It may disguise itself as another agent to visit a system where only authorised agents are entitled to enter. If the perpetrator were an agent host, the consequences would be more destructive. For example, if the host illustrated in Figure 6.1 (representing Market 2) masqueraded itself as another honest market of good
reputation, it provides no products or services, but cheats visiting agents and steals their private information. Consequently, all agents visiting this host would become victims. Moreover, the image of that honest agent host might also be destroyed.

Illegal access

Illegal access is another kind of active attack, which shares some common features with masquerading. A malicious agent claims to be another agent and illegally accesses information it is not allowed to use. The attacker itself may be a legal user of the system, but the information it tries to access is definitely beyond its granted permission. For example, an agent invading a bank system may have a legal account in the bank, but tries to access the information that only a system manager is entitled to operate. Were it to successfully log into the system manager's account and be granted super-user permission, all confidential information maintained by the manager might be unsafe.

Alteration

Alteration attack refers to illegal modification. In the scenario discussed above, if the malicious agent not only accessed the system manager's account but also illegally modified some confidential data, that would be an obvious alteration attack. Apart from a malicious agent, both a malicious third party and a malicious host can launch such an attack. For the ill-intent third party, an apparent target is the message transferred over the network. It may tamper with the data and resend it for illegal purposes. For the malicious host, the confidential information carried by visiting agents is a natural target, because the agent has to expose parts of its code and data when it tries to make a system call to the underlying platform.
Copy and replay

In addition to the traffic analysis attack, copy and replay attack is another example that is effective against ciphered texts. The interceptor does not need to decode the encrypted data, but simply make a copy of a captured message, and re-use it for its own benefit. Taking the simulated system as an example, a mobile agent signs a cheque to pay for its shopping items. The cheque encrypted by the agent's signature is safe against any misbehaviour. However, the malicious market operator can make a copy of this cheque and re-use it more than once. If the receiver has no way of distinguishing the original cheque and its copy, it has to bear the losses. Technically, even a complete agent can be copied and re-transmitted.

Repudiation

Repudiation attack takes place when an agent or a host deliberately denies a previous communication exchange. For example, a mobile shopping agent purchases some goods from the market, paying with a signed cheque and waiting for product delivery. Unfortunately, the market operator intentionally denies having received the cheque. In order to help a referee solve possible disputes, either the agent or the agent host needs to provide certain non-repudiable evidence to safeguard its rights or prove its innocence. Sometimes an unintentional programming error or even different understanding of an operational rule may raise similar disagreements.

Denial of service

Denial of service attack is also known as resource exhaustion. A common form is that the attacker dispatches lots of agents to the target machine to irresponsibly consume the resources such as CPU time, memory, or network bandwidth, until the target host
cannot function properly. For a service provider hosting many other agents from the same platform, all the agents would be affected if the resource were exhausted. Similar attacks can be found within many traditional client-server systems, the difference lies in that the mobile agent itself can be directly used as a resources-consuming entity, which makes this attack much simpler.

6.2.3 Security requirements for both agents and agent hosts

Based on the above discussion, it is clear that the best protection scheme is effectively preventing these attacks from taking place. If a fully preventive solution could not be achieved, at least some detection mechanism should be in place to deter a possible perpetrator from performing illegal actions. Before exploring various prevention and detection schemes, the setting up of some basic security requirements is another essential step, which can be used to measure and test the feasibility and effectiveness of a suggested solution. Five major security requirements have been broadly identified (Tardo & Valente, 1996; Jansen & Karygiannis, 1999). These are confidentiality, integrity, accountability, availability, and anonymity. A brief summary will be given below.

Confidentiality

In the proposed e-shopping system, possible confidential information includes the communication message between the agent host and the bank, the private information stored in the host machine, and the collected data carried by the mobile shopping agent. In fact, the first two cases can also be found within many traditional client-server systems, and the cryptograph technique has already provided a reasonably satisfactory solution. Even an entire agent can be encrypted and directly transmitted
over the network. However, keeping the private data within a mobile agent secret from a visited host is a truly difficult issue, because when a shopping agent conduct its business activity at a remote site, most data and computing functions have to be in a readable and executable form.

**Integrity**

When a mobile agent is travelling around the network to perform business activities remotely, it is basically out of control from its home site. Therefore, two entities will have special concern for an agent's integrity: the agent owner and the visited host. The agent owner is concerned that the private data embedded in the agent's memory will not be tampered with. Although illegal alteration cannot be fully prevented so far, detecting such violation is still meaningful to the agent owner. At least it can tell whether the information collected by the agent along the way is reliable. The visited agent host is concerned as to whether the agent is carrying any harmful code to its computational environment. Therefore, a remote host is required to check the integrity of a visiting agent before running the agent's program from its own platform.

**Accountability**

Accountability means that any action being made on an agent or an agent host must be accountable, the primary aim being for non-repudiation checking. When a mobile agent visits a potentially malicious host, an accountability mechanism can help the agent owner detect any illegal actions that have occurred on its agent, and trace the possible wrongdoers thereafter. From an agent host perspective, an accountability facility provides a complete record about what has taken place from its platform.
Whenever a disagreement between an agent and a host occurs, all audit trails can be used as convincing proof for dispute settlement.

**Availability**

Availability requires an agent host to keep its services available all the time, which not only includes its various computing resources (e.g., CPU time and network bandwidth), but also covers its quality of service (e.g., speed and reliability). Denial of service attack might be a major threat for satisfying this requirement. Fortunately, many ready-made techniques designed for traditional client-server applications such as concurrency control, exclusive access, and deadlock management remain valid for the mobile agent system. In addition, fault-tolerance facilities are also applicable for ensuring service availability in case system failures are encountered.

**Anonymity**

This requirement can be considered in two senses. In one sense, a mobile agent might wish to use anonymous identity to access an agent host's service to enjoy transaction freedom. In another sense, a travelling agent might hope to keep the identities of its visited platforms anonymous to many other hosts en route for privacy. Unfortunately, facilitating anonymity always meets with operational difficulties. For the former scenario, anonymous transactions may not conform to an agent host's interests because it has the requirement for accountability. For the latter, it is not easy to convince a host to accept an arbitrary agent from unknown origin without any precautions. Therefore, balancing an agent expectation on privacy and a host requirement on its security concern is a complex task for anonymity fulfilment.
6.2.4 Traditional and agent-oriented solutions for agent host protection

Agent host security is actually not a new issue, because a host is simply a network service provider. Therefore, most strategies designed for network server protection might be directly applied for an agent host. Security issues suitable for this type of solution are various system resources such as computing environment (Wahbe et al., 1993), interpretation code (Ousterhout et al., 1997), and CPU time (Lal & Pandy, 1999). However, the structure of a visiting agent is much more complex than that of a migrating procedure, which might carry out more complicated or dangerous activity at a visited platform. Therefore, when trying to protect a host against malicious agents, a set of agent-oriented characteristics have to be seriously considered, for example, a mobile agent identity and authority (Karjoth et al., 1997), its path history (Ordille, 1996), and its code safety (Lee & Necula, 1997). Table 6.2 gives a list of existing approaches specific to this area:

<table>
<thead>
<tr>
<th>Protection types</th>
<th>Suggested solutions</th>
<th>Protection targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>Software-based fault isolation</td>
<td>Computing environment</td>
</tr>
<tr>
<td></td>
<td>Safe code interpretation</td>
<td>Interpretation code</td>
</tr>
<tr>
<td></td>
<td>Constraint-based CPU scheduling</td>
<td>Sharing resources</td>
</tr>
<tr>
<td>Agent-oriented</td>
<td>Agent authentication and authorisation</td>
<td>Agent identity and authority</td>
</tr>
<tr>
<td></td>
<td>Path history verification</td>
<td>Mobile agent itinerary</td>
</tr>
<tr>
<td></td>
<td>Proof carrying code</td>
<td>Mobile agent code</td>
</tr>
</tbody>
</table>

Table 6.2: A brief summary of potential strategies for agent host protection

**Software-based fault isolation**

Protecting computation environment against illegal access is an important task for a system designer. Technically, instructions enabling processes of jumping to or storing in an address which cannot be statically verified can be regarded as unsafe, such as jumping through registers or using a register to hold procedure-returned addresses. Fault isolation is a common approach for preventing such happenings, where each
software module is placed into its own address space. Modules loaded into a separate space can only call each other through an enhanced procedural-calling interface. In some senses, fault isolation technique blocks the capability of malicious code to corrupt contents in other computing entities through unsafe instructions (Anderson et al, 1991). Unfortunately, a hardware-based fault isolation approach always meets a higher performance cost, especially for those tightly coupled modules, because cross-module communication through hardware boundaries is expensive. In order to overcome this limitation, Wahbe et al (1993) presented a software-enforced approach to provide the same functionality. Two associated techniques are introduced: segment matching and address sandboxing. Initially, an application's virtual address space is divided into several segments, each fault domain contains two segments (one for the module's code and another for its static data) and each segment has a unique identifier. A legal module's object code can only jump to targets within its own code segment and write to addresses within its own data segment, and all must share the same segment identifier. The segment-matching technique inserts a portion of checking code before each unsafe instruction. With the help of this checking code, it is easy to determine whether the target address generated by the offending instructions has the correct segment identifier, as a system error will be triggered if the check fails. In order to further reduce runtime overhead of segment matching operation, an address sandboxing technique has been suggested. A sandboxing address sets each segment identifier to a correct value, which is not for catching the source of faults, but for preventing those distrusted instructions from being tampered with by other fault domains. In summary, software-based fault isolation provides a feasible solution to an agent host's computing environment protection. For tight-coupled modules where cross-domain communication is frequent, this approach could achieve a good
performance. However, if the cross-domain communication was not heavy, normal execution time would be increased, and this solution would not be very efficient.

**Safe code interpretation**

Code interpretation is another essential operation for agent host protection, because many mobile codes are written in interpreted languages such as Obliq, Telescript and Tcl. An investigation about safe code interpretation has been given in subsection 2.4.5, where a safe-Tcl model with twin interpreters was discussed (Ousterhout *et al.*, 1997). Briefly speaking, the twin-interpreter model has two advantages. First, untrusted code is separated from trusted code, and different interpretation environments may have different security properties. Second, security policy is not necessarily fixed for all foreign codes, but may vary based on their different owners and origins. Unfortunately, slow performance is an obvious drawback of the twin-interpreter's implementation.

**Constraint-based CPU scheduling**

CPU time is another important system resource that needs to be protected. An incoming client may ask for CPU time as much as possible in order to complete its task quickly. However, a server could not simply allocate its limited CPU resource to all clients as they requested. System security and quality of service are two major concerns. Lal and Pandy (1999) suggested a constraint-based CPU scheduling scheme to improve a server's control over this issue, where both the client and the server are allowed to specify a set of constraints for resource usage. The requirements from a client might include a lower bound (the expected minimum amount of CPU in a schedulable unit), a share (the expected relative amounts of CPU in each schedulable
unit), and a deadline (the expected completion time). The server tries to satisfy all its clients' needs, but has different strategies for allocating its resource. Requests from different clients are initially classified into different groups, each group can only share a specific proportion of the total CPU resource. Within each group, the server imposes another set of constraints including priority (relative importance), upper bound (the maximum amount of CPU for each program in each schedulable unit), and lifetime (the total amount of CPU time over all execution of a specific program). Based on these constraints, three associated algorithms are suggested. The first is enforcing upper bound and lifetime constraints, the second is enforcing share and priority constraints, and the last is enforcing real-time deadline-based requirement. If the requirements from a client were in conflict with the policy of the server, a composition policy in favour of the server would be employed to settle the disputes. However, such an arrangement presents a drawback as well: the clients' group and their priorities are fixed at the designing time and not changeable thereafter, which lacks flexibility within real operations. Some other CPU resource scheduling algorithms can be found in literature, such as the Lottery and Stride algorithm offered by Waldspurger and Weihl (1995), the charge-based proportional algorithm suggested by Maheshwari (1995), and the market-oriented algorithm presented by Bredin et al (1998a).

Agent authentication and authorisation

All of the above strategies are initially designed to protect a network server against distrusted codes. When replacing a server with an agent host, and distrusted code with a malicious agent, these strategies can be easily converted for agent host protection. However, as Lange and Oshima (1998) argued, a mobile agent might bring more
potential threats to a hosting system. Firstly, an agent can masquerade as a trusted user invading a host. If this agent used some fee-paying services, it is quite possible fees would be charged to a wrong user's account. Secondly, an agent may access information exceeding its authority. This attack may occur even if the agent represents a correct identity. Thirdly, although its initial functions are well designed by its owner, an agent might still perform illegal actions because it has been tampered with by other visited hosts en route. Facing all these potential threats, some agent-oriented protection mechanism has to be properly facilitated at the agent server side. Agent authentication and authorisation scheme is the first step (see subsection 2.4.5 for more general discussion). In short, the authentication process is based on digital signature technique. Each agent is required to carry enough information in its memory such as its owner's identity and its home context identity, and signed by its owner's private key for non-repudiation purposes. The authorisation process is mainly reliant on access control. A security policy includes a set of named privileges and a mapping function which is in charge of granting different privileges to different principals. Unfortunately, only the static part of an agent is suitable for digital signature protection. An agent owner can not simply sign the entire agent with its private key, because the agent has a migratable state which may vary during its lifetime.

**Path history verification**

Different from an ordinary mobile object which only needs to be transferred between two network nodes, a mobile agent is expected to visit a group of hosts in sequence, therefore, the agent usually has a well-defined itinerary to indicate which hosts it has visited and is planning to visit. A host may need to decide whether to accept an incoming agent or what privilege to grant it, based on its path history because the
agent may become more unreliable if it has been processed by a malicious host along the way. Ordille (1996) suggested two types of scheme for the purpose of agent itinerary verification. In the first design, each visited host is required to sign a separate path record digitally and insert this record into the agent's memory. In the second scheme, each agent server is required to add its identity to an entire itinerary list and sign the complete path history before dispatching the agent to its next destination. Therefore, the subsequent host might either check the individual entry or entire path history to get a clear view of all the platforms that the agent has visited. However, two disadvantages can be noticed. First, the cost of the verification procedure will be very high when the number of visited hosts increases. Moreover, neither arrangement can prevent the itinerary history from being erased by the visited hosts afterwards.

**Proof carrying code**

If an agent has already been mishandled by a malicious host, its function would be affected badly. Therefore, a subsequent host may be more eager to know whether the program carried by a visiting agent is still safe to execute. The Proof Carrying Code (PCC) method proposed by Lee and Necula (1997) provided a feasible solution to this problem, where a code producer (agent owner) and a code consumer (agent host) were studied. The agent host defines a safety policy and makes it public, specifying various authorised operations and associated rules about how to determine the safety of a foreign program. The code producer is required to generate a safety proof out of its source program, to indicate that it conforms to the specifications defined by the policy. Both the native code and safety proof are to be delivered to the server side. As a code consumer, the agent host will launch a validation process before loading the native code for execution. Compared with the proof construction process on the client
side, the server-side proof validating process is much faster, because it only needs to check whether the foreign program's safety proof is valid. Any modification would result in either the proof being no longer valid, or a mismatch between the proof and the source code. Several advantages of this strategy have been observed. First, no cryptography technique is needed in the proof-checking process because what the agent host checks is the intrinsic property of the program, not the code origin. Second, as most of the proof-generation work is carried out at the client side, the agent host saves time in determining a foreign program's safety. Third, this process is performed before any untrusted code is executed within the server's computing environment, therefore, all other potential security threats can be avoided. However, two drawbacks can also be observed. One is the scalability of the certifying-compiler, which is still an uncertain issue in relation to how this proof-generation approach can be employed to complex programs. In addition, both the language used for safety policy specification and the logic for proof-checking still need to be further standardised.

6.3 Special concerns for mobile agent protection

Just as agent host protection can be classified into traditional and agent-oriented solutions, mobile agent protection can be analysed in the same manner. An agent travelling around the network is a self-contained entity, there is little difference between itself and a piece of information during network transmission. Some identified attacks such as eavesdropping and illegal alteration remain potential security threats. Fortunately, the public-key cryptograph technique developed by RSA Security (Tanenbaum, 2003) has provided a suitable solution to this problem. As part of the standards for current Internet computing, this scheme employs a pair of asymmetric keys for information encryption and decryption. Data encrypted with a
public key can only be decrypted with the corresponding private key, and vice versa. Secure Socket Layer (SSL) protocol implemented by Netscape and its recent successor Transport Layer Security (TLS) protocol supported by most web servers are all living examples of employing this scheme to ensure privacy across the network between communicating applications. This technique is still effective for agent transmission, because all its relevant data and transferable state will be marshalled into a byte stream before the agent is ready to move.

6.3.1 Difficulties of protecting agents within a malicious environment

However, an obvious difference between code transmission and agent transmission lies in the fact that the agent is expected to resume its execution at the destination host. Therefore, the remote host is in fact acting as both a program interpreter and a computing environment maintainer. The agent has to use this interpreter and system maintainer to restart its program and communicate with its partners. Moreover, the mobile agent also needs the underlying platform to help its migration to achieve computational advantages. This is why protecting mobile agents against malicious host is always a difficult issue, which has not been fully addressed so far. To illustrate the difficulty of protecting a travelling agent against a dishonest host, Hohl (1998a) introduced an abstract machine model called RASPS (Random Access Stored Program plus Stack). Four basic elements are included: a vector of memory cells, a program counter, a stack, and a stack pointer, as depicted in Figure 6.2:
Technically, the RASPS model is a modified version of the RASP (Random Access Stored Program) model suggested by Elgot and Robinson (1964). One obvious improvement is the added stack, which might be implemented as part of the memory vector. But this modification is helpful in clarifying program execution mechanism without altering the basic characteristics of the original model. When an instruction is ready to be executed, the RASPS machine takes a value stored in the memory cell referenced by the program counter index. Some instructions may consist of several memory cells and associated parameters. Therefore, when these instructions are decoded and executed, additional parameters will be pushed onto the stack and referenced by a stack pointer. After the current instruction is executed, the program counter will increment by a certain number and the next instruction will be ready for computation. With the help of this abstract machine model, Hohl (1998a) presented an abstract attack model to illustrate how an attack takes place within a malicious computing environment, as given in Figure 6.3:
Two main components are included in this attack model: a visiting agent RASPS and an attacker's RASPS. The visiting agent is expected to run a carrying program on its own RASPS, whereas a malicious attacker executes its attack procedure in a separate RASPS. However, the agent cannot sense the computing environment by itself, as that environment has been modelled as a part of the attacker's machine. Whenever the agent wishes to communicate with the runtime environment, it has to make a system call which is actually just a call to parts of the attacker's program. Therefore, the latter's RASPS can not only run its own attack program, but also execute the agent's private program. A malicious attack is quite likely to happen in the ensuing scenario. Before an instruction in the agent's RASPS memory executes, the attacker's RASPS fetches and decodes it. The instruction and associated parameters will be stored onto the attacker's stack. An offensive procedure is launched to analyse this instruction. After analysis, it will be executed with its original parameter, and a final result sent back to the agent's stack. A program counter of the agent RASPS is calculated and updated accordingly before the next instruction is executed. In fact, as a computing environment provider, the agent host RASPS can monitor every instruction loaded into the agent's RASPS memory and modify all execution results stored in the agent's RASPS stack. Through manipulating the program counter in the agent RASPS, the
attacker may even execute the agent's program selectively. To some extent, the visiting agent is almost subordinate to the attacker within this malicious computing environment.

However, does this mean mobile agent protection is entirely impossible in this scenario? The answer is not so pessimistic. Two important reasons can be argued. Firstly, a protection strategy may have two components: prevention and detection. A prevention technique tries to eradicate potential threats through defending a vulnerable item directly, whereas a detection solution tries to discover the correct culprit, thus deterring possible malicious attempts thereafter, which is also quite effective in many circumstances. Secondly, mobile agent protection is a big problem associated with four major requirements: confidentiality, integrity, accountability, and anonymity. An intuitive observation is that if dividing the whole problem of mobile agent protection into four corresponding smaller problems and addressing them individually, the risks of a travelling agent would be minimised to a great extent. The following discussion will follow this divide-and-conquer strategy. Four security requirements will be discussed in as many subsections.

### 6.3.2 Promising solutions for safeguarding data confidentiality

The attack model in Figure 6.3 demonstrates that every instruction executed by the agent must be readable to the host. Therefore, safeguarding data confidentiality at a malicious host's site is not a simple issue. Honestly speaking, there have been no effective solutions so far that can be used to defeat such so-called reading attack. However, it seems that researchers never give up their efforts in this direction, and some suggested strategies are quite promising. The time-limited blackbox approach presented by Hohl (1998b), the computing with encrypted function proposed by
Sander and Tschudin (1998), and the environmental key generation method suggested by Riordam and Schneier (1998) are three worthy examples.

**Time-limited blackbox agent**

The basic idea of this approach is scrambling the source code of an agent with a mess-up algorithm and creating a blackboxed agent. Three requirements are pre-set. First, the blackboxed agent specification cannot be readable, but must be executable and able to perform the same task as the original one. Second, the conversion mechanism for generating blackboxed agents must be flexible enough so that different blackboxed-agents can be created from the same origin. Third, each of them may have an expiration date, which means the confidentiality property can only be valid for a certain interval of time. Following that protection interval, attacks would be meaningless to the attacker, even if they succeeded. A simplified blackbox agent generation process is shown in Figure 6.4:

![Figure 6.4: Time-limited blackbox approach for mobile agent protection](Hohl, 1998b)

The property of the blackboxed agent looks quite attractive. However, its implementation is not so simple. Hohl (1998b) suggested three mess-up algorithms: variable re-composition, conversion of control flow elements, and deposited keys. The variable re-composition algorithm tries to re-organise the set of program variables. The content of each variable is firstly cut into several segments, and then re-organised into new variables. Thus, the relationship between the original variables and
processing elements will be disjointed and more difficult to comprehend. The second algorithm is called "converting control flow elements into value-dependent jumps". As the name indicates, this approach re-formats some common control flow statements into different forms, making them more difficult to recognise than before. Deposited-keys algorithm tries to externalise parts of the information (or keys) to another trusted server. The agent can ask for these keys when needed but will have to submit its current execution state. This method prevents the attacker from reading and analysing the agent directly, because a certain execution state is only available at runtime. In summary, these three mess-up algorithms increase the attacking difficulty for a malicious host and decrease the risks for the agent owner. Unfortunately, at least two obvious drawbacks exist. The first is the higher computation costs (e.g., the cost of agent creation, transmission and execution), which would become very serious if the agent has to communicate with a remote server to get its execution key. Besides, it is difficult to determine the protection strength of these mess-up algorithms. In contrast to the traditional cryptography technology, which is based on a well-defined mathematical model, the time-limited blackbox approach lacks convincing proof regarding its protection strength, although an added expiration time might be helpful to some extent.

**Computing with encrypted function**

Computing with Encrypted Function (CEF) was proposed by Sander and Tschudin (1998). This approach does not try to keep the entire program secret from a hosting system, but focuses on some executable functions' confidentiality. It requires that an encrypted function remains workable, but its ciphered form cannot be understood by the executor. Sander and Tschudin (1998) explained this approach using a two-
entities' scenario, where Alice represents an agent owner, and Bob is a dishonest agent host, as shown in Figure 6.5:

![Diagram](image-url)

Figure 6.5: Computing with encrypted functions
(Sander & Tschudin, 1998)

Alice wants to compute a function $f(x)$ at a remote host maintained by Bob. However, she does not want Bob getting the content of $f(x)$ because it is confidential. So, Alice encrypts the function and works out an encrypted version $E(f)(x)$. If $P(f)$ denotes the program implementing the function $f$, Alice simply sends the program $P(E(f))$ instead of $P(f)$ to Bob. Bob is required to compute $P(E(f))(x)$ with its input $x$ and sends the results back to Alice. After decrypting the results, Alice obtains the final outputs $f(x)$. The secret function $f(x)$ appears in encrypted form at the remote platform, therefore, the potential reading attack would be invalid. A very attractive application of this technique is remote signing. If $f(x)$ is a signature function, a mobile agent can simply carry this function to a possibly malicious host and produce a digital signature there, without the need to expose its signing function and private key. A random number can be employed to avoid the dishonest host from reusing the encrypted function more than once. However, as Sander and Tschudin (1998) acknowledged, only a special class of functions -- polynomials and rational functions are currently proved suitable for this purpose.
There is no known mathematical scheme that can be used to encrypt and transform arbitrary functions to achieve this kind of mobile cryptography.

**Environmental key generation**

In this method, the confidential information carried by an agent is also in an encrypted form, but the decryption key or keying materials needs to be retrieved from its running environment. Before the key is generated and the cipher-text decrypted, it is difficult to determine the agent’s function through direct analysis. As Riordam and Schneier (1998) argued, this environmental-key generation approach is enlightened by the ephemeral keys technique (Myers, 1994), where keys are generated randomly only at the time of use and destroyed immediately thereafter. With this method, the agent will not reveal its purpose until some temporal or spatial conditions are met from its environmental observations. More specifically, time-based construction means that keying materials can be generated either after a given time (forward-time construction), or before it (backward-time construction). Spatial-based construction is based on a direct or indirect data channel such as an online database or local network resources, or a remote trusted server. Suggested key-activation methods include blind search, intrusion detection, and network-supported operations. All these approaches can be combined and nested in various forms to meet different working conditions. One obvious advantage of this approach is that it successfully separates confidential information from its decryption key, which makes a direct reading attack more costly. However, considering the attack model introduced by Hohl (1998a), this does not seem to address the problem fundamentally, as the underlying platform has access to both the agent and its activation environment. In addition, the network-supported keying-material construction is a dangerous operation because it is quite possible to
introduce a directed virus into the hosting machine. A visited host might simply reject such an operation out of concern for its own security.

6.3.3 Facilitating non-repudiable accountability for attack detection

Compared with data confidentiality, accountability is much easier to satisfy, because its protection aim is not to fully prevent malicious entities from launching attacks, but simply to detect associated misbehaviours that have been launched on the agent, and help the agent owner trace the wrongdoer(s). Two potential solutions are quite successful to this end: one is the execution tracing technique suggested by Vigna (1998), and the other is the state appraisal function presented by Farmer et al (1996a).

Execution tracing technique

This technique tries to detect any unauthorised modification performed on the agent's code and state (Vigna, 1998). When a mobile agent finishes its computation and is ready to leave, the visited host is required to create a non-repudiable log recording all operations performed by the agent from its platform. This tracing log is composed of a set of <statement identifier, signature> pair. Values from external sources would also be included in the signature part, in case it is needed for task completion. This tracing log (together with a hash value of the agent's execution trace and that of the agent's final state) would be signed by the visited host, and carried by the agent to its subsequent host. An additional copy will be submitted to a trusted third party as a summary. The subsequent host will be asked to perform the same action until the agent returns home with the final results. If the agent owner doubted the computation results, it could simulate the agent's execution by itself. Any misbehaviour performed by the visited host would lead to a mismatch between the simulation results and the
execution log returned by the agent, and the culprit will be easily identified. One major advantage of this approach is that the protection strength is provable because it employs hash value and the asymmetry-key cryptography technique for the tracing log generation. However, at least two limits can be argued. First, computation of this execution log would become costly with the growth of the number of visited hosts, which presents a serious limitation to its applicability. Second, this technique assumes the agent is single-threaded and does not share memory with other agents on the same platform, which is not practical for many large-sized applications.

**State appraisal mechanism**

In contrast to the execution tracing method, the state appraisal mechanism focuses on a mobile agent's state protection only. After all, based on Farmer et al's observations (1996b), those malicious alterations performed on an agent's state may lead directly to an agent turning harmful, and thus may need to be examined more frequently. This mechanism combines its state appraisal function with an agent host's authentication and authorisation process, with the aim of benefiting both the agent owner and the visited hosts. After a mobile agent dispatch, it carries a specific piece of code -- state appraisal function with it. The visited host might use this function and the agent's current state to compute a set of privileges that the agent could be granted. If a malicious host altered the agent's states in a dangerous way, the state-appraisal check performed at the next site would not meet some important invariants. In that case, the agent will be either granted a reduced privilege or rejected directly by that host. Integrating state appraisal function with a host's authentication and authorisation scheme is especially suitable for an agent working within an environment where some of the parties are in competitive or even hostile relations. Unfortunately, one serious
drawback of this solution is that not all kinds of state alterations can be predicted. It is not always easy to distinguish deceptive alterations on the agent states from normal results computed at a remote platform.

6.3.4 Successful approaches to protecting forward integrity

In addition to accountability, data integrity is another achievable requirement of mobile agent protection. Since a mobile agent is supposed to visit a group of hosts in sequence, the terms of forward integrity and data integrity are always used interchangeably in this area. Specific items suitable for this kind of protection include partial results that an agent collected en route and the itinerary information that it has to carry along its journey. For the former, the partial result authentication code structure suggested by Yee (1997) and the encapsulated results chaining method proposed by Karjoth et al (1998) are two well-designed solutions. For the latter, a mutual itinerary-recording strategy has been presented by Roth (1998).

**Partial result authentication code**

Protected items in this strategy is the partial results obtained by an agent along its route, forward integrity is guaranteed by the secret-key cryptograph technique (Yee, 1997). Before an agent dispatch, the owner generates a list of secret keys. After the agent finishes its computation at a visited host, it summarises the partial results into a message, and encrypts this message with one of the secret keys from its prepared list. The encrypted message will be carried by the agent to its next destination, while the used key will be destroyed at the current platform. One obvious advantage of this strategy is its computation efficiency, because it employs secret-key cryptosystem (instead of public-key digital signature) to encrypt the message, which is faster to
compute. Moreover, if the agent visited a sequence of hosts and one of them was malicious, this method can ensure that partial results generated by all previous hosts cannot be forged. In fact, only the agent owner has the complete secret-key sets, and is capable of checking the forward integrity for all the results. Yee (1997) also suggested using a one-way function for secret-key generation. In that case, the agent does not need to carry a list of pre-selected keys before dispatch, but creates them one by one at each visited host. However, no matter which way the secret-key set is generated, it seems that it is difficult to prevent colluding attacks where two or more malicious hosts along the way launched the attack in collusion. Given that a visited host has full knowledge of all subsequent secret keys, two dishonest hosts working together could illegally forge all the partial results in between to cheat the agent owner, and it is quite possible that another honest host would finally become a scapegoat.

**Encapsulated results chaining**

In order to overcome the above limitation, Karjoth et al (1998) presented an encapsulated results-chaining architecture. In this structure, both a visited host's private key and the agent owner's public key are employed, so it provides not only forward integrity but also data confidentiality. More importantly, it introduces a chain-like relationship. At each visited host, the mobile agent constructs an encapsulated message before continuing its migration. This message contains two items. The first is the current computation results encrypted by the agent owner's public key, which guarantees that only the agent owner can retrieve the inner information. The second is a hash value, which links the encapsulated message passed from its previous host with the identity of its subsequent host. Finally, these two
pieces of information will be sealed by the current host's signature for non-repudiable and unforgeable purposes. Such a chain-like structure successfully prevents two or more colluding hosts from modifying any encapsulated message within the chain. Even the visited host itself cannot modify its own message later on, because another copy of its message has been encapsulated in the subsequent host's message, and protected by that host's signature. Slightly changing the encryption order -- encrypting the computation results with the visited host's private key first, and then encrypting the whole message with the agent owner's public key, this chain-like structure can even hide the identities of all its previous hosts. Unfortunately, this architecture requires a heavy performance cost due to its combined use of the agent owner's public key, a remote host's private key, and a secure hash function.

Co-operating agents recording

Unlike the above two strategies, the co-operating agent recording method is trying to protect an agent's itinerary information, which is another vulnerable item stored in its memory. Instead of employing any secret-key cryptography technique, this scheme takes advantage of two co-operating agents (Roth, 1998). When an agent $\alpha$ finishes its task at a host $h_i$ and is ready to leave, it sends a message to its partner $\beta$, which is supposed to hold another copy of $\alpha$'s itinerary information, and running at a different server. Contents of the message include identities of both its previous host $h_{\text{prev}}$ and its subsequent host $h_{\text{next}}$. Clearly, if the malicious host $h_i$ sends $\alpha$ to a host $h_{i+1} \neq h_{\text{next}}$, agent $\beta$ would immediately detect such a happening. In fact, agent $\beta$ is not necessarily a stationary agent, it might be another mobile agent to perform another task. Therefore, it is indeed a multi-agent mutual recording approach. An attractive advantage of this scheme is reduced computational costs. However, three serious
drawbacks have been identified. First, it does not take any collusion attack into
consideration (where a malicious host may masquerade as its next host with the help
of an accomplice and perform illegal actions thereafter). Second, if one of the co-
operating agents is temporarily unavailable, there seem to be no obvious remedial
measures to take. Third, the message between two co-operating agents is required to
transmit over an authenticated communication channel, which is not a simple issue
within a malicious computing environment.

6.3.5 Providing non-communication anonymity for privacy

In the network communication field, anonymity refers to keeping the identity of a
communicator untraceable. But in an agent-mediated e-business environment, one
cannot expect a host to accept an anonymous agent to conduct business activities from
its platform freely, due to an obvious security concern. However, keeping a mobile
agent's itinerary information untraceable is regarded as reasonable. In an agent's
itinerary, only the identities of the agent's previous and subsequent hosts cannot be
hidden from the current system maintainer, because both of them are needed for host-
to-host communication and agent migration support. Therefore, anonymity within a
mobile agent application is a kind of non-communication anonymity. Clearly,
anonymous connection mechanisms within the network communication field remain
the major candidates for an agent anonymity protection. The mix technology proposed
by Chaum (1981) and the onion-routing strategy suggested by Reed et al (1997) are
two ready-made solutions.

The mix technology was originally designed for an e-mail system, with the aim
of keeping the identity of a sender untraceable from a receiver (Chaum, 1981). A
specific computer called 'mix' is set up between two communication participants.
Each item in the original mail will be pre-processed at the mix machine before being re-delivered to its destination. When a user wishes to send a message to a receiver, it initially generates a combined message and sends it to the mix, which contains two parts. The inner part (including a real message and a challenge number) is encrypted by the receiver's public key, and the outer part indicates the receiver's address. The whole message will be protected by the mix machine's public key. The mix opens this combined message, getting the real address of the receiver, deleting related information for itself and re-sending the inner part to the receiver. The receiver gets an untouched inner message from the mix, but without any knowledge about the sender. If the sender needs a reply from the receiver, it can simply add its own address into the outer part, and let the mix machine help it return the reply. The mix technology makes the traffic analysis attack costly because the sender's identity is already hidden from the receiver. In addition, a series of mixes can be employed in a cascade manner, making the traffic analysis attack more complicated.

Based on Chaum's (1981) mix theory, Reed et al (1996) proposed an onion-routing strategy for anonymous connection. Firstly, onion-routing can be viewed as a network infrastructure designed to replace the traditional TCP/IP socket connection, with the help of a sequence of neighbouring onion-routers. Each of these is like a mix machine. Secondly, a layered-like data structure called 'onion' is employed in this strategy. The innermost layer is the real message for the final receiver, the second layer is intended for the onion-router next to the receiver, and so on until the outermost layer designed for the onion-router next to the sender. Each layer is sealed by the corresponding receiver's public key. When the first router receives this data onion, it peels off the outermost layer and gets to know where the remaining onion should be sent. When the final receiver gets this onion, it opens the innermost layer
and reveals the real message, but does not know how many routers this onion has gone through, because all previous layers have been peeled off and thrown away. If the sender wishes a reply from the receiver, it may add its own identity into either the innermost layer or the outermost layer. That means the initiator can reveal or hide its identity to the responder. After all, the onion-routing strategy tries to achieve anonymous connection, not anonymous communication. To enhance this strategy, each onion-router along the route is required to clean up its connection table after passing each data onion. Therefore, many kinds of traffic analysis attacks can be effectively avoided.

Westhoff et al (1999) extended this strategy for a mobile agent's itinerary protection. The agent's route information is like the message that needs to be protected. The agent owner views all visited hosts as a sequence of onion-routers, and tries to keep its agent's travelling plan secret from all of them. At each visited site, only the identities of its two neighbouring hosts will be revealed. When the agent arrives at the first destination, the visited host peels off the outermost layer and gets to know it is the correct recipient. After the agent finishes its computation, the current host disposes of the outermost layer and only dispatches the agent with the remaining onion to its subsequent destination. If any agent host along the route mishandled parts of the itinerary onion, its subsequent host would immediately detect such misbehaviour. However, the onion-like structure brings some disadvantages as well. If one of the hosts is temporarily unreachable, the mobile agent has to stop its journey and return home although all other hosts are available, after all, only that unreachable host is capable of breaking the outermost layer of the remaining onion.
6.4 Protecting mobile agent against malicious hosts

Through the above investigations, it is not difficult to see that two kinds of information carried by a mobile agent have been identified as vulnerable: one is the agent's original itinerary, the other is the partial result collected en route. Without proper protection arrangements, a malicious host may misuse these pieces of information for its own benefits, or working with its accomplices to harm other hosts on the network. Most existing approaches typically employed some pre-defined schemes to protect certain data components separately. This section seeks to introduce a set of flexible structures to protect more vulnerable items in a dynamic way.

6.4.1 Limitations of current strategies for mobile agent protection

Detailed discussion about partial result protection and agent itinerary protection has been offered in previous sections. In order to identify the limitations of those strategies, a brief summary will be given in Table 6.3.

<table>
<thead>
<tr>
<th>Protect schemes</th>
<th>Merits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SP:</strong> A host signs the partial result using its private key and attached it to the agent like a sealed parcel.</td>
<td>Both data confidentiality and non-repudiability of the partial result are guaranteed via the digital signature technique.</td>
<td>It is difficult to prevent an attached parcel from being thrown away by malicious hosts afterwards.</td>
</tr>
<tr>
<td><strong>PRAC:</strong> A list of secret keys is used to authenticate an agent's intermediate state by way of cryptographic technique.</td>
<td>It is faster to compute than digital signature technique. It ensures that partial results generated at a remote host cannot be forged.</td>
<td>It is difficult to detect and prevent colluding attacks, where a number of hosts within an agent's original itinerary perform illegal alterations in collusion.</td>
</tr>
<tr>
<td><strong>ERC:</strong> A host digitally signs an entry with its private key and uses a secure hash function to link the results with concerned identities.</td>
<td>The chain-like structure successfully prevents a host from modifying any entries in the chain without breaking the chaining relationship.</td>
<td>It comes at a serious performance cost because of the combination use of the public-key crypto-system and a secure hash function.</td>
</tr>
</tbody>
</table>

Table 6.3: Merits and drawbacks of partial result protection

<table>
<thead>
<tr>
<th>Protect schemes</th>
<th>Merits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPE</strong></td>
<td>Agent behaviour will be enforced through a piece of pre-designed tamper-proof hardware. All agents residing on the environment are protected from either unintentional disclosure or malicious manipulation.</td>
<td>The TPE hardware has to be pre-installed in every agent executor's machine, which limits its feasibility and scalability.</td>
</tr>
<tr>
<td><strong>CAR</strong></td>
<td>Two agents are co-operating with each other for mutual recording and mutual protection. Computational cost of this method is relatively low because there is no cryptographic technique involved.</td>
<td>The implementation of an authenticated channel for communicating within a malicious environment is an uncertain issue.</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>A pre-defined itinerary is encrypted layer by layer in an onion-like structure. Legal route extension is also supported. Strong data confidentiality and forward integrity can be ensured. The protection strength is convincing and easily measurable.</td>
<td>It lacks flexibility when one of the hosts in the itinerary is unreachable. Colluding attacks through legal route extension are also difficult to prevent.</td>
</tr>
</tbody>
</table>

Table 6.4: Merits and drawbacks of agent itinerary protection

From the above summary, it can be noted that most current research focuses on either an agent itinerary protection or its partial results protection. These two kinds of information are viewed as separate and static as an ordinary packet transferred over a network. Researchers are constantly distinguishing their vulnerability and suggest various protection schemes to address those weaknesses. There has been little work on exploring whether these two kinds of information have certain inner relations, thus providing a mobile agent with some sort of self-protection capability. In fact, a mobile
agent differs from a transmitted network packet in that it is a self-contained entity, which brings not only complexity but also flexibility to mobile agent protection. Taking a mobile agent-mediated e-business application as an example, the route and quote information carried in a mobile agent's memory are not totally separate, but share certain dynamic relations. If let $P_i \ (1 \leq i \leq n)$ denote the address information of the $i$-th remote platform, $Q_i \ (1 \leq i \leq n)$ denote the partial results obtained by the agent at that context, and $i \leq j \leq n$, from an information volume viewpoint, there was at least an inverse relationship where $\sum_{i=1}^{j} Q_i$ is increasing gradually, while $\sum_{i=j}^{n} P_i$ is decreasing accordingly. Such phenomena prompted speculation as to whether such characteristics could be used directly for mobile agent protection purpose and working out a dynamic solution for both an agent itinerary protection and its partial result protection.

6.4.2 A suggested reversible-onion structure

This subsection suggests a couple of structures to protect a mobile agent against malicious hosts, which seeks to extend Westhoff et al's work (1999), with two further aims. The first is taking advantage of mobile agent-specific characteristics to protect both an agent itinerary and its partial results. The second is achieving both stronger protection functionality and lower development costs. Therefore, a reversible-onion structure is proposed, as depicted in Figure 6.6, where a mobile agent travels among a collection of remote hosts, some of them are malicious (Yang & Nguyen 2003a).
The itinerant agent starts its journey with a route onion, which will be peeled off layer by layer along its way. Making use of certain elements in the route onion, partial results will be sealed piece by piece, and a quote onion created accordingly. After the agent finishes its journey, the route onion is used up, and a quote onion will be taken back home (as shown in Figure 6.7). A decreasing route onion $R_{\text{Reversible-Onion}}$ is expressed in Equation (6.1) and an increasing quote onion $Q_{\text{Reversible-Onion}}$ is expressed in Equation (6.2). Table 6.5 summarises the element notation used in both equations:

$$R_{\text{Reversible-Onion}} = E_{P_{(1)}}[P_R, P_2, K_{P(h)}(P_1, t, r_i, K_{S(1)}, P_2, E_{P(2)}[...]),
E_{P_{(2)}}[P_1, P_3, P_h, K_{P(h)}(P_1, P_2, t, r_2, K_{S(2)}, P_3, E_{P(3)}[...]),
......
E_{P_{(i)}}[P_{i-1}, P_{i+1}, P_h, K_{P(h)}(P_{i-1}, P_i, t, r_i, K_{S(i)}, P_{i+1}, E_{P(i+1)}[...]),
......
E_{P_{(n-1)}}[P_{n-2}, P_n, P_h, K_{P(h)}(P_{n-2}, P_{n-1}, t, r_n, K_{S(n-1)}, P_n, E_{P(n)}[...]),
E_{P_{(n)}}[P_{n-1}, P_h, K_{P(h)}(P_{n-1}, P_n, t, r_n, K_{S(n)})]]][...]]].$$

$$Q_{\text{Reversible-Onion}} = K_{S(n)}[P_n, P_h, t, r_n, Q_n, K_{S(n-1)}[...],
K_{S(n-1)}[P_{n-1}, P_h, t, r_{n-1}, Q_{n-1}, K_{S(n-2)}[...],
......
K_{S(i)}[P_i, P_h, t, r_i, Q_i, K_{S(i-1)}[...],
......
K_{S(2)}[P_2, P_h, t, r_2, Q_2, K_{S(1)}[...],
K_{S(1)}[P_1, P_h, t, r_1, Q_1]]][...]].$$
Reversible-onion structure takes advantage of a mobile agent’s self-carrying and self-verification capabilities. The agent owner initialises a route onion $R_{\text{Reversible-Onion}}$ and dispatches a mobile agent to the Internet. When the agent reaches a remote platform $P_i$ ($1 \leq i \leq n$), the working context decrypts the route onion using its private key, and notices the agent coming from platform $P_{i-1}$, and will continue its journey to platform $P_{i+1}$. With the help of $P_h$’s public key, it also makes sure that the agent owner $P_h$ wishes to share a session key $K_{S(i)}$ and a challenge number $r_i$ with it. A unique trip marker $t$ is also included for non-replaying purpose.
If the negotiation is successful, the visiting agent will ask $P_i$ to record its offer into its memory. Without a session key, $P_i$ has to encrypt its quote using both its private key and $P_i$’s public key for data confidentiality and forward integrity, as suggested in Chess et al (1995). Increased computation cost is a major disadvantage. Employing a session key instead of public-key cryptosystem provides a sound solution, because session key cryptography is typically 100 to 1000 times faster than an asymmetric-key approach (Microsoft, 2000). However, it is difficult for an agent’s home platform to share a set of session keys with all visited hosts along the route. Thanks to the reversible onion architecture, a remote host gets its session key from the route onion safely, and with the least effort. It encrypts its offer $Q_i$ and the challenge number $r_i$ as well as the trip marker $t$ with its session key $K_{s(i)}$, forming a quote onion $Q_{Reversible-Onion}$ and sends the agent to its next stop. Before dispatching the agent to its subsequent platform, $P_i$ explicitly removes its own layer from the route onion. The next platform $P_{i+1}$ gets a remaining route onion and a growing quote onion, but without any knowledge about the previous session key and previous offer. The same process will be carried out at all visited platforms. When the agent’s journey ends, it uses up the itinerary onion and returns home with a complete quote onion. The agent owner decrypts the quote onion with a set of shared session keys, and gets all the offers from platform $P_i$ to $P_n$. Challenge numbers and the trip marker within each piece of the quote onion guarantee the information is up-to-date.

To demonstrate the feasibility and effectiveness of the proposed reversible-onion structure, two aspects need to be considered: one is potential threat, the other is a security requirement. A protection solution works well if it meets all security requirements against all possible misbehaviours. When a roaming agent visits a
dishonest host, at least four kinds of attack are predictable: inserting, deleting, modifying, and colluding, as described below:

**Inserting attack:** A malicious host may insert some worthless platforms into the agent’s initial itinerary or insert irresponsible offers into the agent’s memory. Such an attack may not be harmful, but definitely annoying.

**Deleting attack:** A malicious host may intentionally delete its competitor’s address from the agent’s remaining itinerary, or delete competitive offers provided by its previous hosts. It is obviously a harmful attack.

**Modifying attack:** A malicious host may replace its competitor’s address with a less competitive one, or alter any existing offers for an illegal purpose. It is even worse than a deleting attack because all offers collected by the agent then become unreliable.

**Colluding attack:** Two or more dishonest hosts, acting in collusion, launch the attack. They may work together to tamper with the agent’s itinerary or collected offers, finally placing the blame onto other honest hosts en route.

Facing all these possible attacks, it is necessary to identify a set of security requirements to measure the effectiveness of a proposed solution. Four basic requirements identified in the previous sections remain suited to the current scenario. These are data confidentiality, forward integrity, event non-reputability, and non-neighbours anonymity. A concrete explanation is as follows:

**Data confidentiality:** Only the expected recipient host could extract its session key and the corresponding challenge number. Only the agent owner could open the original offers provided by all visited hosts.
**Forward integrity**: It would not be possible to tamper with the remaining contents in the original itinerary. None of the offers generated by previous hosts can be altered by any of the hosts.

**Event non-reputability**: An offer created by a specific host cannot be repudiated. Misbehaviours of any dishonest host can be identified correctly.

**Non-neighbours anonymity**: None of the identities in the original itinerary need to be disclosed to any other platforms unless for communication.

The performance of the proposed reversible-onion structure is examined in light of security requirements outlined above.

**Data confidentiality**: If the public-key cryptosystem were proven to be secure, no one other than $P_i$ could peel off its own layer of the route onion and get its session key. When the agent returns home, only its owner could decrypt each layer from the quote onion and get the offer $Q_i$ created by $P_i$. A random number $r_i$ circumvents any attack of trying to recognise the cipher text generated from the same plain text.

**Forward integrity**: It is relatively difficult to prevent a dishonest platform from mishandling a visiting agent from a forward-integrity viewpoint. Fortunately, a well-designed structure should detect this event, and effectively identify the troublemaker.

- Suppose a dishonest host $P_i$ inserts another platform after itself without the agent owner’s permission, such misbehaviour would be detected when the agent moves to $P_{i+1}$, because the identity of the added platform is not included in the original itinerary. Besides, without a shared session key, the added platform cannot insert its offer into the quote onion properly. If the agent was rejected and sent back by $P_{i+1}$ to its home, the agent owner ensures that $P_i$ is the offender. A legal itinerary
extension is allowed, but could only be performed through an itinerary extension protocol, which will be discussed subsequently.

• Suppose a dishonest host $P_i$ deletes part of the contents from the original itinerary, such an action can be detected by its subsequent platform $P_{i+1}$ immediately, because it cannot decrypt the remaining onion with its private key as expected. However, if it deletes part of the offers from the growing quote onion before creating its own layer, this misbehaviour cannot be detected by the next platform, because $P_{i+1}$ only receives a quote onion in cipher text. Fortunately, the agent owner can detect this event when the agent returns home, because the quote onion layer generated by $P_{i-1}$ cannot be decrypted with their shared session key.

• A dishonest host $P_i$ may want to replace its competitor’s address with another one, but has no way of knowing its competitor’s correct location in the initial route onion. Even if its competitor happens to be $P_{i+1}$, the malicious host $P_i$ can only send the agent to an arbitrary host, because it does not know $P_{i+2}$’s address. Therefore, the attack becomes invalid. The quote onion is intact when $P_i$ receives it, and any alternation cannot be performed without detection from the agent owner. This arrangement overcomes one of the limitations of the chain-like quote protection structure (Farmer et al., 1996a), where a current working platform has to know all the identities of its previous platforms in order to ensure offers’ forward integrity, which is not feasible due to the non-neighbours anonymity requirement.

• An attack where two malicious hosts act in collusion is deemed to be more harmful than an attack triggered by a single host. The route-onion structure guarantees that a working context can only reveal the identities of its previous and
subsequent hosts. Therefore, two adjacent malicious hosts are unable to commit more illegal actions to the route onion than a single dishonest one.

Deleting and modifying attacks are also invalid when using the quote onion structure. This is because the offers are sealed layer by layer, so cannot be broken (provided the session-key cryptography technique is secure).

An inserting attack from colluding hosts may have harmful consequences. For instance, a malicious host $P_i$ may send the agent to its accomplice $P_{acc}$ instead of to $P_{i+1}$. The accomplice $P_{acc}$ then inserts its own offer into the quote onion and sends the agent back to $P_i$ which in turn resends the agent to $P_{i+1}$. The host $P_{i+1}$ cannot detect what has happened. Consequently, when the agent returns home, host $P_{i+1}$ will be identified as the culprit, because the quote onion fails to open in $P_i$’s layer. Technically, the chain-like protection structure does not provide a satisfactory solution against this type of attack either (Farmer et al., 1996a).

Hypothetically, if $P_{i+1}$ and $P_{i-1}$ are in a colluding group, host $P_{i+1}$ sends the agent back to $P_{i-1}$, and $P_{i-1}$ maliciously replaces $P_i$’s offer. When the agent owner checks the offer chain, the chaining relation appears stable, but $P_i$’s offer cannot be opened, thus $P_i$ becomes a culprit. This drawback can be addressed by an enhanced labelled-onion structure.

**Event non-reputability**: An encapsulated offer layer in the quote onion created by platform $P_i$ would not be repudiated, because no one other than $P_i$ can get its session key $K_{S(i)}$ and one-time challenge number $r_i$ from its own layer in the route onion.

**Non-neighbours anonymity**: A working context only knows the identities of its previous and subsequent neighbours for communication. All other identities either have been erased or are still sealed within the remaining onion. Regarding the quote
onion, the offer layer was encrypted with a secret session key other than the private signature of a visited host, so dictionary attacks would also be effectively prevented.

In summary, the proposed reversible-onion structure satisfies the security requirements of data confidentiality, event non-reputability, and non-neighbours anonymity. However, forward integrity protection is another issue, if a single dishonest host launches the attack, then this misbehaviour can be detected, and the host can be identified either by its subsequent platform or by the agent creator. However, the agent owner cannot correctly identify the culprit when colluding hosts perform the attack. An enhanced labelled onion structure tries to address this problem.

6.4.3 An improved labelled-onion structure

In a hypothetical situation, two dishonest hosts $P_i$ and $P_{acc}$ belong to a colluding group. The host $P_{acc}$ mishandles the quote onion and sends the agent to $P_{i+1}$ in the name of $P_i$. The host $P_{i+1}$ then receives the quote onion in its entirety, and has no way of checking its integrity before processing it. If host $P_{i+1}$ simply creates its own layer and assists the agent to continue its journey, $P_{i+1}$ will eventually be identified as a culprit, because the agent owner cannot decrypt the offer layer generated by $P_i$. In order to overcome this shortcoming, a labelled-onion structure is proposed. An inside and outside label must be worked out prior to host $P_i$ sending the agent to a subsequent host $P_{i+1}$. The outside label assists a recipient host to check the integrity of a quote onion from its direct sender. The inside label will be checked by the agent owner after the agent returns home, as expressed below:
Outside Label: $P_i \rightarrow P_{i+1} \quad K_{P(i)}[H(K_{S(i)}[...]), P_h, t, P_{i+1}] \quad (1 \leq i \leq n)$

Inside Label: $P_i \rightarrow P_{i+1} \quad H(K_{S(i-1)}[...]) \quad (1 < i \leq n)$

$$Q_{\text{Labelled-Onion}} = K_{S(n)}[P_n, P_h, t, r_n, Q_n, H(K_{S(n-1)}[...]), K_{S(n-1)}[...],$$
$$K_{S(n-1)}[P_{n-1}, P_h, t, r_{n-1}, Q_{n-1}, H(K_{S(n-2)}[...]), K_{S(n-2)}[...],$$
$$......$$
$$K_{S(i)}[P_i, P_h, t, r_i, Q_i, H(K_{S(i-1)}[...]), K_{S(i-1)}[...],$$
$$......$$
$$K_{S(2)}[P_2, P_h, t, r_2, Q_2, H(K_{S(1)}[...]), K_{S(1)}[...],$$
$$K_{S(1)}[P_1, P_h, t, r_1, Q_1][...][...]]. \quad (6.3)$$

This structure starts from the first visited platform $P_i \quad (i = 1)$, which is required to create an outside label only. It generates its offer layer and calculates a hash value out of this layer. This hash value, relative identities and the trip marker will be encrypted with $P_i$’s private key and form an outside label, which will then be sent to the platform $P_{i+1}$ with the agent. Upon receiving the mobile agent, $P_{i+1}$ checks this label first. It opens the label and recalculates the hash value. An unmatched hash value indicates an error occurred during transmission, or possibly from a colluding attack. The host $P_{i+1}$ would then reject the coming agent. Otherwise, $P_{i+1}$ includes this hash value as an inside label into its own layer for the quote onion, removes the old outside label, and creates a new one as $P_i$ did, then sends the agent with this new outside label to its subsequent platform $P_{i+2}$. It is possible for host $P_{acc}$ to illegally insert, delete or modify the quote onion, and send the agent to the platform $P_{i+1}$ in the name of its accomplice $P_i$. However, it cannot change the hash value which is protected by $P_i$’s signature. If $P_i$ assists $P_{acc}$ to create the outside label, it has to leave a non-repudiate record in either the inside or outside labels. The agent owner can easily identify the wrongdoer later. Therefore, potential colluding attacks would become invalid.
6.4.4 A set of itinerary extension protocol

Itinerary extension property is one of the most important advantages for an onion-like structure. A visited platform is able to extend a route through introducing another itinerary onion by itself. However, it also creates some difficulties for the agent owner. The agent owner might consider platforms extended by a visited platform as untrustworthy. Moreover, the task of tracing and identifying malicious hosts becomes more complicated if an extended itinerary is overlapped several times or completely out of control. In order to keep the extensible property for the proposed reversible-onion structure, it is necessary to consider how to establish mutual trust between the agent owner and the extended platforms. Sharing a secret session key is the way.

For the initial itinerary in the reversible-onion structure, a shared session key has been included in each layer. For those extended platform en route, such keys are not likely to be prepared in advance, therefore, a key distribution scheme needs to be introduced in an on-demand fashion. A MATE (Mobile Agent iTinerary Extension) protocol is suggested, which could be illustrated in Figure 6.8:

![Figure 6.8: A MATE protocol](image_url)

There are three entities involved in the MATE protocol: a home platform $P_h$, a visited platform $P_i$, and an extended platform $P_x$ (as explained below). The notation for the additional elements in this protocol is summarised in Table 6.6:
Step 1. \[ P_t \rightarrow P_x \quad E_{P(x)}[P_t, P_x, t, r_{tx}, P_h, K_{S(i)}(P_h, P_x, t, r_t)] \]

Step 2. \[ P_x \rightarrow P_h \quad E_{P(h)}[P_x, P_h, K_{P(x)}(P_h, P_x, t, r_{tx}), K_{S(i)}(P_h, P_x, t, r_t)] \]

Step 3. \[ P_h \rightarrow P_x \quad E_{P(x)}[P_h, P_x, K_{P(h)}(P_h, t, r_{sh}, K_{S(x)}), P_x] \]

\[ P_h \rightarrow P_t \quad K_{S(i)}(P_h, P_t, t, r_t, P_x) \]

Step 4. \[ P_x \rightarrow P_t \quad K_{P(x)}(H(K_{S(i)}[\ldots], P_h, P_x, r_{tx})) \]

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{tx} )</td>
<td>A random number used by platform ( P_t ) to challenge platform ( P_x )</td>
</tr>
<tr>
<td>( r_{sh} )</td>
<td>A random number used by platform ( P_x ) to challenge platform ( P_h )</td>
</tr>
<tr>
<td>( K_{S(x)} )</td>
<td>The session key ( P_h ) shared with the extended platform ( P_x )</td>
</tr>
<tr>
<td>( E_{P(x)} )</td>
<td>Encryption using the public key of the extended platform ( P_x )</td>
</tr>
</tbody>
</table>

Table 6.6: Notation of the elements used in the MATE protocol

The MATE protocol starts from a visited platform \( P_t \), which requires the route to be extended. It creates an inner encrypted message for the agent owner \( P_h \) and generates a challenge number \( r_{tx} \) for the extended platform \( P_x \) and includes the trip marker and related identities, sending it to \( P_x \). When \( P_x \) receives this message, it knows \( P_h \) is the agent owner and \( P_t \) requests the itinerary extension. Then, it generates another random number \( r_{sh} \) used for challenging \( P_h \), constructs a similar inner encrypted message, and re-sends to the agent owner.

When the platform \( P_h \) checks this message from \( P_x \), it detects one of the inner messages is encrypted with the session key \( K_{S(i)} \) and contains the random number \( r_t \), so believes the extension request from \( P_t \) is valid. It may still reject the extension by sending a rejection message to \( P_t \). Otherwise, it selects a new session key, encrypts a new message including the challenge number \( r_{sh} \) with the same format as that in the initial route onion, and sends it back to \( P_x \). At the same time, an approval of extension
A message will be sent in reply to $P_i$. The platform $P_x$ receives the session key from $P_h$’s reply and utilises it when creating its own offer layer for the quote onion. After finishing its task, $P_x$ sends the agent back to $P_i$ with its outside label (see Step 4). This time the challenge number $r_{ix}$ will be included in the label as a unique identifier. The MATE protocol is then complete. The route extension could be nested further to $P_x$ in the same manner. The quote onion is changed accordingly, as illustrated in Equation (6.4). Additional elements notation is summarised in Table 6.7:

$$Q_{Reversible-Ontion}^{EX} = K_{S(n-1)}[P_x, P_h, t, r_{ix}, Q_x, H(K_{S(n-1)} [...] , P_x^t), K_{S(n-1)} [...] ,$$

$$K_{S(i+1)}[P_{i+1}, H, t, r_{i+1}, Q_{i+1}, H(K_{S(i+1)} [...] , P_{i+1}), K_{S(i+1)} [...] ,$$

$$K_{S(x)}[P_x, P_h, t, r_{hx}, Q_x, P_x, H(K_{S(x)} [...] , P_x), K_{S(x)} [...] ,$$

$$K_{S(i)}[P_i, P_h, t, r_i, Q_i, H(K_{S(i)} [...] , P_i), K_{S(i)} [...] ,$$

$$K_{S(i)}[P_i, P_h, t, r_i, Q_i] [...] ]]]...].$$

(6.4)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_x$</td>
<td>Partial result collected at the remote platform $P_x$</td>
</tr>
<tr>
<td>$Q_x'$</td>
<td>Partial result collected at the remote platform $P_{x'}$</td>
</tr>
<tr>
<td>$r_{hx}$</td>
<td>A random number used by $P_x$ to challenge $P_h$</td>
</tr>
<tr>
<td>$K_{S(x)}$</td>
<td>The session key $P_h$ shared with the extended platform $P_x$</td>
</tr>
</tbody>
</table>

Table 6.7: Notation of the elements used in Equation (6.4)

The agent owner $P_h$ shared a secret key with an extended platform $P_x$ in a one-to-one authentication manner. However, from a performance viewpoint, this approach results in a higher computation cost because of the employment of both a public key ($P_h$) and a private key ($P_x$). In fact, only $P_x$’s private key is needed for its outside label, so better performance can be achieved with lower computation cost if the agent owner shares a session key with an extended platform without using a public-key cryptosystem. Such an arrangement is also desired when each other’s public key is
temporarily unavailable. Enlightened by the Otway-Rees protocol (Otway & Rees, 1987), which proposed a mutual-authentication method with the help of a KDC (Key Distribution Centre), a MATE with KDC protocol is suggested as in Figure 6.9:

![Figure 6.9: A MATE with KDC protocol](image)

Four entities are involved in this protocol: a home platform \( P_h \), a visited platform \( P_i \) (which wants to extend the original itinerary), an extended platform \( P_x \), and a key distribution centre. The proposed MATE with KDC protocol consists of six steps as explained below. Additional elements notation is summarised in Table 6.8:

**Step 1.** \( P_i \rightarrow P_x \) \( E_{P(x)}(P_i, P_x, t, r_{ix}, P_h, r_{ih}, K_{S(i)}(P_h, P_x, t, r_i, r_{ih})) \)

**Step 2.** \( P_x \rightarrow P_h \) \( P_x, P_h, r_{sh}, r_{ih}, K'_{P(x)}(P_x, P_h, t, r_{sh}, r_{sd}), K_{S(i)}(P_h, P_x, t, r_i, r_{ih}) \)

**Step 3.** \( P_h \rightarrow KDC \) \( P_x, P_h, K'_{P(x)}(P_x, P_h, t, r_{sh}, r_{sd}), K_{P(h)}(P_x, P_h, t, r_{sh}, r_{hd}) \)

**Step 4.** \( KDC \rightarrow P_h \) \( K'_{P(h)}(P_x, P_h, t, r_{sd}, K_{S(i)}) \)

\( KDC \rightarrow P_x \) \( K'_{P(x)}(P_x, P_h, t, r_{hd}, K_{S(i)}) \)

**Step 5.** \( P_h \rightarrow P_i \) \( K_{S(i)}(P_h, P_i, t, r_i, P_x, r_{ih}) \)

**Step 6.** \( P_x \rightarrow P_i \) \( K_{P(x)}(H(K_{S(i)}[...], P_h, P_i, r_{ix})) \)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{sd} )</td>
<td>A random number used by an extended platform ( P_x ) to challenge the KDC.</td>
</tr>
<tr>
<td>( r_{hd} )</td>
<td>A random number used by the platform ( P_h ) to challenge the KDC.</td>
</tr>
<tr>
<td>( K'_{P(x)} )</td>
<td>Secret key shared between platform ( P_x ) and the KDC.</td>
</tr>
<tr>
<td>( K'_{P(h)} )</td>
<td>Secret key shared between platform ( P_h ) and the KDC.</td>
</tr>
</tbody>
</table>

Table 6.8: Notation of the elements used in MATE with KDC protocol
The MATE with KDC protocol starts from a visited platform $P_i$. The first message is sent to $P_x$ and contains a pair of random numbers, $r_{ix}$ used to challenge $P_x$ and $r_{ih}$ used to challenge $P_h$, and an encrypted inner message used for the agent owner $P_h$ to check its identity. When $P_x$ receives this message, it generates another pair of random numbers, $r_{sx}$ used to challenge $P_x$ and $r_{sh}$ used to challenge $P_h$, and $r_{sd}$ used to challenge the KDC. It then constructs a new message containing two challenge numbers $r_{sx}$ and $r_{sh}$, and two inner encrypted messages (one for the KDC, one for the agent owner $P_h$). A secret key $K_{P(x)}$ is shared between the platform $P_x$ and the KDC only.

Once the message has been received from $P_x$, the platform $P_h$ verifies the challenge number $r_i$ and the inner challenge number $r_{ih}$ to ensure the extension request from $P_i$ is valid. Because an Internet intruder has no way of breaking the encrypted part generated by $P_i$, any alteration will result in a mismatch of the two challenge numbers. If the extension is approved, the agent owner will construct an analogous message of its own containing the challenge number $r_{sh}$ and a new random number $r_{sd}$, and send this message to the KDC. Subsequently, another approval message will be sent back to $P_i$. The KDC checks to see if the $r_{sh}$ in both parts is the same. As only the KDC knows the shared secret keys $K_{P(x)}$ and $K_{P(h)}$, if an Internet intruder tampers with $r_{sh}$ or replaces the encrypted part in the previous message, this misbehaviour will be detected immediately. The KDC then selects a session key and encrypts it twice, once for $P_h$ and once for $P_x$. The receiver’s challenge number will be included in each message.

Platforms $P_h$ and $P_x$ will recognise their individual challenge number in their reply from the KDC, and so will believe it is from the KDC, not an Internet intruder,
and that the session key $K_{S(x)}$ is valid. Platform $P_s$ will then use this session key to create its own offer layer for the quote onion and send the agent back to $P_i$ with a new outside label after everything is complete (see Step 6). The challenge number $r_{ix}$ will be embedded in the label at this stage as a unique identifier. The MATE with KDC protocol process is then complete.

### 6.4.5 A re-organised multi-layered inverted-pyramid structure

As discussed previously, labelled reversible-onion architecture satisfies a travelling agent’s security requirements such as data confidentiality, forward integrity, event non-reputability and non-neighbours anonymity, because it is naturally unbreakable. However, it has an obvious weakness when one of the remote platforms is temporarily not available, a working context has to send the travelling agent back to its owner after trying a certain number of times. This may become a serious problem if an agent travels within an open network where some hosts are often unreachable. It affects the route onion more than the quote onion, because no one other than the next platform is able to reveal the remaining itinerary. Such a phenomenon pressed efforts to redesign the itinerary protection structure, which could separate the protected itinerary at each visited platform, while retaining its protection advantages.

This subsection attempts to overcome such limitations by introducing a MIP (Multi-layered Inverted-Pyramid) structure with a VTTP (Via Trust Third Party) protocol for an agent’s itinerary protection that is presented in Equation (6.5). The labelled quote onion structure will remain unchanged.
The agent owner generates a multi-layered itinerary using its private key and a visited platform’s public key. Each layer contains the identities of a visited platform’s previous and subsequent hosts. Challenge number \( r_i \) and session key \( K_{S(i)} \) are included, and will be used for a visited platform creating its own layer for the quote onion. To ensure forward integrity for the itinerary, the agent owner calculates a group of hash values in advance for each layer. Each visited platform will get a set of hash values for all its following layers. For example, the first platform will get \((n-1)\) hash values if there are \(n\) visited platforms in total, the \(i\)-th platform will get \((n-1)\) hash values, and the last platform will get no hash value at all. So, the length of each encapsulated layer is shorter and shorter, shaping the proposed structure like an inverted pyramid from its top to the bottom.

Each visited platform is allowed to open the top layer of the structure when it receives a coming agent and erases this layer before sending the agent to its next stop. If one of the platforms en route is temporarily unreachable, a VTTP protocol will be employed with the help of a trust third party, as depicted in Figure 6.10. Additional elements notation in the VTTP protocol is summarised in Table 6.9:
Figure 6.10: A VTTP protocol

Step 1. \( P_{i-1} \rightarrow P_{TTP} \) 

\[ E_{TTP}[P_{i-1}, P_{TTP}, K_{P(i-1)}(P_{TTP}, P_h, t, P_i, r_{i-1})], \]

\[ E_{TTP}[P_h, P_{TTP}, K_{P(h)}(P_{i-1}, r_{i-1}, t, P_i, r_i, ..., P_n, r_n)] \]

\[ P_{i-1} \rightarrow P_h \] 

\[ K_{S(i-1)}[P_{i-1}, P_h, t, r_{i-1}, P_i] \]

Step 2. \( P_{TTP} \rightarrow P_i \) 

\[ K_{TTP}[P_{TTP}, P_i, E_{P(i)}(P_{TTP}, P_h, t, P_{i-1}, r_i)] \]

Step 3. \( P_{TTP} \rightarrow P_{i+1} \) 

\[ K_{TTP}[P_{TTP}, P_{i+1}, E_{P(i+1)}(P_{TTP}, P_h, t, P_i, r_{i+1})] \]

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{TTP} )</td>
<td>Internet address of the trust third party</td>
</tr>
<tr>
<td>( K_{TTP} )</td>
<td>Private key of the trust third party</td>
</tr>
<tr>
<td>( E_{TTP} )</td>
<td>Asymmetrical encryption using the public key of the trust third party</td>
</tr>
</tbody>
</table>

Table 6.9: Notation of the elements used in VTTP protocol

The VTTP protocol starts from a visited platform \( P_{i-1} \), which discovered its subsequent platform \( P_i \) was not available. It encrypts the challenge number \( r_{i-1} \) and related identities with its session key \( K_{S(i-1)} \), and sends a message to the agent owner \( P_h \). At the same time, it sends the travelling agent with another encrypted message containing the same information to the trust third party \( P_{TTP} \) using its private key. When the platform \( P_h \) gets this message, it notices the random number \( r_{i-1} \) is included and encrypted by the session key \( K_{S(i-1)} \), and it believes the skipping request from \( P_{i-1} \) is valid. If \( P_h \) agrees with such request, it will send \( P_{TTP} \) an approval message.
containing the identity of \( P_{i-1} \) with its challenge number \( r_{i-1} \), identity of \( P_i \) with its challenge number \( r_i \), and identity of \( P_{i+1} \) with its challenge number \( r_{i+1} \).

Firstly, the trust third party checks to see whether the challenge number \( r_{i-1} \) in both messages is the same. Because only \( P_{i-1} \) and \( P_h \) share this number, an Internet intruder has no way of accessing it. After ensuring agreement on both sides, it sends the agent and a message to \( P_i \) containing the challenge number \( r_i \) with the identity of \( P_{i-1} \). If the platform \( P_i \) is reachable, it opens the message, and believes such replacement is authorised by the agent owner because the challenge number \( r_i \) in \( P_{TTP} \)'s message is the same as that in the original route layer. If the platform \( P_i \) is still unavailable, the trust third party will delete \( P_i \)'s layer from the MIP structure, and send the agent with a message in the same format to the following platform \( P_{i+1} \). This time \( P_{i+1} \) will perform the same verification described above and process the agent accordingly. The VTTP protocol is then completed.

Due to the combined use of the agent owner’s private key and a visited platform’s public key, three of four basic security requirements, data confidentiality, event non-reputability and non-neighbours anonymity will not be affected for the proposed MIP architecture. Therefore, the following security analysis will focus on its forward integrity protection against possible inserting, deleting, modifying and colluding attacks, to observe if any misbehaviour can be detected and the culprit can be correctly identified.

**A dishonest platform \( P_i \) inserts another platform into the initial itinerary**

Such misbehaviour will be detected by its subsequent platform \( P_{i+1} \), because a corresponding hash value is not included in the original route layer, which is protected by the agent owner’s signature. If \( P_i \) simply inserts a platform behind it, the next
platform $P_{i+1}$ will detect the identity of its previous platform unmatched. In addition, the illegally-added platform has no way of creating an outside label based on the current quote onion without revealing its real identity. So, inserting attack is invalid.

**A dishonest host $P_i$ deletes certain layers from the original route structure**

This occurrence will also be detected by its subsequent platform, because it could not delete corresponding hash value records from all those layers above. For the same reason, if $P_i$ deletes its subsequent platform $P_{i+1}$ from the MIP structure, the following platform $P_{i+2}$ will notice the identity of its previous platform is different from the one that was encrypted in the original route. Besides, $P_i$ cannot forge an outside label for the quote onion without $P_{i+1}$’s private key. The itinerary protection and quote protection demonstrate a mutual-verification property here.

**A dishonest host $P_i$ wants to replace its competitor’s layer with another one**

Host $P_i$ has no way of knowing its competitor’s correct location within the original architecture. Any replacement will cause a mismatch of the corresponding hash value being detected by platform $P_{i+1}$. If $P_i$ tries to replace $P_{i+1}$’s layer, it does not have the agent owner’s private key and cannot forge the encrypted part and containing relevant identities with the correct format. The agent carrying any mis-formatted information will be rejected by the receiving platform and sent back to its owner.

**A dishonest host tries tampering with the agent with an accomplice host**

In an onion-like architecture, it is difficult for a dishonest host to discover its accomplice on the remaining onion, because the route onion is unbreakable. However, for the MIP structure, every layer is separate. Dictionary attacks may happen where a malicious host $P_i$ broadcasts copies of the remaining route to its accomplice $P_{acc}$, which then tries to decrypt each layer. If host $P_{acc}$ obtains a reasonable result, it
means it is a member of the initial route. Thus, they could work in collusion. The inserting attack seems invalid because they could not inject corresponding hash values into any layer (which is protected by the agent owner’s signature). However, they could simply delete all layers in between from $P_{i+1}$ to $P_{\text{acc}-1}$ without being detected by any other platforms en route. Fortunately, they can never get correct session keys and challenge numbers, and thus could not forge their offers. Host $P_{\text{acc}}$ would be identified as a culprit when the agent returns home because the agent owner will discover the quote onion in $P_{\text{acc}-1}$ layer cannot be opened as expected. If $P_{i}$ intends to replace any layers in between, either its own identity or $P_{\text{acc}}$’s identity has to be disclosed to the added members, and be included in the inner label of the quote onion. Again, without a shared session key, the agent owner cannot open the quote onion from $P_{\text{acc}-1}$’s layer, because $P_{\text{acc}}$ has to take this responsibility.

In general, the proposed MIP architecture also meets all security requirements for a travelling agent, and allows flexibility while one of the platforms along the route is temporarily unreachable. Potential attacks can be detected without implicating honest platforms.

### 6.4.6 A strengthened trust third party checkpoint structure

Although colluding attacks has been proven invalid for the MIP architecture, they are still an annoying threat. To avoid these occurrences, it is necessary to further encrypt each layer within the MIP structure. An improved TTPC (Trust Third Party Checkpoint) structure is therefore proposed, as shown in Figure 6.11. A modified route protection structure is expressed in Equation (6.6). A state-transaction diagram of the travelling agent in each host is depicted in Figure 6.12.
\[ R_{TTPC} = E_{P(TTP)}[P_h, P_{TTP}, K_{P(h)}(P_{TTP}, t, K_{S(TTP)})], \]
\[ K_{S(TTP)}[P_1, t, E_{P(i)}(P_{TTP}, P_h, K_{P(h)}(P_1, t, r_i, K_S(t)), H(K_{S(TTP)}[P_2, \ldots]), \ldots, H(K_{S(TTP)}[P_{i+1}, \ldots]))], \]
\[ \ldots \]
\[ K_{S(TTP)}[P_1, t, E_{P(i)}(P_{TTP}, P_h, K_{P(h)}(P_1, t, r_i, K_S(t)), H(K_{S(TTP)}[P_{i+1}, \ldots]))], \]
\[ \ldots \]
\[ K_{S(TTP)}[P_n, t, E_{P(i)}(P_{TTP}, P_h, K_{P(h)}(P_n, t, r_n, K_S(t)), H(K_{S(TTP)}[P_h, t]))], \]
\[ K_{S(TTP)}[P_h, t]. \]

(6.6)

\[
\begin{align*}
R_{TTPC} \text{ in [1]} & \quad R_{TTPC} \text{ in [2]} & \quad R_{TTPC} \text{ in [3]} \\
E_{P(TTP)}[\ldots] & \quad E_{P(i)}(\ldots) & \quad K_{S(TTP)}[P_1, t, E_{P(i)}(\ldots)] \\
K_{S(TTP)}[P_1, t, E_{P(i)}(\ldots)] & \quad \ldots & \quad \ldots \\
\ldots & \quad \ldots & \quad \ldots \\
K_{S(TTP)}[P_n, t, E_{P(i)}(\ldots)] & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] \\
K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] \\
R_{TTPC} \text{ in [4]} & \quad R_{TTPC} \text{ in [5]} & \quad R_{TTPC} \text{ in [6]} & \quad R_{TTPC} \text{ in [7]} \\
E_{P(i)}(\ldots) & \quad K_{S(TTP)}[P_n, t, E_{P(i)}(\ldots)] & \quad E_{P(n)}(\ldots) & \quad K_{S(TTP)}[P_h, t] \\
\ldots & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] \\
K_{S(TTP)}[P_n, t, E_{P(i)}(\ldots)] & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] \\
K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t] & \quad K_{S(TTP)}[P_h, t]
\end{align*}
\]

Figure 6.12: A state-transition diagram of the agent's route structure

The agent owner initialises an original itinerary \( R_{TTPC} \) and sends the agent to a trust third party \( P_{TTP} \) first. When the trust party receives the agent, it opens the top layer of
the itinerary and gets the trip marker \( t \), and its session key \( K_{S(TTP)} \). With this session key, it decrypts the second layer of the structure and identifies the first platform \( P_1 \) that the agent wants to visit. Then it deletes all irrelevant contents for itself and sends the agent with the remaining structure to \( P_1 \). When \( P_1 \) accesses the agent, it decrypts the top layer with its private key, and believes the agent owner is \( P_s \) and the agent is coming from a trust third party. At the same layer, it gets both its challenge number \( r_1 \) and its session key \( K_{S(1)} \). After providing its own offer and constructing a labelled quote onion, it deletes this layer and sends the agent back to \( P_{TTP} \).

After receiving the agent, the trust third party \( P_{TTP} \) would help to check the quote onion and the outside label provided by the platform \( P_1 \) and ensuring its integrity. If nothing suspicious is detected, \( P_{TTP} \) decrypts the third layer of the itinerary and gets the identity of the second platform \( P_2 \) where the agent wishes to travel. It then deletes irrelevant information from this layer and sends the agent with the remaining contents to \( P_2 \). The same process will go on and on until the itinerary concludes. The trust party sends the agent back to its owner. With the help of the trust third party, the agent’s multi-hop travel is changed to a set of one-hop travel. All potential attacks performed by either a single dishonest host or two colluding hosts are diminished. Even the identities of a visited platform’s neighbours need not be revealed. Unfortunately, slow performance is a penalty of this arrangement, especially when the original itinerary is already very long, this would seriously limit the benefits of mobile agent technology.
6.5 Chapter summary

The vulnerability of a mobile agent working in a remote platform lies in that it has to communicate with its computing environment with the help of foreign system services, unfortunately all those components can be modelled as part of the attacking machine maintained by that host. Therefore, when a mobile agent works remotely, it is difficult to get enough protection from its owner, but is more or less subordinate to a visited platform. Through the analysis of current strategies for mobile agent protection, it could be realised that a mobile agent with its private information was always viewed as a passive item transferred over a network. There has been little work on exploring whether the travelling agent has certain active characteristics, which hopefully can be used for agent protection purposes.

As observed in this chapter, a mobile agent has a kind of self-carrying and self-verification capability which can be skilfully employed for mobile agent protection. Therefore, a reversible onion and a labelled onion structures are suggested. A route onion is used to protect a mobile agent's itinerary, while at the same encryption layer it carries a secret key which can be used for its quote onion construction. A quote onion is used to protect the agent's partial results, while at the same time an outside label is attached to verify the integrity of the accompanying quote onion. Technically, these two structures are natural extensions of Westhoff et al's work (1999), but presented here with two further improvements. One is protecting both an agent's partial result and its itinerary information within the same scheme, and the other is achieving both higher protection strength and lower computational costs. In order to demonstrate the protection functionality of the suggested structures, four kinds of attacks are tested: inserting, deleting, modifying, and colluding attack. Facing all these attacks, the suggested protection structures meet all security requirements.
In addition, a Mobile Agent iTinerary Extension with Key Distribution Centre protocol is proposed to help a travelling agent extend its itinerary along its way. A re-organised Multi-layered Inverted-Pyramid structure with a Via Trust Third Party protocol is also suggested to deal with the situation where one of the hosts en route is temporarily out of service. Finally, a strengthened Trust Third Party Checkpoint structure is presented, which in fact changes an agent's multi-hop travel to a series of one-hop travel in order to eliminate all potential threats in the course of migration. Unfortunately, enhanced protection strength comes with performance costs, because it limits many expected advantages of employing mobile agent technology within associated e-business applications.

After all, protection strength and performance degradation are two common conflicting factors. It seems that a difficult problem for mobile agent protection is not simply the design of a complex security mechanism, but the consideration of agents' security requirements and their computational performance in an all-round way. An optimised protection scheme in the future might be established on different agent-versus-system trust levels and/or different security-versus-performance tradeoffs.
Chapter 7

Conclusions and Future Work

7.1 Summary

The mobile agent computing paradigm has shown great promise in its potential to change the way of traditional business transactions and system maintenance. However, before mobile agents can be widely employed for e-business applications, some major obstacles have to be identified and properly overcome. This thesis seeks to answer these research questions presented in the introduction and achieve a set of pre-defined objectives.

 Initially, a comprehensive review on the multi-agent and mobile agent technology was presented. The study showed that a software agent with certain secondary properties had attracted much attention from the research community due to its potential for adding value to an agent system. Intelligent agents with adaptability and mobile agents with mobility were both good examples. For the mobile agent technology, it followed a traceable line of evolution: from peer-to-peer messaging and client-server remote procedure calling, to process migration, remote evaluation, code-
on-demand, and object migration. However, it was not intended to replace them once and for all, but to work with them, thus enabling a mobile agent to enjoy aggregate advantages over many other individual techniques. Such characteristics have been well incorporated into various mobile agent platform designs and resulted directly in their natural support for a wide range of network systems development. A cross-language comparison on three major agent platforms has been made, with focuses on their execution environment establishment, migration and communication supports, and security protection mechanisms. In general, IBM's Aglets performed better than Telescript and Agent Tcl, it apparently benefited from its creation language -- Java. In fact, Java-based platforms have emerged to be the mainstream in current mobile agent-mediated system construction, although there were weak facets.

Three mobile agent-oriented e-market structures were examined in Chapter 3: the e-shopping centre in Telescript, the TabiCan market in Aglets, and the Yellow-Pages services in Agent Tcl. The study showed that mobile agent technology firstly changed the shape of a traditional market, where not only the scope of business activities was expended significantly, but also the means of various business actors participating in the market became more flexible. A generic architecture was therefore proposed, which sought to combine advantages from all three structures cited above, with four components involved: consumer, vendor, home, and public e-market. Each of these was supposed to take part in this dynamic e-market network in a self-contained manner. In addition to the shape changes, mobile agents were also expected to contribute the e-business models in an either collaborative-agents or society-of-agents fashion to benefit various business roles on their profits generation. Five mobile agent-supported e-business models were suggested: the SBSS and MBMS models were transplanted from their agent prototypes but adding some specific features for
mobile buyers or vendors, whereas the SBMS, SSMB and MB models were native to
the mobile agent-mediated e-business domain. They supported dynamic market
formation and value network development through enhancing an e-business's external
relationship with a large number of market participants.

Chapter 4 explored the applicability of software patterns for helping mobile
agent system developers communicate their design experiences in a standard style.
Common problems for most agent application development were lacks of: an effective
way for design knowledge documentation, a specific method for system-level
abstraction, and an in-depth discussion on problem-solving strategies. OO design
patterns provided a meaningful solution for these problems, as a mobile agent itself
could be viewed as an active object. A layered-like pattern for a functional agent and
an architectural pattern for an agent system were examined to demonstrate the
applicability of introducing OO design patterns into the agent world. A set of patterns
specific to Internet shopping and agent-mediated e-business was also reviewed to
illustrate how business operators could use software patterns to simplify their systems
design. The final efforts were put into the mobile agent-supported e-business field. At
first, a group of domain-dependent general forces were identified: market openness
and service quality, free mobility and self-control capability, remote delegation and
effective management, and group delegation and inter-group collaboration. Four
associated design patterns were then proposed: the market-organiser pattern was
related to virtual market management, which tried to improve its service quality to all
the visiting trader agents while keeping its market openness. The agent-side and
server-side comparison-shopping patterns were directly associated with agent-
mediated shopping scheme. The former focused on facilitating a shopping agent's free
mobility while enhancing its self-navigation property, while the latter sought to
strengthen an agent owner's control over its delegating agent for security fulfilment. Finally, an itinerary-balancing pattern was designed for a group of co-operating agents to realise inter-group collaboration during their travelling. All of these patterns sought to minimise potential conflict among those identified forces.

Chapter 5 examined several decision-making techniques that could assist a mobile agent to deal with various uncertainties within a practical e-business situation. Two major approaches were investigated: SC technologies and business rules. The study showed that both methods had been frequently employed to address problems with partial truth and approximate information. However, a common limitation was also observable: both lacked the flexibility to handle those online problems that might be encountered in many real applications related to financial games. Competitive analysis was then introduced, with general proof techniques and different adversary models being analysed. Its applications within the business field were also studied. Employing a competitive algorithm for agent-oriented task scheduling and decision-making was finally explored. For the former, competitive analysis might help multiple agents working in the same system carry out a given task with minimised computation costs. A mobile agent-based after-sales service system incorporating an online RES algorithm proved this point. For the latter, an online decision-making algorithm had the potential to improve an agent's adaptability within an ever-changing e-business environment. An online TAFTA problem was thereby identified with a TAB algorithm being derived. Enlightened by the ENH algorithm for the online Snacks problem, the proposed TAB algorithm presented a number of meaningful improvements such as optimised measurement for online costs, broader coverage for problem-solving, and enhanced operability for decision-making. In addition, two sets of simulations and
associated algorithms' comparison were also conducted to prove the effectiveness of
the suggested approach to mobile agent-based e-business applications.

The last chapter sought to address major obstacles impeding mobile agents from
being employed in commercialised systems. Compared with other surveys in the same
area (which analysed associated security issues from general technical perspective),
Chapter 6 put relevant discussion in a mobile agent-based e-business context. From
the victim aspect, five kinds of potential threats were identified: malicious host attacks
host(s), malicious agent attacks host(s), malicious agent attacks agent(s), malicious
host attacks agent(s) and malicious third party attacks agent-and-host. From the
protector aspect, five security requirements for both agent and agent host protection
were also clarified: confidentiality, integrity, accountability, availability and
anonymity. For the agent hosts protection, strategies came from either traditional
network server implementation or agent-specific solutions. For the mobile agent
protection, a RASPS-machine-based attacking model explained its complexity.
However, the toughness did not erase all the possibilities. Two intuitive observations
were made. First, the divide-and-conquer strategy might minimise risks involved in
allowing an itinerant agent to visit a set of dishonest hosts. Second, prevention and
detection approaches might be employed in combination to safeguard a mobile agent
from being attacked along its way. A collection of existing solutions associated with
this issue was investigated, limitations of current schemes were thereby identified. In
order to overcome those drawbacks, a reversible-onion structure and an enhanced
labelled-onion structure were suggested. As a technical extension of the original
onion-routing strategy, the proposed protection scheme made an obvious
improvement, it explored a mobile agent's self-carrying and self-verification
capability which might be skilfully used for an agent's self-protection and led directly
to optimised computational costs and extended protection items. A re-organised MIP structure and a strengthened TTPC structure were also offered, which indicated that balancing protection strength and performance degradation was more important than merely satisfying one of them.

7.2 Directions for the future

There are many interesting directions that can be taken from this work. Three immediate directions lend themselves to further research. The primary aim is consummating associated techniques to promote the employment of the mobile agent paradigm for commercialised systems development.

- A fundamental and comprehensive design on the mobile agent-mediated e-business model is still lacking. Based on the observation that mobile agents always demonstrate their potential within an agent society, it is more desirable to consider multi-agent-based e-business models as a whole. On the one hand, it might be helpful to analyse mobile agent-oriented contributions to the dynamic market or value network construction in a broader technical background. On the other hand, it might improve a business operator's flexibility to select optimised models to enhance its network presence and work for its profit generation strategy.

- The proposed mobile agent-mediated design patterns are merely isolated cases, a manageable and extensible pattern classification method is thus required. The explored pattern category can be arranged in multiple levels. The top-level category is more likely to come from mobile agent-based secondary properties or the major roles they can play within an online market. The sub-level category deals with various concrete patterns, these are not necessarily balanced in number, but must be informative. One objective of such an arrangement is to help those
pattern designers continue their efforts in this specific area, and assist those new practitioners to grasp associated patterns more conveniently.

- Additional efforts still need to be devoted into the mobile agent protection field. Initially, a software-based arrangement is always a preferable solution. The problem lies in how to effectively provide security against a malicious entity face-to-face. Innovative ideas are required. In addition, with reduced costs for network communication and hardware facilitation, employing hardware-supported techniques for mobile agent protection is becoming increasingly feasible. However, at least two essential tasks have to be addressed in advance: setting up a technical standard for such equipment, and providing a rigorous measurement for its protection strength.
Appendix A

Basic Classes for Mobile Agent Design Patterns

A.1 Market-organiser pattern

The Lookup class

```java
public class Lookup extends MobilityAdapter {
    AgletProxy marketGuideProxy, myPartnerProxy;
    Vector tradeCommodityVec, myReplyGroupVec;

    public Lookup (Vector myCommodityVec)
    {tradeCommodityVec = myCommodityVec.clone ();}  

    public void onArrival (MobilityEvent me) 
    {  try  // looks up the market guide proxy on arriving 
        {marketGuideProxy = (AgletProxy) getAgletContext ().getProperty ("MarketGuide");
         myReplyGroupVec = (Vector) marketGuideProxy.sendMessage (new Message ("Search",
                                                      (Object) tradeCommodityVec));
        }catch (Exception e){} 
    }

    public void enter ( ) // retrieves all the available trading partners proxies 
    {for (int  i = 0, i <= myReplyGroupVec.size (), i ++)  
     {myPartnerProxy = (AgletProxy) myReplyGroupVec.elementAt (i);}}

    public void onDispatching (MobilityEvent me) // exits the trading group before leaving 
    {try  
     {  marketGuideProxy.sendOnewayMessage ("Exit", (Object) tradeCommodityVec);
     } catch (Exception ex) { }
    }
}```
The marketGuide class

```java
public class marketGuide extends Aglet {
    AgletProxy marketProxy;
    Hashtable tradingGroupHash;
    Vector tradingGroupVec, myBuyerDBVec, mySellerDBVec, myTraderVec;
    String myTraderStatus;

    public void onCreation (Object obj) // constructs the MarketGuide class registering as "MarketGuide"
    {
        marketProxy = this.getProxy ();
        getAgletContext ().setProperty ("MarketGuide", marketProxy);
    }

    public boolean handleMessage (Message msg)
    {
        if (msg.sameKind ("Search")) // deals with the Search message
        {
            myTraderVec = (Vector) msg.getArg ();
            if (tradingGroupHash.containsKey (myTraderVec.elementAt (1)))
            {
                tradingGroupVec = (Vector) tradingGroupHash.get (myTraderVec.elementAt (1));
                myBuyerDBVec = tradingGroupVec.elementAt (0);
                mySellerDBVec = tradingGroupVec.elementAt (1);
                myTraderStatus = (String) myTraderVec.elementAt (2);
                if (myTraderStatus.compareTo ("buyer")) // sends back available sellers' database for buyers
                {
                    msg.sendReply ((Object) mySellerDBVec);
                    myBuyerDBVec.put (myTraderVec.elementAt (0)); // refreshes the buyers' database
                }
                if (myTraderStatus.compareTo ("seller")) // sends back available buyers' database for sellers
                {
                    msg.sendReply ((Object) myBuyerDBVec);
                    mySellerDBVec.put (myTraderVec.elementAt (0)); // refreshes the sellers' database
                }
            }
        }
        return true;
    }

    elseif (msg.sameKind ("Exit")) // deals with the Exit message,
    {
        myTraderVec = (Vector) msg.getArg ();
        if (tradingGroupHash.containsKey (myTraderVec.elementAt (1)))
        {
            tradingGroupVec = (Vector) tradingGroupHash.get (myTraderVec.elementAt (1));
            myTraderStatus = (String) myTraderVec.elementAt (2);
            if (myTraderStatus.compareTo ("buyer")
            {
                myBuyerDBVec = tradingGroupVec.elementAt (0);
                myBuyerDBVec.remove (myTraderVec.elementAt (0));
            }
            if (myTraderStatus.compareTo ("seller")
            {
                mySellerDBVec = tradingGroupVec.elementAt (1);
                mySellerDBVec.remove (myTraderVec.elementAt (0));
            }
        }
        return true;
        else return false;
    }
}
```
The visitingTrader class

public class visitingTrader extends Aglet
{
    Lookup myLookup = null;
    AgletProxy myProxy;
    Vector myTradingVec;
    String myTradeStatus, myCommodity;

    public void onCreation (Object obj)  // constructs a visiting trader agent
    {
        myProxy = this.getProxy ();
        myTradingVec.addElement (myProxy);
        myTradingVec.addElement (myCommodity);
        myTradingVec.addElement (myTradeStatus);
        myLookup = new Lookup (myTradingVec);
        addMobilityListener (myLookup);  // adds a mobility listener when arriving at a new market
    }

    public void run ()
    {
        myLookup.enter (); // starts entering the group for trading
        if (_everythingDone)
            try
            {
                dispatch (next_destination); // triggers the onDispatching method before leaving
            } catch (Exception ex) {}  
    }
}
A.2 Agent-side comparison-shopping pattern

The shoppingAgent class

```java
public class shoppingAgent extends Aglet {
    Routing aRouting = null;

    public void onCreation (Object obj) {
        try {
            aRouting = (Routing) obj; // gets the Routing reference after creation
            aRouting.refAgent (this); // initialises the routing module and goes to the first destination

            addMobilityListener (new MobilityAdapter () { // instantiates a mobility listener
                public void onArrival (MobilityEvent me) {
                    try {
                        aRouting.compare (); // starts comparison-shopping task
                    } catch (Exception e) {};
                }
            });
        } catch (Exception ex) {};
    }
}
```

The Routing class

```java
public abstract class Routing implements Serializable {
    AgletProxy myAgentProxy;
    Vector myRoutingVec;

    public Routing (Vector routingVec) { // constructs the Routing class
        myRoutingVec = routingVec.clone ();
    }

    public void refAgent (Aglet shoppingAgent) { // holds a reference to the shopping agent
        myAgentProxy = shoppingAgent.getAgletContext ().getAgletProxy (shoppingAgent.getAgletID ());
        goWhere ();
    }

    protected void move (URL destination) throws Exception {
        if (destination != null) {
            myAgentProxy.dispatch (destination); // dispatches the agent to its destination
        } else { myAgentProxy.dispose (); // or disposes it remotely
    }

    public abstract void compare (); // abstract method to be implemented
    public abstract void goWhere (); // abstract method to be implemented
}
```
The myRouting class

```java
public class myRouting extends Routing {
    AgletProxy marketProxy;
    Vector myRoutingVec, myShoppingVec;

    public myRouting (Vector routingVec, Vector shoppingVec) // constructs the myRouting class
    {
        super (routingVec);
        myRoutingVec = (Vector) routingVec.clone ();
        myShoppingVec = (Vector) shoppingVec.clone ();
    }

    public void compare () // implements specific comparison algorithm, for example
    {
        try
        {
            marketProxy = (AgletProxy) getAgletContext ().getProperty ("Market");
            Double returnPrice = (Double) marketProxy.sendMessage (new Message
                ("Query", (String) myShoppingVec.elementAt (0)));
        }catch (Exception e) {}
        if (returnPrice > myShoppingVec.elementAt (1))
        {goWhere ();}  // if the price is not satisfied, continues the trip
        else {move (myRoutingVec.elementAt (0));}  // otherwise, stops the trip
    }

    public void goWhere () // retrieves the next destination
    {
        if (myRoutingVec.size () > 1)
        {
            move ((URL) myRoutingVec.elementAt (1));
            myRoutingVec.removeElementAt (1);
        } else {move (myRoutingVec.elementAt (0));}
    }
}
```

The Creator class

```java
public class Creator extends Aglet {
    public void onCreation (Object obj)
    {
        try
        {
            Vector routingVec = new Vector ();  // initialises the concrete itinerary
            routingVec.addElement ((URL) homeAddress);
            routingVec.addElement ((URL) destination -1);
            routingVec.addElement ((URL) destination -n);
            Vector shoppingVec = new Vector ();  // initialises the concrete shopping items
            shoppingVec.addElement ((String) shoppingQuery);
            shoppingVec.addElement ((Double) expectedPrice);
            getAgletContext ().createAglet (getCodeBase (), "shoppingAgent", new myRouting
                (routingVec, shoppingVec));  // develops a shopping agent
        }catch (Exception e) {}
    }
}
```
A.3 Server-side comparison-shopping pattern

The Collector class

```java
public abstract class Collector extends Aglet {
    AgletProxy policyProxy;
    Vector myVector;
    String shoppingQuery;

    public void onCreation (Object obj) {
        myVector = (Vector) obj;
        policyProxy = (AgletProxy) myVector.elementAt (0);
        shoppingQuery = (String) myVector.elementAt (1);

        try {
            addMobilityListener ( new MobilityAdapter () { // instantiates a mobility listener
                public void onArrival (MobilityEvent me) {
                    try { // sends a message back home reporting its querying result
                        policyProxy.sendOnewayMessage (new Message
                            ("Compare", query (shoppingQuery)));
                    } catch (Exception ex) {}
                });
            } catch (Exception exc) {} }

    public abstract double query (String myQuery);
}
```

The myCollector Class

```java
public class myCollector extends Collector {
    AgletProxy marketProxy;
    Double myResult;

    public void onCreation (Object obj) {
        super (obj); }

    public double query (String myShoppingQuery) {
        try { // gets the market proxy and starts query, for example
            marketProxy = (AgletProxy) getAgletContext.getProxy ("Market");
            myResult = (Double) marketProxy.sendMessage (new Message
                ("Query", (String) myShoppingQuery));
        } catch (Exception e) { }
    }
}
The Policy class

```java
public class Policy extends Aglet
{
    AgletProxy myProxy, myCollectorProxy;
    Vector policyVec, myCollectorVec;

    public void onCreation (Object obj)
    {
        policyVec = (Vector) obj;
        myProxy = this.getProxy;
        myCollectorVec.addElement (myProxy);
        myCollectorVec.addElement (policyVec.elementAt (0));
        myCollectorProxy = getAgletContext ().createAglet (getCodeBase (), "myCollector",
                (Object) myCollectorVec);  // creates a collector agent with its policy proxy and the shopping item
        goWhere ();  // goes to the first destination
    }

    public boolean handleMessage (Message msg)
    {
        if (msg.sameKind ("Compare")) // performs comparison function, for example
            { Double returnPrice = (Double) msg.getArg ();
                if (returnPrice > policyVec.elementAt (1))
                    { goWhere (); }  // if the price is not satisfied, continues the trip
                    else { myCollectorProxy.dispose (); }  // otherwise, disposes it of remotely
                return true;
            else return false;
        }

    public void goWhere ()  // retrieves the next destination
    {
        if (policyVec.size () > 3)
            { move ((URL) policyVec.elementAt (3));
                policyVec.removeElementAt (3);
            } else { myCollectorProxy.dispose (); }
    }

    public void move (URL destination) throws Exception
    {
        if (destination != null)
            { myCollectorProxy.dispatch (destination);}  // dispatches the agent to its destination
        else { myCollectorProxy.dispose (); }  // or disposes it of remotely
    }
}
```

The homeShopper class

```java
public class homeShopper extends Aglet
{
    public void onCreation (Object obj)
    { try  // develops a Policy agent at the local context and hands it all related information
        { Vector compareShoppingVec = new Vector ();
            compareShoppingVec.addElement ((String) shoppingQuery);
            compareShoppingVec.addElement ((Double) expectedPrice);
            compareShoppingVec.addElement ((URL) homeAddress);
            compareShoppingVec.addElement ((URL) destination -1);
            compareShoppingVec.addElement ((URL) destination -n);
            getAgletContext ().createAglet (getCodeBase (), "Policy", (Object) compareShoppingVec);
            }catch (Exception e) { }
    }
}
A.4 Itinerary-balancing pattern

The doubleCheck class

```java
public class doubleCheck extends MobilityAdapter {
    AgletProxy myLogBookProxy;
    Vector myLocationVec, myPlannedVec;
    URL myNextDestion;

    public doubleCheck (AgletProxy myProxy, AgletProxy logBookProxy, String myJobID)
    {myLogBookProxy = logBookProxy; }   // initialises the listener

    public void onArrival (MobilityEvent me)
    {
        myLocationVec.addElement (myJobID);
        myLocationVec.addElement (me.getProxy.getAddress ());
        try   // reports its current location when arriving at a new destination
        {
            myLogBookProxy.sendMessage (new Message ("doUpdate", (Vector) myLocationVec));
        } catch (Exception ex) {} }   // reports its current location when arriving at a new destination

    public check ()  // retrieves a planned itinerary vector and wait for balancing
    {
        myPlannedVec.addElement (myJobID);
        myPlannedVec.addElement ((URL) retrieved-destination-1);
        myPlannedVec.addElement ((URL) retrieved-destination-n);
        try  // contacts the logBook to balance its itinerary with other group members
        {
            myNextDestion = (URL) myLogBookProxy.sendMessage (new Message ("doBalance", (Vector) myPlannedVec));
            myProxy.dispatch (myNextDestination);
        } catch (Exception ex) {} } 
}
```

The Employer class

```java
public class Employer extends Aglet {
    doubleCheck myDoubleCheck = null;
    AgletProxy myProxy, myLogBookProxy;
    String myJobID;

    public void onCreation (Object obj)  // constructs and initialises an employer agent class
    { myProxy = this.getProxy;
      myDoubleCheck = new doubleCheck (myProxy, logBookProxy, myJobID);
      addMobilityListener (myDoubleCheck); // initialises and plugs in a mobility listener
    }

    public void run ( )
    { myDoubleCheck.check();}
}
```
The logBook class

```java
public class logBook extends Aglet {
    Hashtable myLogBookHash;
    Vector myItineraryVec, myHistoryVec, myFutureVec, locationVec, plannedVec;

    public void onCreation (Object obj) // constructs and initialises an logBook class
    {getAgletContext ().setProperty ("logBookProxy", getProxy ()); }

    public boolean handleMessage (Message msg)
    {
        if (msg.sameKind ("doUpdate")). // deals with "doUpdate" message
        {
            locationVec = (Vector) msg.getArg ();
            myItineraryVec = myLogBookHash.get (locationVec.elementAt (0));
            myHistoryVec = myItineraryVec.elementAt (0);
            myHistoryVec.addElement (locationVec.elementAt (1)); // updates the history record
            return true;
        }
        else if (msg.sameKind ("doBalance")). // deals with "doBalance" message
        {
            plannedVec = (Vector) msg.getArg ();
            myItineraryVec = myLogBookHash.get (plannedVec.elementAt (0));
            myHistoryVec = myItineraryVec.elementAt (0);
            myFutureVec = myItineraryVec.elementAt (1);
            for (int i = 1, i < plannedVec.size (), i ++) // takes one of the planned destinations
            {
                for (int j = 0, j < myHistoryVec.size (), j ++). // checks whether it is in the history vector
                {
                    if (!plannedVec.elementAt (i).equals (myHistoryVec.elementAt (j)))
                    {
                        for (int k = 0, k < myFutureVec.size (), k ++). // checks whether it is in the future vector
                        {
                            if (!plannedVec.elementAt (i).equals (myFutureVec.elementAt (k)))
                            {
                                myFutureVec.addElement (plannedVec.elementAt (i)); // adds into the future vector
                            }
                        }
                    }
                }
            }
            msg.sendReply ((URL) myFutureVec.elementAt (order ++)); // retrieves one from the future vec
            return true;
        }
        return false;
    }
}
```
Bibliography

Albers, S. 1996, ‘Competitive Online Algorithms’, *BRICS (Basic Research in Computer Science) Lecture Series LS-96-2*, Department of Computer Science, University of Aarhus, USA.


