Infrastructure of agent-based application service providing model

Chong, Chee Seng

2005

Chong, C. S. (2005). Infrastructure of agent-based application service providing model. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/4150>

<https://doi.org/10.32657/10356/4150>

Nanyang Technological University

Downloaded on 26 Apr 2023 15:51:51 SGT

Infrastructure of Agent-Based Application Service Providing Model

Chong Chee Seng

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Master of Engineering

2005

HF
5548.32
C548
2005

Acknowledgements

I would like to express my gratitude to Dr. Miao Yuan who is a very responsible and supportive supervisor, not to mention his boundless patience. His guidance and encouragement made my graduate study bearable. He is also a caring supervisor who is concerned about the well fare of his students.

I would like to thank Prof. Robert Gay for his suggestions on certain aspects of my work, and his kindness in advising me about the scholarship opportunity. Without his prompt and thoughtful responses to my email enquiries, I might not have come to this wonderful country for my studies.

Also, I wish to extend my appreciation to NTU and the ASP Center for granting me a scholarship and the opportunity to do research. I would like to thank ICIS for the provision of the necessary computing facilities for the research.

I would like to thank Ms Li Bing. With her assistance, I was able to carry out the implementations more smoothly.

Last but not least, I would also like to extend my appreciation to everyone who in one way or the other contributed to the successful completion of this project.

Table of Contents

Acknowledgements	i
Table of Contents	ii
List of Figures	v
List of Tables	vii
Summary	viii
Chapter 1 Introduction	1
1.1 Background	1
1.2 Motivations	3
1.3 Objectives	5
1.4 Contribution	6
1.5 Organization of Report	6
Chapter 2 Literature Review	7
2.1 ASP Application Delivery Infrastructure	7
2.1.1 Remote Services Technology – Citrix MetaFrame	8
2.1.2 Network Computer (NC) Model	9
2.1.3 Issues of Remote Services and Network Computing technologies	10
2.2 Software Agents	11
2.2.1 Definition	11
2.2.2 Agent-Based Applications	14
2.2.3 Software Agents for the ASP Paradigm	15
2.3 Application of Software Agents in Service Provision	17
2.3.1 Agents for Service Discovery and Downloading	17

2.3.2	Agents for Accessing Data Services.....	20
2.3.3	Agents for Accessing Services at Distributed Sites.....	22
2.3.4	Agents for Managing Application Services.....	25
2.4	Task Automation.....	27
2.4.1	Methods of Task Automation.....	28
2.4.2	Software Tools for Task Automation.....	28
Chapter 3 Proposed Agent-based ASP Infrastructure.....		32
3.1	Motivations.....	32
3.2	Designing the Infrastructure.....	36
3.3	Infrastructure of Agent-based ASP Model.....	39
3.3.1	Overview of the Infrastructure.....	39
3.3.2	Application-Oriented Automation (AOA) set.....	40
3.3.2.1	Knowledge Base Subsystem (KBS).....	43
3.3.2.1.1	KBS 1 st Issue – <i>Case</i> Capture	44
3.3.2.1.2	KBS 2 nd Issue – <i>Case</i> Representation	45
3.3.2.1.3	KBS 3 rd Issue – <i>Case</i> Matching	61
3.3.2.1.4	KBS 4 th Issue – Combining <i>Cases</i>	64
3.3.2.2	Case Base Interpreter (CBI).....	77
3.3.2.2.1	<i>Case</i> Navigation.....	78
3.3.2.2.2	<i>Case</i> Interpretation	85
3.3.2.3	Application Maneuver Mechanisms (AMM)	87
3.3.3	Supporting Components set.....	90
3.3.3.1	Language Supports.....	91
3.3.3.2	Communications, Registration, and Logging Services	94
3.3.3.3	Scheduler Services.....	96
3.3.3.4	Agent Shell	97
3.3.4	Core Agents set.....	99
3.3.4.1	Usher Agent	99
3.3.4.2	Coordination Agents.....	101
3.3.4.3	Lord Of The Agents (LOTA).....	105
Chapter 4 Agent-based ASP Model.....		107
4.1	Functionalities in the Agent-based ASP Model.....	107
4.2	Overall Agent-based ASP Model.....	113

4.3	Components of the Model.....	116
4.3.1	Lord Of The Agents (LOTA).....	116
4.3.2	Authentication Agent.....	117
4.3.3	Service Directory Agent.....	118
4.3.4	FTP Agent.....	119
4.3.5	Knowledge Base Agent.....	120
4.3.6	Technician Agent.....	121
4.3.7	ASP Server Pool Agent.....	122
4.3.8	Coordination Agent.....	123
4.3.9	Usher Agent.....	124
4.3.10	Information Dissemination Agent.....	125
4.3.11	Client Agent.....	126
4.4	A Use Case Example.....	127
Chapter 5	Prototype Implementation.....	130
5.1	User Interfaces of Technician Agent and Client Agent.....	130
5.1.1	Technician Agent.....	130
5.1.2	Client Agent.....	134
5.2	Using Technician Agent to create a case.....	136
5.3	Client Agent to Perform Automation.....	145
Chapter 6	Conclusion	153
6.1	Discussion.....	153
6.2	Conclusion.....	155
References		158
Appendix A		167

List of Figures

Figure 2.1	Typical ASP Application Delivery Model.....	7
Figure 2.2	Communication Between Thin Client and Application Server.....	9
Figure 2.3	Agent-environment interactions.....	13
Figure 2.4	Client/server vs. Mobile Agents.....	18
Figure 2.5	Arrangements of agents in DDM.....	21
Figure 2.6	Example of Carchiolo's proposed agent mechanism.....	23
Figure 2.7	APMS connecting users with service providers.....	24
Figure 2.8	Rental of Service Agent.....	25
Figure 2.9	Example of a dependency graph.....	26
Figure 3.1	Proposed Infrastructure of Agent-based ASP Model.....	40
Figure 3.2	Components of Application-Oriented Automation.....	43
Figure 3.3	Elementary diagrammatic representation of a <i>Case</i>	46
Figure 3.4	<i>Case</i> Graph: A more complex <i>case</i> with branchings.....	48
Figure 3.5	Spy++ and SysTree++ tools.....	51
Figure 3.6	Algorithm for traversing all the child windows of an active window	52
Figure 3.7	Example of a dialog box and its list-symptom captured by our program..	53
Figure 3.8	A sample action flow in well-formed XML format	56
Figure 3.9	Example to illustrate an action flow	60
Figure 3.10	Example of a component with unfixed parameters	63
Figure 3.11	Steps for combining 2 <i>cases</i>	67
Figure 3.12	Steps for extracting a real <i>case</i> from a compound case	70
Figure 3.13	A stage with choice.....	71
Figure 3.14	Two extremes of combining <i>cases</i>	73
Figure 3.15	The Case Base Interpreter.....	77
Figure 3.16	<i>Case</i> with branching.....	79
Figure 3.17	A complex <i>case</i>	83
Figure 3.18	A fragment of an action flow XML file (taken from Figure 3.8).....	85
Figure 3.19	GUI program to create Action-Tool Link Information	90

Figure 3.20	Infrastructure's Supporting Components.....	91
Figure 3.21	Language Supports.....	91
Figure 3.22	A sample XML-based message.....	92
Figure 3.23	Core Agents of the proposed Infrastructure.....	99
Figure 3.24	Adding and registering a new agent.....	104
Figure 4.1	Core Functionalities of Agent-based ASP Model.....	109
Figure 4.2	Agent-based ASP Model.....	115
Figure 4.3	Sample Use Case.....	128
Figure 5.1	Technician Agent GUI – Create <i>case</i>	131
Figure 5.2	Technician Agent GUI – Retrieve <i>case</i>	133
Figure 5.3	Client Agent GUI – Install Application.....	136
Figure 5.4	To create a <i>case</i>	139
Figure 5.5	Saving a <i>case</i>	141
Figure 5.6	Having variable value in action flow.....	143
Figure 5.7	A stage with choice.....	144
Figure 5.8	Case ID1 = <My type.Me.Centurion.Version 1 .Program-Install >.....	144
Figure 5.9	Client Agent making use of created <i>case</i>	151
Figure A.1	XML Schema for the action flow XML file	168
Figure A.2	Header file for <i>MyCommObject</i> class	172
Figure A.3	Header file for <i>MyScheduler</i> class	173
Figure A.4	UML Class view of Agent Shell	174
Figure A.5	UML Class view of Authentication Agent	175
Figure A.6	UML Class view of Service Directory Agent	176
Figure A.7	UML Class view of FTP Agent	177
Figure A.8	UML Class view of Knowledge Base Agent	178
Figure A.9	UML Class view of Technician Agent	179
Figure A.10	UML Class view of Coordination Agent	180
Figure A.11	UML Class view of Usher Agent	181

List of Tables

Table 1.1	Breakdown of Total In-House Hosting Costs.....	5
Table 3.1	Functions of CBI: Navigates and Interprets Cases.....	77
Table 4.1	Features of Lord Of The Agents (LOTA).....	117
Table 4.2	Features of Authentication Agent.....	117
Table 4.3	Features of Service Directory Agent.....	118
Table 4.4	Features of FTP Agent.....	119
Table 4.5	Features of Knowledge Base Agent.....	120
Table 4.6	Features of Technician Agent.....	121
Table 4.7	Features of ASP Server Pool Agent.....	122
Table 4.8	Features of Coordination Agent.....	123
Table 4.9	Features of Usher Agent.....	124
Table 4.10	Features of Information Dissemination Agent.....	125
Table 4.11	Features of Client Agent.....	126
Table A.1	AMM's Toolbox Functions.....	170
Table A.2	Link between <mouse_drag> action and its tools	171

Summary

In Application Service Provider (ASP) scenarios, many ASP tasks are often repeated. These tasks include ASP backroom functions and client-side tasks. In view of this situation, valuable human technician manpower and time can be saved if automation mechanisms can be built to deal with repetitive tasks. We believe that software agents can play an important role in this respect, because software agents are an emerging technology for enabling the implementation of repetitive, complex, and distributed tasks (such as those in an ASP setting) in an autonomous way.

Therefore, this thesis proposes an efficient and scalable infrastructure for an agent-based ASP model, with emphasis on ASP side task automation and client-side automation/technical support. The proposed infrastructure incorporates agents into the overall model and enables ASP backroom functions to be easily implemented. Also, the ASP model is scalable and modular. By using a "Knowledge Base plus Agent" approach, client-side task automation and technical support are realized in a client-PC-adaptive manner. We have defined *cases* that are small in size, highly reusable, non-static, and extensible (able to include newly acquired knowledge), for storing captured knowledge. Essentially, a *case* is used to store the steps/procedures taken while performing a newly encountered task, so that the *case* can be retrieved for automatically handling the same task at another PC or at a latter time.

Also, the proposed infrastructure addresses some weaknesses in existing task automation software tools. The weaknesses are: i) lack of a mechanism to classify, group, and distribute automation tasks, ii) automation tasks created are not versatile for handling different users, iii) automation tasks created are very vulnerable to changes in application, and iv) the way to identify a window is not effective.

Prototype implementation of the proposed infrastructure yields encouraging results, which show that our proposal is viable in real-life situation.

Chapter 1 Introduction

This chapter introduces the background, motivations, objectives and contributions of the research project. The organization of the thesis is also laid out.

1.1 Background

The flourishing of Internet and widespread use of personal computers have spawned a generation of PC users that, more heavily than ever before, rely on computer applications and communication software in their work and personal lives. The importance of computer applications is reflected even more in the business community, them being among the most essential components to support business processes.

Crucial business applications are usually sophisticated and costly. Thus, most small and medium enterprises (SMEs) (and even some large multinational corporations) face difficulties in implementing full integration of business applications because of their limited in-house IT capabilities [1].

Hence, the Application Service Providers (ASPs) were spawned. The term ASP was coined in 1998 and it refers to **any company that hosts applications on remote centralized servers, and delivers the hosted applications to its customers over a wide area network (WAN) or the Internet**. Applications are delivered in exchange for a rental fee. For the ASP, a wide range of applications can be hosted, from e-mail and messaging, Web hosting, financial and accounting applications, to the more complicated customer relationship management (CRM), enterprise resource planning (ERP) and supply chain management (SCM) applications ([1], [2], and [3]). In recent years, ASP concept has been borrowed by various types of web-related implementations, such as providing web services through Sun's J2EE or Microsoft's .NET [4]. Our current work mainly applies to the "true" ASPs which provide hosted applications as their core services.

CHAPTER 1

The IDC group (International Data Corporation) had forecasted that worldwide spending on ASPs will reach US\$24 billion by year 2005, up from US\$1 billion in 2000 [5]. Because of economy slowdown, a later report by the same group downsized the forecast to US\$20 billion by year 2006 [6]. Nonetheless, what's more significant is that the ASP market presents a trend of growth. In the U.S. alone, ASP market hit US\$1.8 billion in 2002, representing a 49% growth over the previous year. A survey result that appeared in the Summit Strategies' May 2004 shows that ASPs are growing in acceptance; 50% of SMEs and 41% of large enterprises* project an increase in related expenditures in the following year and a half [7]. Interestingly, a recent development saw IBM acquiring Corio Inc. – a top-rated ASP company (As at February 2005, Corio was listed as the #1 ASP based on reliability, comprehensiveness of service, and feedback from corporate decision-makers [8].) – in a move which will help IBM's hosted application services business [9]. Evidently, ASP is a very popular concept in business IS (Information Systems) solution.

The advantages of ASP have been discussed in numerous articles ([10], [11], [12], and [13]). A shortlist would include:

1. rapid installation
2. reduced downtime
3. elimination of support overhead on the customer's part
4. reduced piracy
5. instant upgrades and a consistent user base
6. reduced cost
7. seamless access to applications
8. application usage monitoring.

As pointed out by [10], lack of in-house IT expertise is the overwhelming reason why users are adopting ASPs.

Since every ASP is invariably a model of "one-to-many" (an ASP serving many clients), therefore many similar tasks are regularly and repeatedly performed by an ASP's staff for different clients. These tasks are such as software installation,

* With ≥ 1000 employees. Half of the survey populations are organizations with ≥ 10000 employees.

upgrading and patching, modifying program settings and configuration, etc. This presents an opportunity for an attempt to conserve time and effort by devising a method to automate those tasks. In this respect, software agents can play an important role.

The software agent concept was formally introduced in 1995 [14]. Now, Agent-Oriented-Programming (AOP) has attracted a lot of interest from the research community and the industry. Software agent technology is one of the popular enabling technologies in the research community for tackling complex distributed software systems (such as an ASP's platform) that require some degree of task automation.

Now the term "software agents" is widely used, but there is no single universally accepted definition. In most contexts, including in our case, a software agent can be defined as programs to which one can delegate tasks, and will carry out their delegated tasks autonomously. And in doing so, the software agent is within an environment which is both the source from which input information is being sensed, and the target on which output actions are being performed. Autonomy is often the essential component of a set of software agent attributes (enumerated in page 12). The idea of an autonomous software agent is meant to automate tasks with minimal user intervention. This convenience lends well to the implementation of various software systems. Thus it is the purpose of this project to exploit the potential of software agents in implementing the ASP model.

1.2 Motivations

As mentioned earlier, lack of in-house IT expertise is the overwhelming reason why users adopt ASPs. Thus if the ASP model is to be successful, it has to instill a high level of confidence in its clients, the confidence that the ASP infrastructure is capable of providing consistent and reliable services, with minimal service interruption and prompt technical support. The ASP infrastructure can be classified into 2 groups: 1) the application delivery infrastructure, and 2) the backroom functions for streamlined operations.

The ASP application delivery infrastructure is responsible for transmitting applications to the user machine over a network using a variety of thin client models. For example, two common approaches are the Citrix MetaFrame and the Network Computer (NC)/Java models [15]. Various readily available hardware and software platforms are offered by major vendors (IBM, Hewlett-Packard, Microsoft, etc) to support the delivery infrastructure, such that technically it can be considered matured.

Other than the application delivery infrastructure, ASPs need to have the backroom functions to help streamline operations, including billing, authentication, directory services, customer account management, application monitoring, client management, client technical support and others [16]. In fact, the backroom functions are actually the core value of an ASP because these are the ASP's proprietary technology, as compared to the hardware and software used for the application delivery infrastructure, or the third-party applications being delivered.

The backroom functions are constantly ongoing. Many of the functions are frequently repeated, especially the client technical support that is usually done via conversations between clients and ASP's technicians at call-centers, or visits to clients' sites by technicians. All these functions require 24/7 (24 hours a day, 7 days a week) operations by technical staff because consistency of service is of utmost importance. Thus, this situation presents an opportunity for employing autonomous software agents to take care of most of the routine tasks. Besides allowing tasks to be automated, software agents can act as "representative" for ASP to remotely carry out chores at the users' computers. The usage of autonomous software agents will potentially save a lot of manpower, time, and unnecessary traveling on the technician's part. In fact, the most expensive component of the IT Total Cost Of Ownership (TCO) is actually the IT labor costs [17].

Cost Item	% of Total In-House Hosting Costs
IT Staff	49%
Facilities	29%
Configuration	14%
Network	6%
Monitoring and Management	4%

Table 1.1 Breakdown of Total In-House Hosting Costs [17]

Also, software agents are an innovative technology for the efficient and intelligent realization of complex, distributed, and highly interactive heterogeneous systems and applications [18]. An ASP's underlying infrastructure is exactly that type of system – complex, distributed and with highly interactive components. Complex tasks can be decomposed into multiple, autonomous components that interact [19]. Agent-oriented decompositions are an effective way of partitioning the problem space of a complex ASP system.

In this work we propose an infrastructure to support our hypothetical agent-based ASP model, with emphasis on ASP side task automation and client-side automation/technical support. We also attempt to improve on current task automation methods by addressing some of their shortcomings, as will be describe in Chapter 2.

1.3 Objectives

The main objectives of this research project is to propose an efficient and scalable infrastructure for implementing an agent-based ASP model, whereby the infrastructure provides automation support for ASP backroom functionalities and tasks. This is in order to save ASP technical staff's time and manpower, by capitalizing on the fact that many of these tasks are regularly repeated.

1.4 Contribution

The main contribution of this thesis is the proposal of an efficient and scalable infrastructure for an agent-based ASP model. The proposed infrastructure enables ASP backroom functions to be easily implemented as agents are incorporated into the overall model, which is highly scalable and modular. Also, by using a "Knowledge Base plus Agent" approach, client-side task automation and technical support are realized in a client-PC-adaptive manner. We have defined *cases* that are small in size, highly reusable, non-static, and extensible (to include newly acquired knowledge). A *case* is used to store the steps/procedures taken while performing a newly encountered task, so that the *case* can be retrieved for automatically handling the same task at another PC or at a latter time. Overall, the proposed infrastructure enables delivery of various client-based services and the savings of human manpower and time.

1.5 Organization of Report

This thesis is organized as follows. Chapter 2 is the literature review that includes a brief introduction to ASP application delivery infrastructure, software agents, existing work on the applications of agents in parts of the service provision field, and various task automation techniques. Chapter 3 presents the proposed infrastructure for agent-based ASP model. It consists of 3 main categories of components, which are: 1) Application-Oriented Automation set, 2) Supporting Components set, and 3) Core Agents set. Chapter 4 proposes an agent-based ASP model, built on top of the proposed infrastructure. Chapter 5 presents the prototype implementation of the proposed infrastructure and ASP model. Chapter 6 concludes the thesis.

Chapter 2 Literature Review

This chapter provides a literature review that includes a brief introduction to ASP application delivery infrastructures, software agents, existing work on applications of agents in parts of the service provision field, and various task automation techniques.

2.1 ASP Application Delivery Infrastructure

Currently in the ASP approach, applications are transmitted to the user machine over a network using a variety of thin client infrastructures. **Figure 2.1** shows a typical ASP delivery model.

Figure 2.1 Typical ASP Application Delivery Model

Applications are hosted on servers in a server farm, at the ASP site. Based on available capacity, a switch will dynamically allocated server access to customers. The actual transmission of applications to customers is done via the Internet or private WAN* connection.

There are two common infrastructures for the ASP delivery model: 1) the Remote Services technology using Citrix MetaFrame, or 2) the Network Computer (NC)/Java

* A private WAN is a network a multiple locations (consisting of, for example, an ASP application center and multiple client locations), with fixed, dedicated, and reliable (usually high speed) connectivity between the locations.

CHAPTER 2

models. Other than these two methods, there are other ways of distributing applications. They include local area network (LAN) file servers with mapped drives, X Windows, and Microsoft Terminal Services. However, these systems require high-bandwidth networks and are not practical over WAN links and thus are not suitable for ASP. This is because most leading ASPs operate over whole countries (even across international borders), with data centers located at different places worldwide. An even newer proposal of software application and service delivery model using advanced networking technology was presented in [20], which discusses Web service provision in the Open Grid Service Architecture. Even though these alternative methods of distributing applications may prove to be useful computing models in certain situations, they have not been adopted by the major ASPs [15]. Thus, we discuss about Citrix MetaFrame and the Network Computing models in the following sections.

2.1.1 Remote Services Technology – Citrix MetaFrame

Remote Services technology is one of two most common approaches to the ASP application delivery model. It is essentially a server-based computing model in which applications are deployed, managed, supported and executed 100% on a server. It uses a multi-user operating system and a method for distributing the presentation of an application's interface to a remote client device [21]. This server-based computing model was suggested by Furht et al in [22] for ASP.

Two pieces of software are needed to enable Remote Services. Software is needed on the application server to service the remote clients. Very thin and generic software is needed on the remote client machine to provide the remote application's interface. The sophisticated server-side software supports "farms" of application servers, while doing load balancing to efficiently handle large loads. The client-side software is very lightweight and can be easily installed. [23] Applications run at the application servers only, with screen images being transmitted from the server-side to the client-side while user inputs (keystrokes, mouse movement data) are transmitted in the opposite direction.

Citrix is now the leading vendor of remote services technology, with its MetaFrame product [24]. The Citrix Server is most commonly used with Microsoft

CHAPTER 2

Windows applications but also supports a number of versions of Unix. The Citrix thin client is called the Citrix Independent Computing Architecture (ICA) client.

Figure 2.2 Communication Between Thin Client and Application Server

Figure 2.2 shows data communications between the client and the server using the Citrix MetaFrame remote services technology. Only keystrokes, mouse movement data, and screen changes are transmitted across the wire rather than actual code or data. This protocol consumes less than 20KB of bandwidth, regardless of application complexity [23]. This is a rather low requirement of bandwidth, which could be provided by using even the modest modem dial-up connection, not to mention the now widely available broadband connections. Overall, application performance is normally good, even over extremely poor connections, and can be indistinguishable from a local application given decent connections. No-compromise applications and even graphics manipulation can be executed with excellent performance.

Most operating systems (old DOS-based PCs, Unix workstations, MS Windows PCs) can act as the thin client, because Citrix client can run on many platforms and uses few resources. The hosted applications in this remote services delivery model are written the same way as traditional desktop or client/server applications, so the model supports most legacy systems that run on platforms for which the Citrix Server software has been written.

2.1.2 Network Computer (NC) Model

The Network Computer technology uses a variety of Web-related technology to display and/or execute applications on a thin client [25]. Applications are dynamically

CHAPTER 2

downloaded from the network into the client. In its most sophisticated configuration, NC systems use the Java model to transmit applications software to the client in bytecode form for client execution. Bytecodes are machine-independent codes that can be run by the client through a middleware called Java virtual machine.

Unlike the Citrix delivery model, initial data transfer can be large, because the application needs to be transmitted to the client. However, once loaded in memory, the application seldom needs to communicate with the server.

Like the Citrix MetaFrame model, the Java approach works with most client computer platforms, because Java virtual machines have been written for most operating systems and are distributed as open source software. Java is especially popular with mobile devices, such as phones and PDA.

Both the Remote Services model and NC model allow applications to sit on the desktop alongside traditionally installed applications. A Web browser can run on either the Remote Services or NC models, to allow access to applications using other browser-based user interfaces. Both models can even be used at the same time.

Both the Remote Services and NC models can be designed to store documents and data either locally at the client machine, or at the application service provider. The difference is in terms of which machine is considered "local" from the point of view of the application. In the Citrix architecture, the application runs at the application service provider site, thus having faster access to provider's servers' hard disks. Access to local client's hard disks is also possible but slower, because transmission across the network is required. In the Java architecture, the application runs locally at the client machine, thus having faster access to the client's hard disks. Data access to the application service provider's servers will require transmission across the network connection.

2.1.3 Issues of Remote Services and Network Computing technologies

Both the Remote Services and NC models have their respective non-crippling problems/issues.

CHAPTER 2

For Remote Services, the total load must be shouldered by the servers because it does not leverage client-side resources (potentially high-end). Remote Services require a client-side application which is an emotional barrier to widespread adaptation by users. Also, Remote Services is not infinitely scalable because of the cost (license fee) which is based on the number of simultaneous client sessions, making it not suitable for horizontal market applications where there could be a great number of simultaneous users.

On the other hand, for NC a drawback is that applications must be written in Java which is an uncommon language for legacy systems (albeit emerging as a popular language for new development). Java applications may have differences in look and feel from local applications, because they are transported across GUIs. Also, NC requires applications to be transmitted to the client-side, so for large/complex applications with many components the download time can be prohibitively long.

Most importantly (as far as our focus is concerned), both Remote Services and NC are essentially application delivery infrastructures and do not have direct support for incorporating technical support or task automation mechanisms. This is the target of our work.

2.2 Software Agents

2.2.1 Definition

The term "software agent" is widely used, but there is no single universally accepted definition. But this is not necessarily a problem, as long as researchers continue to find suitable uses for software agents.

Nevertheless, in most contexts, including in our case, software agents can be defined as programs to which one can delegate tasks, and which will carry out their delegated task autonomously. And in doing so, the software agent is within an environment which is both the source from which input information is being sensed, and the target on which output actions are being performed. Autonomy is often the essential component of a set of software agent attributes.

CHAPTER 2

A further classification of the attributes that agents might have (although not all are necessary attributes) is listed below:

- i) **Autonomy [14].** Agents act without direct intervention from human. They have some degree of control over their actions based on changes in the environment. A simple idea is to declare all the methods of an agent as private, thus preventing external objects from forcing the agent to perform anything it doesn't wish to do. External objects only create events or messages in the environment, for the agent to perceive and react to.
- ii) **Socialability [14].** Agents always exist in a community which consists of many other agents. Socialability is the ability of one agent to communicate with other agents with messaging and agent communication languages. This interaction facilitates collaboration among agents to achieve a certain goal.
- iii) **Reactivity [14].** This refers to the ability of an agent to sense its environment (other agents and the agent platform) and act upon it.
- iv) **Proactivity [14].** Other than being able to react to their environment, agents seek to achieve their goals by taking the initiative.
- v) **Personality [26].** An agent inherits a unique identity. There could be many agents spawn from the same model agent, each with the same behaviors. But, they still have distinct identity.
- vi) **Mobility [27].** Some agents are able to migrate from one host platform to another. They are called Mobile Agents. As opposed to process migration systems, mobile agents choose when and where to move. Usually, this mobility is associated with multi-hop migration rather than single hop migration. Mobile Agents are usually deployed in situations where local processing need to be performed at distributed sites, so that the agents can migrate to the target site and have direct access to local data and resources for more efficient execution of their tasks. Many additional functions are needed in an agent platform that supports mobility, including the functions to

CHAPTER 2

suspend and package an agent together with its state, serialize the agent, and send it over. Additional security issues need to be addressed because an agent platform might be receiving mobile agents from unknown sources. In situation where mobility of agent is not necessary, static agents are used instead. Static agents do not migrate. Static agents at different machines communicate by sending message.

- vii) **Persistence [26].** In order to achieve certain goals, agents need to exist in a unique identity and continuously running over long periods of time.

The above-mentioned attributes generally constitute a "weak" notion of agency. A strong notion of agency sees an agent as a computer program that, in addition to having the attributes mentioned above, can be implemented using concepts that are more usually applied to humans [14]. In other words, a strong notion of agency includes intelligence on the agent's part. Intelligence is the amount of learned behavior and possible reasoning capacity that an agent may possess [28]. Difficulty in the research on artificial intelligence means that this aspect of agent still needs more work.

Figure 2.3 Agent-environment interactions

CHAPTER 2

Figure 2.3 presents a software agent model as a black box in an environment. For its operation, a software agent receives input information from its environment, and performs computation based on the input in order to decide its further behavior. Computation is done by referring to pre-defined condition action rules. The rules are actually programming codes added by developers, that tell agents exactly what to do, how to do it, and when to do it [29].

As shown in the figure, an agent usually exists in a community of other agents, mutually interacting to solve complex problems. Here, an agent's environment is the agency of other agents. Agents are not isolated entities [30]. As opposed to classes, which encapsulate methods and attributes, agents are at a higher-level of abstraction which is described in terms of their behaviors instead of methods and attributes. Thus, in agent-oriented programming (AOP), developers consider agents' behaviors when designing complex software.

2.2.2 Agent-Based Applications

Agent-oriented programming is suitable for solving many complex problems, because agents are task-oriented, whereby developers need to define a solution to a problem in terms of agent behaviors. Developers design and think of a solution at a higher-level of abstraction. For example, by encapsulating the communication protocols in an agent, developers need not be concerned about the interaction between agents. Rather, they can focus on defining the actions to be taken by each agent under specific circumstances. The process of finding a solution becomes the process of defining agents' condition-action rules. Application can be represented in natural way, which makes it easier for programmers to understand applications because the roles of the agents are very clear [31].

Distributed computing models and applications made up of numerous interacting components are the most suitable implementation areas of the agent paradigm. Agent technology is well-suited for use in applications that reason about messages received over a network, which is why agent-based approaches are so popular in applications that utilize the Internet. Agent-based applications are endowed with the capability of

CHAPTER 2

distributed, asynchronous and parallel processing. Also, agents are ideal for applications that require high-level of automation. It is the inherent characteristics of agents (either mobile or static agents) that appeal to the above types of applications. Examples of the most popular fields of applications that benefit from the agent paradigm are: e-commerce, personal assistance, distributed information retrieval, secure brokering, telecommunication network services, workflow applications, monitoring and notification processes, information dissemination and parallel processing [32], [33].

On the other hand, some applications are not so suited for employing agents compared to others, where the agent paradigm is overkill. This is especially true for applications which do not have many interacting parts.

For example, if a website desires to provide an online diary application with an automated SMS (short-messaging service) reminder service, it could be implemented by using any one of the many agent platforms available today. The agent platform would have to be installed on the web server, together with many unneeded services such as agent mobility support, inter agent communications, plus security issues. Or, it could also be implemented in a much hassle-free way by writing a client-server program, with the client at web server retrieving all the diary information and forwarding them to a server program at the mobile phone network service provider computer, which takes care of sending out the SMS's. In this case, the need to automate one simple task does not justify the overhead of adopting an agent platform. A client program which has some degree of automation can do the job.

Therefore, it is important to stress that agents should be viewed as tools to provide better support for applications. The focus should be on the design of the application by using agents as just one of the available tools, rather than to blindly restrain the design of applications to be based on agents [34].

2.2.3 Software Agents for the ASP Paradigm

After looking at the ASP concept and properties of software agents, it is prudent to have an initial evaluation of the viability and suitability of deploying an agent based

CHAPTER 2

ASP model. We believe that the characteristics of software agents lends well to the ASP model in several aspects.

Providing remote technical support at the client side. The fact that agents are autonomous, together with other attributes, contributes to their potential usage in the ASP model for providing IT services. Agents can migrate or be dispatched to an ASP client's computer and take over the whole process of setting up the client-side program (for accessing the server side application). Whenever a technical problem occurs, an agent, which can be dispatched from the ASP or could have already been residing on the client's computer, will carry out the technical support steps for fixing the problem. This feature greatly shortens the response time and saves the travel time that would otherwise be required of the ASP staff. After all, many technical procedures are routinely repeated, which can be assigned to and handled by agents. For instance, when certain settings of a specific application on all ASP clients' computers need to be tweaked, the actual procedures for doing that can be delegated to software agents.

Automation for inexperienced users. Many ordinary end users are inexperienced in the details of application installation, configuration and upgrades, and thus do not welcome the fact that they have to deal with these issues. An agent-based approach has the necessary autonomy for taking over those tasks transparently, based on properly defined rules (often by IT professionals of customer services).

Realizing complex systems. Software agents are an innovative technology for the efficient and intelligent realization of complex, distributed and highly interactive heterogeneous systems and applications [18]. An ASP model is exactly that type of system – complex, distributed and with highly interactive components. The complex model can be decomposed into multiple, autonomous components that interact [19].

Handling of persistent and repetitive tasks. An ASP system as a whole is a collection of various persistent services that cater to the needs of clients. Such services include billing, information management, reporting and logging, client requests processing and handling, etc. These services are repetitive and do not require much innovative contemplation. Software agents can conveniently take over these tasks.

CHAPTER 2

The discussion above highlights some of the potentials of utilizing software agents to derive a more efficient and automated ASP model. An ASP model is considered more efficient if it has some mechanism that contributes to the preservation of time and manpower in its operation (such as remote technical support). Also, an ASP model with a certain degree of automation should be capable of reducing the needs for human involvement in handling routine tasks. These benefits are detailed in a later part (Section 6.1) of this thesis.

2.3 Application of Software Agents in Service Provision

There have been a number of articles on employment of software agents in some particular areas of the service provision field, which are discussed in this section.

Normally in service provision, there are four aspects that need to be addressed. The first aspect is service discovery and downloading (**Section 2.3.1**), so as to find and then (optionally) retrieve the desired services. Mechanisms are also needed to access data services (**Section 2.3.2**) or to access application services (**Section 2.3.3**) at distributed sites. The fourth aspect is the management of application services (**Section 2.3.4**) deployed over large IP networks.

2.3.1 Agents for Service Discovery and Downloading

The migration capability of mobile agents has been utilized in providing service/software discovery and downloading for mobile devices. A mobile agent is able to move from a network server to another while having access to required resources for executing itself.

Mobile agent is useful for service discovery/browsing because it can play a significant role in intelligent information retrieval. A challenging issue in services provision, especially for mobile devices, is enabling the user to choose the desired service from among possibly thousands of available services. Doing this search manually requires prolonged duration of access to the network. But network connection, the associated power consumption, and bandwidth are expensive resources for mobile

CHAPTER 2

devices. Mobile agents can help reduce the duration of network connection by performing local interaction with the user, and then migrate to the service provider network to search for and gather information. The connection can be temporarily closed until reconnection is needed later when the mobile agent finishes its mission.

The advantage of using mobile agents to search for and gather information is better illustrated with **Figure 2.4**, which depicts remote accesses throughout a process where a user is trying to select a desired service. The black areas represent brisk periods of remote access.

Figure 2.4 Client/server vs. Mobile Agents [36]

In the traditional client/server approach, many spurts of data transfer occur because bits of information are regularly downloaded to the client in response to each user's action (for example, mouse clicking on links). These frequent occurrences of data transfer means that a constant network connection needs to be maintained throughout the service-selection process. Whereas by using mobile agents to encapsulate user's preferences/instructions, the total number of remote accesses needed is reduced dramatically due to the local interaction between the mobile agent and user, and local interaction between mobile agent and remote data source. Occasionally, the mobile agent migrates back to the client, bringing intermediate results. Network connection need not be maintained between A to B, C to D, and E to F. Thus even though the total time

CHAPTER 2

required for the service-selection process might be longer, but the charges for network connection is reduced.

The benefit of using mobile agents for software downloading is due to the fact that the movement of code portions can be done transparently. Various commercially available agent platforms provide transparent mechanisms for code mobility, which are generally encapsulated in a single primitive or method.

In [35], a framework based on Java, CORBA and mobile agents that helps mobile telecommunication companies to supply mobile users with downloadable value-added services (VAS) was proposed. The combination of those distributed software technologies was required to cater for the needs of a variety of network equipment and devices. In their model of VAS provision, the service implementation can be divided into a movable mobile terminal part and a stationary server part. The former part is downloadable to mobile terminals and executed there, thus written in Java which is a machine- and operating system-independent language. The server side uses CORBA to provide language support, scalability, and openness to ensure interoperability between various VAS providers. Software agents are used to realize two functions: Service Downloading and Service Discovery. For Service Downloading, the cooperation between Service Provider Agent (SPA), Terminal Services downloading Agent (TSA), and Mobile Provider Agent (MPA) transfers a service from VAS provider to its final destination mobile terminal while eliminating the need for ongoing interaction between VAS provider and mobile terminal. For Service Discovery, the user can relate his wishes/request to a mobile agent called Service Discovery Agent (SDA). SDA will subsequently roam through the sites of various VAS providers. After finding a satisfactory match, SDA returns the results to the user. This scenario is attractive because the required task is performed asynchronously, thus there is no need to maintain a network connection over the unreliable and costly wireless link for the whole duration of the task.

In [36, 37, and 38], an agent-based implementation of a software retrieval service was presented. Some of its ideas are conceptually similar to [35]. From the implementation point of view, the service was based on the use of mobile agents because:

CHAPTER 2

1. they allow the management of knowledge used to solve problems and to take appropriate decisions, and
2. they are able to travel from one computer to another

From the point of view of the functionalities that the system provides:

1. it incorporates a majordomo agent that acts on behalf of the user, and so relieves the user from many tasks
2. it incorporates a ontology-based mechanism that catalogs software and takes care of new releases and new products
3. it helps users in the task of finding the adequate software, and
4. it incorporates mechanisms to get concrete software, carry it to the mobile computer and install it

The problem targeted was how users (especially ones with mobile devices) can find and acquire the exact program needed and suitable for their computer, while trying to shorten the total time in which the remote connection needs to be open.

2.3.2 Agents for Accessing Data Services

Because of the autonomy characteristic of software agents, they are popular candidates for implementing various data related services on the World Wide Web and mobile telecommunications networks. The main reason is that their capability to execute and to carry out tasks on their own means that they are highly suitable for asynchronously operating at distributed locations on the web.

Distributed Data Mining (DDM) is one area where usage of agents is popular. According to [39], the most prominent agent-based DDM systems to date are BODHI [40], PADMA [41], JAM [42], and Papyrus [43]. As shown in **Figure 2.5**, their common usage of agents is to have either a mobile or a static agent running locally at each different data site, and a central facilitator agent. The mining process is distributed to the local agents. The central agent is responsible for coordinating the data mining tasks of

CHAPTER 2

local agents, collect the partial mining results from all the local agents, and then perform aggregation to produce the final result. In short, agent-based DDM systems aim to integrate knowledge which is discovered out of data at different geographically distributed network sites with a minimum amount of network communication, and maximum amount of local computation. Another proposal in [44] goes one step further by developing an architectural model to support on-line DDM services to be provided under the ASP concept.

Figure 2.5 Arrangement of agents in DDM (adapted from [39])

Another application of software agent for data services is the usage of mobile agents for World Wide Web distributed database access. In [45], an agent-based framework was proposed as a better substitute for Java applet/JDBC based database access and connectivity over the Internet. Similar to the central idea behind agent-based DDM systems, in the context of DBMS access, a team of stationary and mobile agents enables asynchronous database operations to go on at distributed sites on the web and frees the web client to perform other tasks. All intended interactions between the client and an SQL server machine are encapsulate within an agent. The responsibility of loading the JDBC driver at the SQL server is passed to the mobile agent (instead of being performed by the client), thus avoiding the unnecessary downloading and initialization of JDBC drivers at the client side applet. The end result was a more flexible, scalable, and

CHAPTER 2

robust framework than the Java applet/JDBC based database connectivity. The effect on performance was demonstrated to be quite significant, especially for slow or expensive networks.

There are many other examples of agent-based frameworks for accessing data services such as data locker services (secure data storage on the web), traditional WWW access, e-mail, yellow pages services, etc. as proposed in [38,46]. Still, the basic idea is the combination of the agent technology with the indirect model. In the indirect model, functions, services, and mobile agents are all executed in an intermediary element (such as a central web server or a Gateway Support Node (GSN) of a cellular network) so that the mobile computers are relieved of many tasks.

It is clear that software agents' capability to execute independently in an asynchronously way is the most desirable characteristic when they are being applied for accessing distributed data locations. Mobile agents (instead of stationary agents) were used in most of the above mentioned applications mainly because mobile agents can be assigned route itineraries. This allows a mobile agent to roam through several distributed sites on the web before returning to the origin/client. This is not possible if merely a downloadable stationary agent was used.

2.3.3 Agents for Accessing Services at Distributed Sites

The WWW is a convenient way to distribute and provide services to anyone in the world who has an Internet connection. Various services can be hosted and offered by service providers, such as e-mail, airplane ticket booking, information queries, etc. Besides traditional web-based techniques of accessing the services, software agent is another option. Agent technology, as discussed in [47,48], possesses particularly interesting features for the development of open systems for the provision of services in an electronic market.

An agent-based platform was proposed for service providers in [49]. As depicted in **Figure 2.6**, in the platform, a virtual service provider (VSP) consists of a pool of heterogeneous non-dedicated hosts.

Figure 2.6 Example of Carchiolo's proposed agent mechanism [49]

Each host provides >1 services. One same type of service is provided by >1 host. Each service is represented as a stationary service agent (SA) at the host. Mobile client agent migrates to the VSP and receives response from only one of the several hosts (for ex. Host 1) which has the desired service. After completing its task (by interacting with the appropriate and active SA1), the client agent returns to the client with the results of the service. Actually, the focus of this article is on having distributed and redundant locations for reliable services, and having a token ring agent to determine which host will become active for receiving client's request for service.

Another example of agent accessing services hosted at distributed locations is in [50], which presents an integrated management architecture for thin-client (mobile devices) purpose, called Agent and Profile Management System which bridges wireless network with the Internet.

Figure 2.7 APMS connecting users with service providers (adapted from [50])

As shown in **Figure 2.7**, Agent and Profile Management System (APMS) is the intermediary between users and the many service providers. It provides an integrated environment to transmit the newest service information and data to mobile users. User profiles and service lists are maintained at APMS. Service providers also store their service list and service agents there, while users select and then download service agents via APMS. Once downloaded, a service agent is the front-end through which the user connects to the service provider. Services can be utilized by sending XML-RPC request and receiving XML-RPC response from the service provider. XML-RPC is a communication protocol working on both wireline and wireless network. Actually, the main idea of this article is the introduction of a bridge between the wireless network and the internet-based service providers. The value-added services are the user profile management and centralized location of service lists.

In [51], the article demonstrates the concept of creation of new functions by combining, coordinating and facilitating other already-existing functions provided by many applications in the network. Each new composite function is encapsulated in a mobile Service Agent, and the Service Agent is provided to users in a rental mode. The actual content of the new function is the control flow and procedures of making use of the already-existing applications.

For that purpose, an Agent Wrapper is written for each existing to-be-used application. Agent Wrapper enables an application to have some autonomous behavior, and manages the application's status and calls the application's functions if a request

from a Service Agent is not-rejected. It is also the platform for the migration of Service Agents.

Figure 2.8 Rental of Service Agent (adapted from [51])

As shown in **Figure 2.8**, a Service Agent moves to each of the machines (one after another, according to itinerary) which host the needed applications, interacts locally, collects results, and in the end shows the final result to user. Comparing with the distributed object models, instead of having each application (or object) directly connected to the other application and calls its methods one after another, this proposed framework has all the service procedures/control flows collected in a Service Agent. Changes to a service only concern a Service Agent and do not affect the various applications.

In the above examples of agents accessing services at distributed sites, actual execution of the services still occur at the providers' site while agents play the role of the services' front-end interface with user.

2.3.4 Agents for Managing Application Services

Software agents have long been used for network management tasks, such as in the Simple Network Management Protocol (SNMP) to manage TCP/IP networks. They are used in a poll-response scheme to monitor the operational status of various devices on

CHAPTER 2

the network. However, the role of software agents can also be extended to take part in the management of application services deployed over large IP networks.

An approach for end to end management of application services was proposed in [52, 53]. The objective was so that network service providers, who operate commercial network management platforms, can offer new functionalities by enhancing their existing infrastructures to include management of application services.

Application services that are deployed over the network (WAN) comprise many distributed components. An important step is to perform dependency analysis to determine which component of the application service depends on what resources of the network, such as in **Figure 2.9**. A typical customer eCommerce application would directly or indirectly depend on the following list of network resources: Web Application Service, Web Service, Database Service, Name Service, IP Service, and Operating System. With the dependency information available, each application management agent associated with each application service will be able to construct a list of network resources the application depends on.

Figure 2.9 Example of a dependency graph [53]

Then, the agent will request event notifications from the network management system's Mid-Level-Managers (MLM) which are responsible for monitoring the network resources. The agent builds a view of the service that it maintains. With the service view, a service manager can observe the status of the service and do typical drill down

operations for troubleshooting. The role of software agent in this scenario is to constantly query for updated resource information and to construct a unified service view for the service manager to watch over.

2.4 Task Automation

One of the main purposes of adopting software agents in our project is to automate various ASP client-side and server-side tasks (such as application installation, performing configuration steps, information dissemination, client authentication, etc). In this section, we describe some existing methods of task automation in software.

In the broadest term, computer task automation is the functioning/operation of tasks with minimal human intervention. From the implementation point of view, automation can be realized at different software levels, depending on the complexity of the task to be automated.

According to the definition of task automation above, a computer software itself is also an automation entity. It is a proper collection of computer language instructions to be carried out one by one in pre-dictated sequences. The exact sequence depends on some external factors, such as keyboard and mouse inputs. A computer program automatically executes a sequence of computer instructions. For example, during the initial stages of opening a Windows-based application, the program will perform a series of operations, such as loading some information into memory, drawing a frame of the window, painting the window, drawing the font characters, painting the controls, etc.

Moving to a higher level of implementation, there is the application-oriented task automation. At this level, computer programs and the operating system are manipulated directly to accomplish various tasks, such as the opening of two web pages on every morning, running the antivirus scan at a certain time, installing an application on behalf of user, or even troubleshooting application problems. The methods for task automation at this level are such as batch files, scripts, record & playback, macros, and wizards.

CHAPTER 2

2.4.1 Methods of Task Automation

There are various methods of task automation. The simplest is a batch file, which is a text file that contains a sequence of commands for a computer operating system. The automation capabilities of a batch file are very limited, being confined by the available commands supported by the operating system. A macro is a file that contains a sequenced list of actions: instructions/commands, keyboard strokes, and mouse action inputs. Macro files are built by either writing a script or by a recording utility. Besides several popular scripting languages, some programs allow users to write macro files using special (proprietary) scripting languages. There are also many software tools that enable users to build macros by using a recorder utility, which is capable of recording all the actions done by a user with the keyboard and mouse, and eventually generating the corresponding macro file. We are also familiar with the "wizard", which is a software program specifically written to implement a certain task. Most often than not, the user is required to provide some feedback to the wizard so that it can successfully complete the intended task.

The methods of task automation described above are usually sufficient for ordinary low level automation usage by the casual end users. But for ASP scenarios, they are inadequate because of several shortcomings. They lack a mechanism to properly classify, group and distribute user-created automation tasks. Also, the created tasks are not versatile / dynamic, plus the fact that they are very vulnerable to changes in application layouts. We will further elaborate on these points in the next section.

2.4.2 Software Tools for Task Automation

There are many commercially available software tools which implement task automation by adopting the macro approach. Some even provide GUI-based task editors with a set of pre-defined commands.

For example, Automize [54] is a scheduling program designed to automatically execute tasks. It has a predefined set of commands/tasks that can be selected and scheduled to execute. But its capabilities are limited by the relatively small set of commands available, and it is not very extensible.

CHAPTER 2

AutoIt [55] is a script-based simple automation tool that can simulate key presses, mouse movements and window commands (maximize, minimize, wait for, etc.) in order to automate windows based tasks. Tasks are created by writing script files. When AutoIt is executed, it reads a specified script file. This script file allows AutoIt to perform a number of functions, including executing programs, sending keystrokes and mouse clicks, window functions, simple text clipboard functions, simple registry functions, simple string and variable functions. Scripts can also be compiled into standalone executables.

EMac [56], Macro Express [57], and AutoMate [58] are quite similar to AutoIt, in that they too automate tasks using macros. Moreover, EMac, Macro Express, and AutoMate provide recorder utilities in addition to scripting for creating macro files. AutoMate also incorporates a GUI-based drag and drop task creator.

The capabilities of some of these task automation software tools are quite formidable for ordinary computer users. However, for an ASP scenario where we wish to automate tasks on both ASP server-side and client-side, these tools seem inadequate in several aspects.

Firstly, there is a lack of a mechanism to properly classify, group, and distribute user-created automation tasks. Tasks created using the software tools are stored individually as single independent entities, each with a user-given name. The tasks are stored in the machine for local usage. However, in an ASP scenario, the number of automated tasks created and stored will be very huge. Also, as the database of automated tasks grows bigger, some of those tasks are bound to be related to each other. We need a systematic method of organizing this storage of tasks and categorizing them accordingly. Then, a mechanism is needed for distributing this collection of tasks to serve clients (on demand). Therefore, a main functionality of our proposed infrastructure is to systematically classify, group and distribute automation tasks (we denote them as "*cases*"), plus invoking the stored tasks for future usage.

For instance, tasks that are created to address different aspects of a same application should be grouped together. These tasks of the same group may have similar intermediate steps. This presents an opportunity to combine tasks. For example, a task

CHAPTER 2

created to print a document in Microsoft Word and another task created to save a document in Microsoft Word have one same step. The same step is "open the File menu". Thus these two tasks could be classified and grouped using the application name, and have the steps combined / compressed during storage. A knowledge base of all the tasks should be set up which allows remote access and retrieval of an appropriate task when needed.

Secondly, automation tasks created with existing software tools are not versatile / dynamic. A task created by user A might not be suitable for user B, even if they both desire the same end result. For instance, the task may consist of a step that keys in user A's name. Thus for ASP usage, we need a generic format for tasks, which can include links to other sources, such as to customer profile database. This is realized in our proposed infrastructure.

Another inadequacy of existing task automation tools is that the created tasks are very vulnerable to changes in applications. For instance, if an application is upgraded to a newer version and results in just one alteration of the application's GUI, then all the previously created macros/tasks for that application might need to be rewritten or recorded again. In our proposal, created tasks are much less vulnerable to changes in application layouts, whereby in our approach the proper execution of an automation task is much less dependent on the GUI layout/positioning of window components (buttons, text boxes, etc). Thus our approach is, to a certain extent, more robust. Though, major GUI changes (for example, from Windows 98 to Windows XP) cannot be accounted for.

Furthermore, the current way of identifying a window is not effective. Most of the software tools have a crude method to monitor for the emergence of any specific windows, which is by checking the windows' title. This is not an effective method because an application can have many different windows with the same title. For example, the installation program of application **ABC** will display several different dialog boxes (windows) one by one, all with the same title "ABC Installation". Instead of doing this, we use an approach to capture the context of a window by gathering information regarding all the components in a window. By this way, windows are much more distinguishable from each other.

CHAPTER 2

In this project, task automation is one of the main focuses of our proposed infrastructure of agent-based ASP.

Chapter 3 Proposed Agent-based ASP Infrastructure

In this chapter we describe the proposed infrastructure that supports the Agent-based ASP model (the model is described in **Chapter 4**). Firstly, we explain the reasons why do we want to design an infrastructure for agent-based ASP model, even though there exist scores of other multi-agent platforms. We will point out the main distinction of our infrastructure which suites our proposed agent-based ASP model. Then, we give an overview of the whole infrastructure set which consists of 3 main categories of components. Finally we describe each category of the infrastructure set, giving specific emphasis to their functionalities and the relationship between components.

3.1 Motivations

This section will look into what is the most crucial component or aspect of an ASP value proposition. After doing that, we can then decide on the focus of the capabilities of our proposed infrastructure.

The ASP paradigm offers many benefits to users. Numerous articles have discussed about the advantages (from users' point of view) of adopting the services of an ASP, including:

1. **Rapid installation** – ASP software can be used almost immediately, bypassing the time-consuming and potentially troublesome client installation. There's only a slight delay because of integration with other client system and configuration to meet specific requirement. But this is much faster than client installation of the required software.

CHAPTER 3

2. **Reduced downtime** – Most ASPs do a better job of ensuring 24/7 application availability than customers could because they have specialized trained staff and also depend on their products' reliability to stay in business.
3. **Elimination of support overhead on the customer's part** – Users don't need to employ expensive administration and support staff to operate complex software installations and the equipment required to run them.
4. **Instant upgrades** – Bug fixes and new features can be deployed in a timely manner by ASP, without clients having to worry about them.
5. **A wide range of choices** – The Internet gives users a convenient way of accessing any rental application of their choice, online.
6. **Reduced cost** – In addition to the elimination of support overhead, clients also do not have to buy the business applications (most of which are very expensive) which they are going to use. They just need to rent the access rights to the applications, owned by ASP.

Among the many benefits of using an ASP, when a user decides to adopt an ASP it is the ASP's ability to overcome the lack of in-house IT expertise which proves to be more than twice as important as other factors. In other words, technical support provided by an ASP is the most crucial aspect. The graph below (extracted from [10]) proves this point.

Reasons users adopt ASPs [10]

CHAPTER 3

Therefore, if the ASP model is to be successful, it has to instill a high level of confidence in its clients, the confidence that the ASP infrastructure is capable of providing consistent and reliable services, with minimal service interruption and prompt technical support. The ASP infrastructure can be classified into 2 groups: 1) the application delivery infrastructure, and 2) the backroom functions for streamlined operations.

The ASP application delivery infrastructure is responsible for transmitting applications to the user machine over a network using a variety of thin client models. For example, two common approaches are the Citrix MetaFrame and the Network Computer (NC)/Java models [15]. Various readily available hardware and software platforms are offered by major vendors (IBM, Hewlett-Packard, Microsoft, etc) to support the delivery infrastructure, such that technically it can be considered matured. Currently, most major ASPs (Corio, USInternetworking, etc) have their applications hosted in large data centers throughout the world, with state-of-the-art hardware and software platforms and tight physical security.

Other than the application delivery infrastructure, ASPs need to have the backroom functions to help streamline operations, including billing, authentication, directory services, customer account management, application monitoring, client management, client technical support and others [16]. In fact, the backroom functions are actually the core value of an ASP because these are the ASP's proprietary technology, as compared to the hardware and software used for the application delivery infrastructure, or the third-party applications being delivered. Emphasis should be given towards maintaining a high-level of availability of backroom functions, as these are the overwhelming factors being considered by clients.

The backroom functions are constantly ongoing. Many of the functions are often repeated, especially the client technical support. All these functions require 24/7 operations by technical staff because consistency of service is of utmost importance. Service interruption should be minimal and technical support should be prompt. As explained previously, prospective clients are most influenced by those factors, before

CHAPTER 3

deciding on whether or not to take the risk of adopting an ASP and thus handling over part of all of its IT processes. The nature of backroom functions – routine and frequently repeated – presents an opportunity for employing autonomous software agents to take care of most of the routine tasks. As have been covered in **Chapter 2**, software agents are programs to which one can delegate tasks, and which will carry out their delegated tasks autonomously.

Besides allowing tasks to be automated, software agents can act as "representative" for ASP to remotely carry out chores at the users' computers. The software agents take over part of the work load of human technicians. To understand the significance of this ability, we need to consider the situation faced by an ASP. The one-to-many relationship, of one ASP to its many clients, translates into a higher requirement for an efficient and potentially large pool of ASP technical staff. These ASP technical personnel are, supposedly, most versed in running, maintaining and troubleshooting the applications that the ASP is providing. This is because they routinely handle the almost same collection of applications being offered by their company – the ASP. Most of the maintenance work and technical support that the technical personnel provide to their clients will be similar and frequently repeated.

In a simple example of application installation, instead of having a technician to personally travel down to the client's site (in cases where applications are still hosted at client machines) to handle the work, this routine installation process could be done through a software agent. This agent is sent to the client site computer, and performs the steps that otherwise would have to be performed by a technician. The usage of autonomous software agents will potentially save a lot of unnecessary traveling and time on the technician's part, and also saves a lot of manpower by providing the same problem solution automatically, in a much shorter time frame.

This role played by software agents is very useful, because the most expensive component of the IT Total Cost of Ownership (TCO) is actually the IT labor costs (please refer to **Table 1.1 Breakdown of Total In-House Hosting Costs**).

Thus we believe that the participation of software agents will provide a lot of flexibility to the ASP's operation. We also believe in designing a mechanism to systematically capture human technician's know-how into a form understandable and actionable by software agents.

Also, software agents are an innovative technology for the efficient and intelligent realization of complex, distributed and highly interactive heterogeneous systems and applications [18]. An ASP's underlying infrastructure is exactly that type of system – complex, distributed and with highly interactive components. Complex tasks can be decomposed into multiple, autonomous components that interact [19]. Agent-oriented decompositions are an effective way of partitioning the problem space of a complex ASP system.

Thus it is worthwhile for a research on the infrastructure of an agent-based ASP model, and its implementation. In this work we propose an infrastructure to support our hypothetical agent-based ASP model, with emphasis on ASP side task automation and client-side automation/technical support.

3.2 Designing the Infrastructure

As have been discussed in the earlier Section 3.1 Motivations, we believe that an ASP model that utilizes the agent technology offers several key advantages. To that end, we are proposing an infrastructure for an agent-based ASP model, with several distinctive concept and components to address our requirements.

Firstly, the proposed infrastructure needs to have an appropriate set of components for enabling automation capability for client application support. Our proposal includes the following components:

- 1) A Knowledge Base, for all the recorded solution *cases* to store best practice solutions for commonly encountered situations,

CHAPTER 3

- 2) Case Base Interpreter, to interpret a *case* and output an agent action-flow list, which is customer-profile specific and customer-desktop appearance specific,
- 3) Application Maneuver Mechanisms, to make use of the action-flow list to simulate actual mouse/keyboard inputs.

Also, we believe that for the ASP scenario, it is suitable to have an infrastructure that supports a hierarchical arrangement of agents. This is because our agent communities are different from the other agent communities in terms of the amount of centralized control and the amount of autonomy granted to individual agents. Many existing agent communities (for example a market place for personal agents to roam and negotiate) have rather loose centralized control. Each agent acts on behalf of its user and only caters to its user's interest while there isn't any mutual goal of the agents as community. Thus those agent communities have agents with high level of autonomy and less centralized control. On the other hand, a service provider by comparison needs to be able to conveniently gain control to all parts of its system so as to strictly maintain predictable system behaviors and performance, especially in regards to complying with Service Level Agreements (SLA). Since there is a tradeoff between control and autonomy, thus there will be slightly less delegation of autonomy to individual agents.

Therefore a hierarchy of coordination agents affords better overall command and view of the agent community at the higher management plane (by human), and this is in contrast to the more-freely executing manner of a mesh of agents. For this purpose, we have specified a registration service (for registration of lower level agent with upper level agents), the coordination agents and the LOTA (Lord Of The Agents, which provides a unified view and control of all agents in the system) agent. Detailed discussions regarding these are provided in the later part of this chapter.

Thus our infrastructure consists of a set of components for enabling automation capability, a set of Core Agents as the basis for a hierarchical arrangement of agents, plus a collection of agent shell and related supporting sub-systems as the development tools for additional utility agents. We will further elaborate on the proposed infrastructure in the following sections.

CHAPTER 3

For implementation purposes, in both the industrial and research community there are already numerous existing multi-agent platforms, which could possibly be used to implement the proposed infrastructure of agent-based ASP model. However in reality most multi-agent platforms have some common characteristics or features that are not desirable in an ASP scenario.

Multi-agent platforms invariably require an agent execution platform to be installed first, before the resident agents can execute. The main reason for having this platform is to have a security sandbox/middleware, within which mobile agents can execute. But for the ASP scenario, security and permission for agents to execute are less of a concern. This is because in the first place, a high level of trust must already exist between ASP and its clients, such that mission-critical services and data are handled by the ASP. Therefore, installing the agent platform will introduce an unnecessary overhead which is prohibitive for widespread adoption, especially from the point of view of customers of ASP because a client-side agent needs to run at each of their machines.

As well, some features offered by the existing multi-agent platforms are not required in our model, for example the all too famous agent mobility capability and the Directory Facilitator (DF). The former – Agent mobility – is not required as our model involves only static agents that do not migrate on-the-fly. As described in Section 2.2 Software Agents, mobile agents are deployed in situations that require local processing at distributed sites in a multi-hop migration pattern. Many additional functions are needed in an agent platform that supports mobility, not to mention the security issues. In our proposed model, static agents can adequately perform the tasks required. The latter – Directory Facilitator (DF) – provides "yellow pages" services to agents, and is a mandatory requirement specified by the FIPA Agent Reference Model and thus is provided in all FIPA-compliant agent platforms. But, we do not need DF because an ASP model is, for the most part, not a fluctuating system where unknown agents are dynamically added or removed (typical of mobile agents). This dedicated model means that each agent inherently knows about and expects the existence of other specific agents in the model, thus diminishing the need for a DF. In other words, under the nature of the ASP scenario, the agent community is less of an open system. We desire to exclude these

CHAPTER 3

surplus and unneeded features of the existing multi-agent platforms because they are a drawback for client PC performance.

Therefore we decided not to make use of the existing multi-agent platforms for implementing our infrastructure of agent-based ASP model. The components of our infrastructure are developed by using mostly the Microsoft Visual Studio integrated development environment.

We would also like to point out that our focus is on the ASP model functionalities, not on the agent paradigm and associated issues (security, mobility, lifecycle, coordination, etc.). Agents become just part of the tools to develop the distributed system. It is our intention to incorporate agent technology into ASP, and not to contrive another whole new multi-agent platform.

3.3 Infrastructure of Agent-based ASP Model

3.3.1 Overview of the Infrastructure

As shown in Figure 3.1, the proposed infrastructure consists of 10 components. The components can be conveniently classified into 3 categories:

- 1) Application-Oriented Automation (AOA) components set. This includes the Knowledge Base Subsystem (KBS), Case Base Interpreter (CBI), and Application Maneuver Mechanisms (AMM).
- 2) Supporting Components set. This includes the Language Supports, agent shell, communications services registration services, logging services, and scheduler services.
- 3) Core Agents set. This includes the Lord Of The Agents (LOTA), Usher Agent, and Coordination Agent(s).

Figure 3.1 Proposed Infrastructure of Agent-based ASP

The AOA set empowers the agent-based ASP model to provide application-level automation and thus to deliver technical support to customers in an efficient and prompt manner. This will decrease the need for human technician man power. The supporting components encapsulate the various common agent functionalities, and together with an agent shell and language supports, offer a convenient method to develop and implement all the other utility agents of the overall ASP model. The set of core agents lays the foundation for the hierarchical multi-agent ASP system.

Each of the above-mentioned components will be described in detail in the following sections.

3.3.2 Application-Oriented Automation (AOA) set

One of the most important value propositions of the ASP is the provision of enough skilled staff to solve various technical problems faced by ASP's customers. In order to maintain a high level of responsiveness towards potentially emerging new

CHAPTER 3

problems and the frequently recurring problems, it is crucial for an ASP to hire, train and retrain a large number of skilled IT personnel. Many ASPs, and also many other IT-related companies for this matter, still offers the traditional help desk service to receive distress calls from customers, and to orally describe the recommended solutions to customers through this way. As we know, this is very inefficient. Furthermore, sometimes the ASP technical staff even needs to travel to the customer's site to help them. Most of the time many of the problems faced by different customer; are actually very similar (hence we have the famous FAQ in websites).

In view of this, we believe that a large part of a technician's work can be saved and automated by proposing an approach which methodically:

- 1) records and stores application problems encountered and the associated solutions performed by technician, in a non-customer-specific and non-desktop-appearance-specific way;
- 2) consolidates the stored records by combining records that are "near" to each other;
- 3) in the future retrieves the matching problem-solution set (denoted as a "case") from the Knowledge Base, in response to the same problem being detected at another remote customer's PC;
- 4) navigates and interprets the retrieved case to generate a solution list of action-flows; and
- 5) remotely and automatically carries-out the list of action-flows on the customer's computer

An example would be how to install an application ABC on a client's PC. When application ABC's setup.exe program is invoked for the first time (at any client PC), the human technician at ASP will remotely run the program from start to finish for the client (by using pcAnywhere to remotely control the keyboard and mouse pointer of client's PC). All keyboard and mouse actions by the technician are recorded and stored. A case "Running ABC setup program" is created. Next time when application ABC's setup.exe is invoked at another client's PC, it is detected and the matching case will be retrieved. A

CHAPTER 3

list of action-flows will be generated to be carried out on the client's PC, thus successfully automates the installation of application ABC. The whole automated process should incorporate client's profile information (such as name, organization) and should not be affected by the positioning of setup.exe's dialog boxes.

The collective purpose of the Application-Oriented Automation (AOA) set is to address the 5 issues listed above. AOA is the 1st category in our overall proposed infrastructure. AOA consists of 3 parts, as shown in **Figure 3.2**.

Section 3.3.2.1 Knowledge Base Subsystem covers the 1st, 2nd & 3rd issues. **Section 3.3.2.2 Case Base Interpreter** covers the 4th issue. **Section 3.3.2.3 Application Maneuver Mechanisms** will cover the 5th issue.

Before discussing those parts, we need to define what a "*case*" is. Generally, a problem is associated with a specific goal to be achieved (such as to install an application, how to set the font color of a word processor, etc). We define a "*case*" as a problem plus its associated sequence of solution steps or, in other words, a goal plus its associated sequence of solution steps to reach that goal. Below is the formal definition of a *case*.

Definition I:

A *case* C_G is defined as a 3-tuple $\langle G, S, AF \rangle$ where:

- 1) G is the goal to be achieved.
- 2) $S = \{S_1, S_2, S_3, \dots\}$ is the set of all possible stages before reaching G
- 3) $AF = (AF_{(1-2)}, AF_{(2-3)}, AF_{(3-4)}, \dots)$ is the set of action flows ($AF_{(A-B)}$) to move from one stage (S_A) to another stage (S_B).

Figure 3.2 Components of Application-Oriented Automation

3.3.2.1 Knowledge Base Subsystem (KBS)

In general, a knowledge base is defined as a centralized repository of information about a particular domain. In relation to information technology, a knowledge base is a machine-readable resource for the dissemination of information, generally online or with the capacity to be put online. It is used to optimize information collection, organization, and retrieval for an organization.

Knowledge bases may not seem to differ from databases. Both are data structures, in which information is stored and from which information is retrieved. Yet intuitively, knowledge bases and databases are not quite the same. The main difference is in the nature of the information stored. Databases store data while knowledge bases store knowledge. Data are facts that require further computation after being retrieved, in order to generate useful information. On the other hand, knowledge is at a higher level and is an information artifact in its own right, which can be used straight away. One of the definitions of knowledge is actionable information. However, knowledge also includes judgements which are not actionable.

As explained earlier in this chapter, technical know-how is one of the crucial elements that contribute to an ASPs competitive advantage. The ability to provide prompt

CHAPTER 3

corrective measures to clients in the face of technical problems is highly desirable. Often, the correct solution to a problem may be discovered after a lengthy period of trial and error by a technical staff. Therefore, such valuable solutions that have been tested and proven to work in a certain scenario ought to be properly recorded and made available for retrieval in the event of future recurrences of similar scenarios. Having such a knowledge base system will help to decrease the need for a repeat of the effort to rediscover the correct solution, and also speed up the process of applying the correct solutions to recurring problems.

In our proposed infrastructure, the Knowledge Base Subsystem (KBS) is the first component of the AOA and it is the largest and most involved one. KBS is responsible for storing, managing, and retrieving *case* bases. KBS needs to correctly retrieve the matching *case* in response to a condition (detected by other means, not the job of KBS) at customer's PC or in response to a specific descriptive request from human technician or customer. Therefore, to implement the KBS, we need to address 4 issues:

- 1) knowledge acquisition (method to capture *cases*)
- 2) design an appropriate form of representation for stored *cases*
- 3) define the procedures for matching *cases* to help requests
- 4) specify how to consolidate the **KB** by combining any individual *cases* that are near/close to each other, thus forming more comprehensive *cases*

The first issue is actually not the responsibility of the KBS. But because *cases* need to be collected before the KBS has anything to handle, therefore we describe about the method to collect *cases* under this KBS section.

3.3.2.1.1 KBS 1st Issue – *Case* capture

There must be a method to collect *cases*.

A first new problem or condition encountered by customers must first be manually solved by a human expert (very often, a technician), either on-site or remotely. A recorder program can be easily written (shown in **Chapter 5 Prototype Tools**) to

CHAPTER 3

observe and record all the manual keyboard/mouse inputs generated by the technician in order to build *cases*.

For example, a technician may want to create a *case* that automatically installs an application. The technician first invokes the installation program of the application on his/her computer. All the steps taken by the technician, from start to finish of the program, are recorded by our recorder program. A *case* is created. When the same task (to install that application) is encountered by a client in the future, it can be dealt with by the AOA by retrieving the previously created *case*. To handle new problems or conditions (where no corresponding *case* had been created) encountered by customers, a technician can help the customer to carry out the solution steps remotely, through remote desktop display software, for example RealVNC. Since RealVNC is an open source program, the codes of our recorder program can be incorporated into RealVNC so that *cases* can also *be* created under such circumstances (real time). In this case, the small RealVNC server program has to be pre-installed on customer's computer.

3.3.2.1.2 KBS 2nd Issue – *Case* Representation

The 2nd issue regarding KBS is the important part: how to represent the *cases*. As defined earlier, a *case* is: 1) a goal, 2) the possible stages before reaching that goal, and 3) its associated sequence of solution steps to reach that goal. A typical *case* will have an initial stage (signifies the emergence of the problem), several intermediate stages and a final "goal" stage (signifies that the problem has been solved). An example of a goal is "installation of an application XYZ". In that case, the initial stage is the first dialog box of the XYZ installation program.

A stage is an instance of the computer desktop appearance condition. Every stage has its own unique defining characteristics (that lets us know what stage is being display). The defining characteristics are collectively called the symptom S_i of that particular stage. So a stage is actually a symptom and vice-versa. Below is the formal definition of a stage.

Definition II:

A stage S_i is defined as a set $\{Chr_x, 1 \leq x \leq K\}$ where:

- 1) K is the total number of components in a window.
- 2) Chr is a 5-tuple $\langle C1, C2, C3, D, St \rangle$ where $C1, C2, C3, D$, and St respectively are the caption, class name, relative coordinates, dimension, and style of a component in a window.

The meaning of Chr will become clearer in later discussions.

Referring to **Figure 3.3a**), a symptom is shown as a square box S_i . For the example of application installation program, a symptom could be a capture of the characteristics of the presently viewable/active dialog box. The symptom number, i or j or whatever, is just a tag for that stage/symptom. **The number has no particular meaning, so a sorted ordering of the numbers is not explicitly required.**

Transition from one symptom to another is done through a sequence of actions performed by the human technician, such as keyboard/mouse inputs. A sequence of actions is called an action flow. So, there is at least one action flow between two adjacent symptoms. Referring to **Figure 3.3a**), an action flow between two symptoms i & j is shown as a pointed arrow $\text{AF}_{(i-j)}$. For example, an action flow could be to key in some information into several edit boxes and then to click the "OK" button.

Figure 3.3 Elementary diagrammatic representation of a Case

The final "goal" stage has a unique alphanumeric tag or ID that denotes the problem or *case*. In actual storage of a *case* in knowledge base, every *case* also has an accompanying parameterized description that conveys the purpose of the *case*. The parameterized description is covered in **Section 3.3.2.1.3 KB 3rd Issue – Case Matching**.

CHAPTER 3

The goal stage is usually a condition rather than a window/dialog box. For example, the condition is that the installation program has terminated. So sometimes a *case* actually ends with the final action flow arrow. The goal stage is put there to complete the diagram. For example, when the "installation completed" dialog box (last stage before goal stage) is displayed, we know that the condition/goal has been reached. But one final action flow arrow is still needed to click the "Exit" button to close the dialog box.

Referring to **Figure 3.3 a**), the goal stage is shown as a circle S_{IDx} with IDx being the ID of that *case*. (Actually, intermediate symptoms S_i should be $S_{i(IDx)}$ and $AF_{(i-j)}$ should be $AF_{(i-j)(IDx)}$, to differentiate S_i of *case* IDx from S_i of another *case* IDy . But, for convenience's sake, we drop the term " IDx " for this discussion purpose. It is understood that the term should be there.)

So a complete *case* is ONE goal stage plus a series of symptoms and action flows. The simplest *case* is linear without branches from start to end, as in **Figure 3.3b**). A linear *case* is the initially built *case*, after a session of technician interacting with the GUI, and before the freshly built *case* is being combined with other *cases* (described later). Please note that two or more *cases* could have an identical goal stage, even though the series of symptoms and action flows are different. Then these *cases* have the same IDx , but with a subscript, like IDx_I .

On the other hand, some *cases* could have branches. Please refer to the example of a slightly more complex *case* with branches in **Figure 3.4**. Notice the reverse-directioned arrows of some of the action flows.

Figure 3.4 Case Graph: A more complex *case* with branchings

With the complex *case* in **Figure 3.4**, we now formally define the graphical representation of a *case*.

Definition III:

A case $C_G = \langle G, S, AF \rangle$ is represented by a digraph where:

- 1) Goal G is represented by a circle named S_G . It is the goal stage.
- 2) There is one and only one goal stage.
- 3) Each non-goal stage S_i is represented by a rectangle named S_i .
- 4) Each action flow $AF_{(i-j)}$ is represented by a pointed arrow from rectangle S_i to rectangle S_j .
- 5) Two or more arrows pointing out from the same rectangle S_a is called a **branching**, where for $AF_{(a-j)}$, there exist one or more $AF_{(a1-m)}$ where $a = a1$, $j \neq m$.

Note that the diagrammatic representation of a *case* is a non-multi digraph. It's a digraph because of the directed edges (arrows), and non-multi because there are no two parallel edges with the same direction.

The *case* graph looks like a finite state machine [59] graph. But there are some differences. In finite state machine, a situation can stay at any state as long as there is no input or event to trigger the transition to another state. As opposed to finite state machine, there is a goal stage which is the desirable stage to be reached in our *case* graph. If a non-goal stage is reached, transition to the next stage is attempted at once (following the

CHAPTER 3

Navigation Rules, **Section 3.3.2.2.1**) without the need for any event to trigger it, until the goal stage is reached.

As defined, branching occurs when there are > 1 arrows going outwards from the same stage. So in **Figure 3.4**, circle 1 is a branch while circles 2 and 3 are not branches.

Branching occurs when there are choices to be made at certain stages in a case, and the choices are dependent on external information, such as customer preference profiles or prevailing condition of the PC (available hard disk space, etc). *Cases* with branchings are built through the *cases* combination process which will be described in details later. A branching *case* will present a certain amount of difficulties and ambiguities when AOA (or specifically, AMM & CBI) has to determine which next action flow to perform. But that is not the KBS' concern. That is where the Case Base Interpreter comes in handy. Details on how to navigate a complex branching *case* will be covered later when we discuss the CBI and the Navigation Rules.

Now we know that a *case* is stored in the knowledge base (KB) in the form of a series of symptoms and action flows, under an ID tag. So the next step is to describe a stored symptom's actual content and a stored action flow's actual content.

One way to store a stage's symptom is to capture a screenshot of the application window or dialog box during that stage. But this method is cumbersome, ineffective, and slow. Screenshots files are relatively large, even if the light **GIF** format is used. More importantly, a screenshot is a static symptom and thus is highly vulnerable to even the slightest deviations of the application window/dialog box appearance. For example, when a previously recorded case is being carried out now and a screenshot-symptom matching is performed (where the system needs to find a match between a currently detected screenshot-symptom and any stored screenshot-symptom), matching will return negative results if the font color in the current screenshot is different from the font color in the stored screenshot, or if an icon's position is slightly different. This negative result occurs even if two stages are actually the matching stage. Also, matching a screenshot with a large depository of stored screenshot is a slow process.

CHAPTER 3

A more robust and effective method to store a stage's symptom is by capturing the characteristic data of the desktop/window of a stage. For GUI, every window is actually made up of many other smaller and nested windows in a hierarchy. For example, a control button is actually also a small window. Each small window has its own set of parameters: Caption, Class name, Disposition, Width & Height, Window style, etc. The collection of all the nested windows' sets of parameters will uniquely define a window/symptom.

There are readily available "read window" tools that allow the above-said windows' parameters to be read. For example, the Spy++ utility program (**Figure 3.5a**) that is packaged with Microsoft Visual Studio SDK has a feature called "Window Finder" which allows user to point to any window or component (by dragging a crosshair icon over the window) and their parameters will be displayed. Another freeware tool called the SysTree++ (**Figure 3.5b**) is able to show information regarding all windows that are running, although it does not allow user to directly select a desired window on screen.

CHAPTER 3

(a) Spy++

(b) SysTree++

Figure 3.5 Spy++ and SysTree++ tools

CHAPTER 3

However we need to have the "read window" capability incorporated into our infrastructure (Knowledge Base Subsystem, Case Base Interpreter, and Application Maneuver Mechanisms need to "read window") when *cases* are created or when *cases* are invoked and carried out. Therefore we have elected to write the necessary codes.

Our recurring algorithm that traverse (depth first) all the nested child windows of a parent window is given below in **Figure 3.6**. We implemented it for the Windows environment.

```

HWND WindowHandle;

//active window is the current stage
set WindowHandle = handle to the foreground active window;

Traverse(WindowHandle); //Call the Traverse() function

Traverse(HWND WinHandle) //The function for depth first traversing
{
    get the window's caption
    get the window's class name
    get the window's relative coordinates
    get the window's dimension
    get the window's style

    HWND tempHandle;

    //to go down one level of the windows hierarchy
    set tempHandle = handle to the top most child window

    while (tempHandle != 0)
    {
        Traverse(tempHandle);

        //to move horizontally to a sibling window of the same level
        set tempHandle = handle to the sibling window of tempHandle;
    }
    return;
}

```

Figure 3.6 Algorithm for traversing all the child windows of an active window

CHAPTER 3

(a)

(b)

Figure 3.7 Example of a dialog box and its list-symptom captured by our program

CHAPTER 3

To ensure that the list-symptom is not a static symptom, we would prefer to omit information regarding the location of windows so that a *case* is less affected by changes in the layout (for example, positioning of a toolbar) of a window of a stage. By implementing the algorithm of **Figure 3.6** and applying it on "RealPlayer" installation program shown in **Figure 3.7 a)**, an example of captured list-symptom is shown in **Figure 3.7 b)**. The list-symptom shown in **Figure 3.7 b)** can be an option as a stored symptom in the KB.

However, from our implementation experience, we have come across various sizes of list-symptoms. Almost all list-symptoms exceed 1000 characters, while some large list-symptoms with long lists have over 10000 characters, because of many child windows. Since each *case* consists of many list-symptoms (many stages), when the *case* KB grows larger with new *cases* being added, the number of list-symptoms increases considerably. Thus, having smaller list-symptom size is very desirable. We propose a method that can dramatically reduce the amount of storage needed by the symptoms. We call it the Hashed-Window-Symptom method.

This method uses a hash function. A hash function (such as MD5, SHA, RIPEMD-128, RIPEMD-160, etc) can process an input file of any size and generate a unique tag, called a hash code. A hash code is a fixed size string (usually 128bits or 160bits, depending on the hash function), no matter how large the input file is. What's interesting about a hash function is that practically no two or more input files can produce the same hash code (please bear in mind that for a 160bit hash code, $2^{160} = 1.46 \times 10^{48}$ different combinations!). Thus, in practice a hash code is almost guaranteed to uniquely represent its input file. We elected to use RIPEMD-160 as the hashing function in the Hashed-Window-Symptom (HWS) method, and the list-symptoms as input files. Since one character is 8 bits long, so if we store the 160bit hash codes instead of the list-symptoms then the amount of storage saved by using HWS-symptoms is between:

$$(1000 \times 8) \div 160 = \mathbf{50 \text{ times}} \quad \text{and} \quad (10000 \times 8) \div 160 = \mathbf{500 \text{ times}}$$

which is very significant. Thus, HWS-symptom is the chosen storage format of symptoms in the *case* KB. A HWS-symptom is a simple string of $160 \div 8 = 20$ characters.

CHAPTER 3

Another benefit of this short symptom is that future searching and symptom-matching speed is faster.

Having described the stored contents of a symptom, we now describe the stored contents of an action flow. There are 6 fundamental type of actions: i) mouse left-click (MLC), ii) mouse right-click (MRC), iii) mouse double left-click (MDLC), iv) mouse drag (MD), v) mouse point (**MP**), and vi) keyboard input (**KI**). The action flow is stored in the format of XML file.

Each type of action must have an associated child window on which the action was performed. For example, for a MLC, an associated child window could be a button. So, each action is recorded in the action flow together with the parameters of its associated child window. Additionally, MD and KI have a few extra parameters. Each action type's record details are shown in the sample list of action flow in **Figure 3.8**. It is in well-formed XML format so that reading/parsing by Case Base Interpreter is easy. XML format is also very human-readable in the rare occasion when a technician wants to read the action flow. Thus even a raw action flow can be understood and modified by a technician, although in order to prevent human errors it is not recommended to modify an action flow manually this way. More desirable is a tool written to parse the XML action flow and present it to technician by using a action flow GUI editor.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<actions>

  <mouse_left_click>
    <window_caption>B&rowse</window_caption> <!--Comment: & is & -->
    <class_name>Button</class_name>
    <width>42</width>
    <height>16</height>
    <>window_style>2011234304</window_style>
  </mouse_left_click>

  <wait_milliseconds>1500</wait_milliseconds>

  <mouse_right_click>
    <window_caption></window_caption> <!-- Comment: No caption -->
    <class_name>Afx:400000:b:10011:10:0</class_name>
    <width>150</width>
    <height>30</height>
    <>window_style>4194304</window_style>
  </mouse_right_click>

  <wait_milliseconds>1500</wait_milliseconds>
```

CHAPTER 3

```

<mouse_double_left_click>
  <window_caption>Explorer</window_caption>
  <class_name>Afx:400000:b:10011:10:0</class_name>
  <width>120</width>
  <height>28</height>
  <window_style>4194304</window_style>
</mouse_double_left_click>

<keyboard_input>
  <input_string>%Customer_Name%</input_string>  <!--Comment: variable -->
  <window_caption>Name</window_caption>
  <class_name>Edit</class_name>
  <width>182</width>
  <height>23</height>
  <window_style>2011234304</window_style>
</keyboard_input>

<mouse_drag>
  <x_axis_movement>63</x_axis_movement>
  <y_axis_movement>-50</y_axis_movement>
  <window_caption>Display Properties</window_caption>
  <class_name>#32770</class_name>
  <width>303</width>
  <height>100</height>
  <window_style>3012302</window_style>
</mouse_drag>

<start_choice>
  <%Preference_Desktop_Shortcut%>Yes</%Preference_Desktop_Shortcut%>
</start_choice>

<mouse_left_click>
  <window_caption>Add Desktop Shortcut?</window_caption>
  <class_name>Button</class_name>
  <width>42</width>
  <height>16</height>
  <window_style>201232308</window_style>
</mouse_left_click>

<end_choice>
  <%Preference_Desktop_Shortcut%>Yes</%Preference_Desktop_Shortcut%>
</end_choice>

<mouse_point>
  <window_caption>View</window_caption>
  <class_name>Static</class_name>
  <width>38</width>
  <height>20</height>
  <window_style>2011233203</window_style>
</mouse_point>
</actions>

```

Figure 3.8 A sample action flow in well-formed XML format

CHAPTER 3

In **Figure 3.8**, the XML tags related to the 6 type of actions are self-explanatory. Those are our standard action flow tags. But, currently, we also specify 3 additional independent elements/syntaxes:

- 1) The pair of `<wait_milliseconds>xxx</wait_milliseconds>` tags are for specifying a time delay xxx milliseconds between actions. This tag can be added into the action flow by a human technician, through a special purpose action flow editor (implemented in **Chapter 5**).
- 2) `%...blablabla...%` syntax for Keyboard Input indicates a string that is dependent on customer profile/preference. Upon seeing this syntax, the Case Base Interpreter knows that it is a variable (for example, customer name, organization, etc) and will attempt to retrieve the actual value of `%...blablabla...%` from the customer profile
- 3) `<start_choice>` and `<end_choice>` elements. `<start_choice>` element indicates that the actions listed immediately after the `<start_choice>` element and before the following `<end_choice>` element are dependent on external parameters, to decide whether or not the enclosed actions are to be carried out. A pair of corresponding `<start_choice>` and `<end_choice>` elements both contains an identical element `<%...blablabla...%>` (in this example, `<%Preference_Desktop_Shortcut%>`) with content xyz (in this example, "Yes"). If the current value of parameter `%...blablabla...%` obtained from customer profile database (or interpreted by CBI) is equal to "xyz", then the actions enclosed between `<start_choice>` and `<end_choice>` elements are performed. Otherwise, those actions are skipped. So "xyz" is like a key to unlock the optional actions.

The XML tag and syntax definitions can be further extended and enriched if desired. But currently we only implement these 3 independent syntaxes plus the standard action flow tags.

We present a real example in **Figure 3.9** to illustrate what an action flow means. The dialog box displayed is one of the dialog boxes during execution of installation

CHAPTER 3

program for application Centurion. **Figure 3.9a** shows the original look of the dialog box before any action is performed on it. The action flow for the dialog box is in **Figure 3.9b**. There are five actions in this action flow:

- 1) First, mouse left click on the edit box which has a caption of "" ,class name of Edit, width of 173 pixels, height of 20 pixels, window style of 50010080. Refer to **Figure 3.9c**. This is to let the edit box has the keyboard input focus. Notice the emergence of cursor in the edit box (mouse pointer is not captured in the screenshot).
- 2) Second, wait/pause for 1000 milliseconds.
- 3) Third, keyboard input. The string to be input is the value of variable %Customer_Name% for the current customer. This value (in this example, Yao Min) is fetched from customer's profile and then keyed in. Refer to **Figure 3.9d**.
- 4) Fourth, wait/pause for 1000 milliseconds.
- 5) Fifth, mouse left click on the button which has a caption of Next >, class name of Button, width of 87 pixels, height of 36 pixels, window style of 50010000. Refer to **Figure 3.9e**. Notice the button Next > being depressed (again, mouse pointer is not capture in the screenshot). This is the final action for this dialog box (stage).

(a) Dialog box

CHAPTER 3

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<actions>

  <mouse_left_click>
    <window_caption>""</window_caption>
    <class_name>Edit</class_name>
    <width>173</width>
    <height>20</height>
    <window_style>50010080</window_style>
  </mouse_left_click>

  <wait_milliseconds>1000</wait_milliseconds>

  <keyboard_input>
    <input_string>%Customer_Name%</input_string>
    <window_caption>""</window_caption>
    <class_name>Edit</class_name>
    <width>173</width>
    <height>20</height>
    <window_style>50010080</window_style>
  </keyboard_input>

  <wait_milliseconds>1000</wait_milliseconds>

  <mouse_left_click>
    <window_caption>&Next ></window_caption>
    <class_name>Button</class_name>
    <width>87</width>
    <height>36</height>
    <window_style>50010000</window_style>
  </mouse_left_click>

</actions>

```

Diagram illustrating the action flow for the XML code above:

- 1st Action: <mouse_left_click> (first block)
- 2nd Action: <wait_milliseconds>1000</wait_milliseconds>
- 3rd Action: <keyboard_input> (second block)
- 4th Action: <wait_milliseconds>1000</wait_milliseconds>
- 5th Action: <mouse_left_click> (third block)

Additional annotations in the diagram:

- A bracket groups the first three actions (1st, 2nd, and 3rd).
- A bracket groups the last two actions (4th and 5th).
- A comment <!--Comment: variable --> is placed next to the <input_string> tag in the 3rd action.

(b) Action flow

(c) First action

CHAPTER 3

Figure 3.9 Example to illustrate an action flow

For many other usages, a standalone well-formed XML file is good enough thus a schema is not necessary. But for our purpose, well-formed XML is not adequate, because we must be sure of the integrity of all action flow XML files for them to be useful for the CBI. Also, we need to occasionally define some custom entities and data formats to be used in our action flow files because we can't predict what kind of data will be needed by other applications. So, we also need to have a schema for the XML files. For this purpose, we chose to use XML Schema instead of Document Type Definitions (DTD), mainly because of the datatyping allowed by XML Schema. The XML Schema for our action flow XML file is shown in **Appendix A: Figure A.1**. In our XML Schema, we explicitly

CHAPTER 3

define the set of elements, attributes, entities and data formats used in the action flow XML files.

3.3.2.1.3 KBS 3rd Issue – *Case Matching*

We have already covered the first 2 issues of **KBS**, i.e.: how to collect *cases* and what is the form of representation for a stored *case*. Now we describe about the **3rd** issue of KBS: the procedures for matching *cases* to help requests. This is at a higher level than matching of symptoms. Symptom matchings are performed after the right *case* has somehow been chosen to address a problem. Now we are interested in how to choose the right *case* for a specific problem.

Most stored *cases* will be retrieved and used eventually. The Application Maneuver Mechanisms (AMM, **Section 3.3.2.3**), with the help of the Case Base Interpreter (CBI, **Section 3.3.2.2**), will "read" the rightly retrieved *case* and carry it out. To match the right *case* (if available) to a specific problem, there can be several ways:

- (A) Automatic matching of a human (either technician or customer) specified text query to stored *cases'* descriptions
- (B) Human browsing through the *cases'* descriptions.
- (C) Through symptom matching
- (D) The right case is explicitly known.

Method (A) is of course the most convenient and intelligent method. A lot of research has been done in this field by the information-retrieval community. Most versions of solutions involve text-search based systems that accept a text query as input, and output a list of potentially relevant documents. In other words, search engines. For our purpose, the list of potentially relevant documents would be a list of potentially relevant *cases'* descriptions. However, text-based search engines, despite the amount of research work done, currently still do not perform at a desirable enough precision level. Further research in this field alone involves heavy work, and we do not cover this in our current work.

CHAPTER 3

For method (B), we implement it in **Chapter 5** by intuitively classifying all *cases* according to:

- i) application type,
- ii) application developer's name (from "Help > About" dialog box),
- iii) application name (from "Help > About" dialog box),
- iv) application version (from "Help > About" dialog box), and
- v) other widely used terms depending on the application

Categories i) – iv) will precisely point to the application in question. The terms used in category v) are extracted directly from the table of contents of the official user manual of the application, which is arguably the most complete official reference about almost all (if not all) the features in an application. Thus we believe that this set of categories i) – v) is a good description mechanism for the purpose of browsing through *cases*. Collectively, the categories form a parameterized description for a *case*. Each *case* has an associated parameterized description, in addition to an optional normal language description written by the technician. For example, for a *case* that helps set the font size in Microsoft Word 2003, the parameterized description of this *case* would be: < ⁱ⁾Word Processor.ⁱⁱ⁾Microsoft Corporation.ⁱⁱⁱ⁾Microsoft Office Word.^{iv)}2003.^{v)}Appearance-Font-Size-Set >. When we implement *case* browsing, the parameterized descriptions are used.

Sometimes, the right *case* can even be determined by method (C), symptom matching. For example, when a customer encounters an error dialog box on his/her desktop and doesn't know what to do, he/she can invoke the help of AOA. AOA tries to determine the right *case* (if available) to handle this situation by performing symptom matching (HWS-symptom) of the current dialog box/symptom with the symptoms in the *cases* in knowledge base. Because we are using the short 160bit symptom ID, searching and matching is fast. The *case* which has a matching symptom (among its series of many symptoms) will be fetched and subsequently carried out.

Currently, our infrastructure implementation has a shortcoming, in the sense that when a symptom of a window is to be captured, all components in the window have their

CHAPTER 3

characteristics recorded and included as part of the symptom. The disadvantage of rashly including all components into a symptom is that: for a window that has components with unfixed content, the symptom captured for this window will not be able to match future symptoms captured for the same window. For example, if an installation program happens to put the present date as the caption of a static text in a dialog box (Figure 3.10), then the symptom of this dialog box is only valid for that date. Another example is for an application such as Microsoft Excel which has userdefined fields in the main window. If a *case* (such as a *case* which helps user to change font color) is to be created for such a program, then the symptoms in the *case* should not include whatever is in the main window because those are irrelevant context for distinguishing the window. Therefore, a better approach in creating a *case* is to incorporate a tool which allows technician to select those relevant components for a symptom and omit irrelevant context.

Figure 3.10 Example of a component with unfixed parameter

Under certain situations, the right *case* is explicitly known (method (D)). This usually happens when the customer is conducting a session with the ASP and reaches a certain step where the AOA knows that there is *case* available that can help the customer. For example, after a customer has agreed to download an application to install locally, a *case* that handles the application installation steps can be straightaway offered to the customer.

3.3.2.1.4 KBS 4th Issue – Combining Cases

Now we describe the fourth and final issue regarding our KBS, which is how to consolidate the KB of *cases* by combining any individual *cases* that are near or close to each other, thus forming more comprehensive *cases*.

Firstly, we define what is meant by "near". Two *cases* are near to each other if they have at least one cross-duplicate stage/symptom, as shown in **Figure 3.11 a)**. Cross-duplicates are drawn in the same color (except for black color). In discussions, the symbol " $A = B$ " is used to denote that A and B are cross-duplicate.

The result of combination is a compound *case*. The main difference between a compound *case* and a normal *case* is that a compound *case* may have ≥ 1 goal stage (and usually does). Because of this, a compound *case* is not a real *case*, since a real *case* is associated with only one single goal/problem. A compound *case* acts as a collection of near *cases*. When a real *case* is needed, it can be extracted from within its compound *case*. We will first describe the merits of combination, followed by a description and demonstration of the sequential rules for combination, and then a description of the rules for extraction of individual *case* from a compound *case*. We will also show that *cases* learn new knowledge after going through combination and extraction. Finally, we discuss about how compound *cases* are stored in the KBS.

The merits of combining near *cases* are:

- a) Saves space. By performing combination, we can reduce the overall number of stored symptoms, and even the number of stored action flows in the KBS. When 2 near *cases* are combined, 1 instance of 2 cross-duplicate symptoms and/or action flows is discarded. In fact, >2 *cases* can be combined together one after another.
- b) A large compound *case* provides to the human technician a better unified view of the of problem set, instead of having to view too many individual *cases* for one application (we implement a *case* editor in **Chapter 5** which can paint a

CHAPTER 3

- diagrammatic view of cases). Thus, a technician can more easily pinpoint any new case(s) to be built (even before a problem is encountered for the first time).
- c) Most importantly, combination produces more comprehensive real cases, with new branches being added to the original cases. Thus after going through the processes of combination and then extraction, an original case usually is not the same as before combination. It will have more knowledge and is better equipped to handle potentially encountered symptoms. Thus, combination is actually a learning process.

The above mentioned benefits will be more apparent when we describe the combination process below. The algorithm to for the combination process is given below.

Algorithm I: Case combination

```

1  for each pair of one linear case and one linear/non-linear case
2      A = linear case
3      B = linear/non-linear case
4      CD = set of all cross-duplicate sequences of  $\langle S \rightarrow AF \rightarrow S \rangle$  between A
           and B, ordered starting from goal stage
5      while CD is not empty set
6          do merge the first  $\langle S \rightarrow AF \rightarrow S \rangle$  member of set CD
7          erase the first  $\langle S \rightarrow AF \rightarrow S \rangle$  member of set CD
8      SIN = A cross-duplicate  $\langle S \rangle$  which has no adjacent S merged, and which
           is nearest to goal
9      while SIN is not empty
10         do merge SIN
11         update content of SIN

```


CHAPTER 3

Figure 3.11 Steps for combining 2 cases

To easier understand the steps of the algorithm, the steps are actually some sequential rules for combining *cases*:

- Cmb1) Combine only 2 near *cases* at one time. At least one of the 2 near *cases* has to be a linear *case*. We usually refer to this as combining the linear *case* TO a non-linear or linear *case*.
- Cmb2) Starting from goal stage, merge (and discard one) all cross-duplicate sequences of $\langle S \rightarrow AF \rightarrow S \rangle$. See **Figure 3.11 b) & c)**. The same colored (except for black color) *S* or *AF* indicates crossduplicates. We omitted the "*S*" symbol for simplicity in this diagram.
- Cmb3) Next, merge (and discard one) all cross-duplicate $\langle S \rangle$ that has no adjacent *S* already merged. This restriction is to prevent merging of cross-duplicate adjacent *S*'s with non-cross-duplicate *AF* connecting them. See **Figure 3.11 d)**.
- Cmb4) Repeat rule Cmb1) for combining next linear case.

After combination, if a real *case* needs to be retrieved from the compound *case*, extraction is possible. The algorithm to for the extraction process is given below.

Algorithm II: Case extraction

- 1 **do** extract goal stage
- 2 CS = set of latest extracted stage(s) //now, it contains the goal stage
- 3 ADJArrows = set of all adjacent action flow arrows pointing towards each member stage in CS

```

4  while ADJArrows is not empty set
5      do extract every member of ADJArrows
6          ADJStages = set of all stages from which each member arrows in
                    ADJArrows originated
7          extract every member of ADJStages
8          CS = ADJStages
9          update content of ADJArrows

```

To easier understand the steps of the algorithm, the steps are actually some rules for extracting a real *case* from a compound *case*: (shown in **Figure 3.12**):

- Ext1) Start from the goal stage. The goal stage is extracted.
- Ext2) Locate every adjacent stage/symptom which has an arrow pointed from that adjacent symptom towards the current symptom. Those arrows are extracted and will be part of the extracted case.
- Ext3) Move the focus to each of the symptoms located in Ext2. Those symptoms are extracted and will be part of the extracted *case*. Repeat Ext2 for each of those symptoms, until no more adjacent symptom has an arrow pointed towards the current symptoms.

Extraction of a real *case* is just to recover an instance of a real *case* from the compound *case*. Doing so does not shrink or alter a compound *case*. As mentioned earlier, extraction should produce a more-knowledgeable-than-original *case*. We show the extraction steps in **Figure 3.12**, by using the compound *case* from **Figure 3.11 e**) as an example and try to extract *case* IDa. Each symptom or action flow that conforms to either one of rules Ext1, Ext 2, or Ext 3 is painted red to signify that it will be part of the extracted *case*. Compare the newly extracted real *case* IDa in **Figure 3.12 j**) with the original *case* IDa and the table in **Figure 3.11 b**). We can see that the combine-then-extracted new *case* IDa has managed to learn new knowledge and is more comprehensive for solving future problems.

CHAPTER 3

CHAPTER 3

Figure 3.12 Steps for extracting a real case from a compound case

CHAPTER 3

Notice that after a combine-then-extract process, an originally linear case will most likely acquire new branches. This means that some symptoms will have a choice of more than 1 action flows. Therefore, later on a technician may use an action flow GUI editor to include the condition on how to choose which action flow to perform. Essentially, this is done by enclosing each of the action flows with a pair of `<start_choice>` and `<end_choice>` elements. Until this condition is added by a technician, the Case Base Interpreter (CBI) will choose to perform any one of the action flows at a branch.

For example, in an installation program there is a choice between doing either a "Typical", "Minimum", or "Maximum" installation, such as in **Figure 3.13**.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<actions>

  <start_choice>
    <%Preference_Installation_Size%>Typical</%Preference_Installation_Size%>
  </start_choice>
  ...
  ...
  ...
  <end_choice>
    <%Preference_Installation_Size%>Typical</%Preference_Installation_Size%>
  </end_choice>
</actions>
```

Figure 3.13 A stage with choice

CHAPTER 3

For this stage, if three action flows are available, then every one of them are enclosed with a pair of <start_choice> and <end_choice> elements, with content of <%Preference_Installation_Size%> being "Typical", "Minimum" and "Maximum" respectively. The actual value of %Preference_Installation_Size% is fetched from customer profile. If the value is not available from customer profile, or if the choice is a newly created one, then interaction between ASP and customer is necessary in order to acquire the customer-desired value. A suggested method is to have a client agent to prompt customer to enter or select the value and then store it in a new entry in customer profile.

CHAPTER 3

Duplicates: $S_1 == S_6$; $S_5 == S_9$; $AF_{(5-IDa)} == AF_{(9-IDa)}$

Compound *case*. Extracted real *case* will be the same as the compound *case*.

Duplicates: $S_{1(IDa)} == S_{1(IDb)}$; $S_{2(IDa)} == S_{2(IDb)}$; $S_{3(IDa)} == S_{3(IDb)}$;
 $S_{4(IDa)} == S_{4(IDb)}$; $AF_{(1-2)(IDa)} == AF_{(1-2)(IDb)}$;
 $AF_{(2-3)(IDa)} == AF_{(2-3)(IDb)}$; $AF_{(3-4)(IDa)} == AF_{(3-4)(IDb)}$;

Compound *case*. Each extracted real *case* will be the same as its individual original *case*.

Figure 3.14 Two extremes of combining *cases*

There are 2 types of extremes in *case* combination. One is total learning, where the extracted *case* learns every symptom available from the compound *case*. The other extreme is total save space, where the extracted *case* learns nothing, but manages to save the most space by combining.

CHAPTER 3

The first extreme is shown in **Figure 3.14 a)**. The example used is the now familiar installation program. Both the two linear cases' goals are the same, which is to install a specific program. But one linear case was built when the technician chose the path of "minimum" installation option, and the other linear case was built when the technician chose the path of "typical" installation option. Looking at the resulting compound case, we can easily discover that if we were to perform extraction, the extracted case will be exactly the same as the compound case, meaning that the extracted case has learnt everything from this compound case. In fact, this is how branching cases are generated. 2 cases with the same goal but differing intermediate symptoms are combined into 1 larger case with that goal and options for branching within the case. The resulting larger case is better than any of the two original cases.

The second extreme is shown in **Figure 3.14 b)**. Two different cases with different goals have a number of same symptoms at the initial stages. Looking at the resulting compound case, we can easily discover that if we were to perform extraction, the extracted case (either one) will be exactly the same as its original version before combining. Thus the extracted case learns nothing from the compound case. The only benefit is that combination of these two cases saves a lot of space.

Now let's discuss the actual consequences of combination, on how the case base database (**KBS**) stores individual cases and compound cases. Each case is a record/row in the **KB**. For the following discussion, please bear in mind that "row" = "record" = "case". Each row contains entries/fields for all the symptoms and action flows for that case, with each symptom or action flow occupying a field. Other entries in a row are the parameterized description and normal language description of that case. If combination is not deployed, many rows that belong to a certain application (rows that addresses different problems for the same application) will have cross-duplicate symptoms and/or action flows.

If consolidation by combination is carried out, then each record is an extracted case. The contents of crossduplicate symptoms and cross-duplicate action flows are discarded and each is replaced by a reference to a master copy of the symptoms and

CHAPTER 3

action flows. The reference occupies the field that was once occupied by the now discarded crossduplicate symptom or action flow. A symptom/action flow master copy is the symptom/action flow of the first original *case* that brought this symptom/action flow into a compound *case*. Any future cross-duplicate copies (when new *cases* are recorded) of the symptom/action flow will have their contents discarded and replaced by a reference to the first master copy. For example, referring to Figure 3.11 d), after merging, the content of symptom S_{2a} are discarded and replaced with a reference to symptom S_{2b} which is the master copy. The row entry: "symptom S_{2a} " still physically exists in the **KB**, although S_{2a} has no actual content, just a reference.

So, each original *case* in each row is updated and upgraded after consolidation through combination. Update refers to clearing the contents of cross-duplicate symptom/action flow and replacing the contents with references to the master copies. Upgrade refers to new symptom/action flow elements being added into the row because of new knowledge learned from the compound *case*. Each row now represents an extracted *case* instead of an original case. The total number of rows are reduced because *cases* with the same goals are permanently merged (as in **Figure 3.14 a)**), where one row is updated and upgraded while another row is totally deleted. The average number of elements in each row is increased because of upgrading. The total size of the **KB** (or the average size of each row) is reduced because of deletion of crossduplicate contents and updating.

An algorithm can summarize what happens above:

Algorithm III: Modifying Database After a New Case is Added

```

1  if a new case is added
2      then do  $C_N$  = new case
3          SCOM = set of all compound cases which are near to  $C_N$ 
4          if  $n(SCOM) \neq 0$ 
5              then for each member  $SCOM_i$  of SCOM where  $1 \leq i \leq$ 
n(SCOM)
6                  do combine  $C_N$  with  $SCOM_i$ 
7                       $SCOM_i' = SCOM_i$  updated with  $C_N$ 
8                      extract all case from  $SCOM_i'$ 
9                      for each extracted case
10                         do upgrade by adding fields for any new
symptom/action flow
11                         update by replacing contents of any
crossduplicate symptom/action flow with reference to master copy

```

Extracted *cases* are explicitly and physically recorded in rows. But the compound *cases* are not physically stored. They are implicitly and logically stored, with the master copy elements and the references working together to maintain the structure information of compound *cases*. The master copy elements also keep records of the tags of all elements which refer to them. This information is needed to reconstruct the compound cases when welcoming newly built cases.

With that we wrap up the description on consolidating KBS through combination of *cases*. This is the final KBS issue to be discussed, thus we have described all four issues of our KBS. In the next section we will explain about the Case Base Interpreter.

CHAPTER 3

3.3.2.2 Case Base Interpreter (CBI)

Figure 3.15 The Case Base Interpreter

The Case Base Interpreter (CBI) is the second of the 3 components of Application-Oriented Automation (AOA). The inputs to CBI are: 1) a *case* and 2) symptoms (one at a time).

	Functions of CBI	
	Navigate case	Interpret case
Input	1) <i>case</i> 2) symptoms	1) "Varying action flows" which is the result from navigating the <i>case</i>
Knowledge or Resources used	1) Five Navigation Rules	1) Action Flow XML Schema 2) An external customer profile database
Tools used	-	1) XML parser
Actions performed	1) Navigates through the <i>case</i> 2) Selects the matching action flow for each symptom received	1) Interprets "varying" action flows to generate "absolute" action flows
Output	1) "Varying" action flows	1) "Absolute" action flows

Table 3.1 Functions of CBI: Navigates and Interprets Cases

As summarized in **Table 3.1**, the functions of CBI are to navigate and interpret a *case*:

- i) Navigation: Based on every symptom received (input to CBI) CBI will navigate through the *case* and selects the matching action flow.
- ii) Interpretation: Some action flows in a *case* are "varying". Varying action flows are action flows which contain variables (for example, %Customer_Name%) with values not fixed, and contain optional actions that depend on customers' preferences (for example, %Preference_Desktop_Shotcut%). So the CBI has to interpret an action flow by collaborating with some sort of customer profile database. The result is a client-specific "absolute" action flow which is customer-specific and omits non-preferred actions. The absolute action flow is sent as output to the Application Maneuver Mechanisms (AMM) to carry out.

Note that the details of collaboration and communication with a customer profile database are implementation issues and are not covered this section.

3.3.2.2.1 *Case* Navigation

The 1st function of CBI: *Case* navigation is to discover and select a path, starting from the first symptom of a *case* to reach the final target, i.e. the goal symptom. It is not a straightforward job.

To describe about this function, we start by looking at the structure of a case. Recall that the last part of the previous **Section 3.3.2.1** **KBS 4th Issue** described about *case* combinations. Because of the combination and extraction processes, we have non-linear *cases*, i.e. *cases* with bmnchings. Branching occurs when a symptom has more than one arrows pointing out. This happens under either one of the following conditions: 1) when a different action flow leads to a different resulting symptom, or 2) when the same action flow leads to a different resulting symptom.

Figure 3.16 Case with branching

Condition 1) is called "Natural Branching" (shown in Figure 3.16 a)) and is less complicated. This happens when a different action flow is explicitly chosen because of customer profile/preferences or other conditions. We call it natural branching. For example, for an installation program at stage S_2 , installation type is selected (maximum, typical) based on customer preferences, thus different preference results in different branches (either S_6 or S_3). The subsequent stages proceed normally without regard to the other branch.

Condition 2) is called "Forced Branching" (shown in Figure 3.16 b)) and is more complicated. This happens when the same action flow ($AF_{(4-5)} == AF_{(4-9)}$) is carried out but an unexpected problem (already recorded by technician) is encountered, such that fall back to the previous stage is required. We call it forced branching. For example, when installation type maximum was chosen at stage S_2 a dialog box might pop up at a later stage (S_9) saying that disk space is not enough. Stage S_5 becomes an "unreachable" branch (for this installation session only). The flow of events must fall back to S_4 , S_3 , S_2 ,

CHAPTER 3

and then eventually to the other branch S_6 , and not to repeat the unreachable branch. This involves three difficulties that must be addressed:

Diff (A): When control falls back from S_9 to S_4 , how does the CBI prevent itself from choosing the "unreachable" branch again and falls into an endless loop?

Diff (B): When control falls back to S_3 , how does the CBI recognize that it must go backwards to S_2 but not forward again to S_4 ? (Please bear in mind that the symptom at S_3 is the same, either from S_2 to S_4 or from S_4 to S_2)

Diff (C): If control manages to fall back all the way to S_2 , how does the CBI prevent itself from missing the fresh branch of S_6 and undesirably goes back to S_1 ?

In short, we want to find a way such that CBI can navigate with purpose within a *case*'s potential mesh of natural and forced branches, while without having to keep an elaborate amount of navigational history data for this. This can be done efficiently by a set of five Navigation Rules for navigating the *case*:

- 1) 1st choice of action flow: Choose a fresh adjacent action flow (a previously not performed action flow). If tie (more than one fresh adjacent action flow) or doesn't exist, then,
- 2) 2nd choice of action flow: Choose a fresh adjacent stage (have not previously gone into). If tie (more than one fresh adjacent stage) or doesn't exist, then,
- 3) 3rd choice of action flow: Obey customer preferences. If doesn't exist, then,
- 4) 4th choice of action flow: Choose ANY fresh adjacent stage. If doesn't exist, then,
- 5) 5th choice of action flow: Choose ANY fresh adjacent action flow. If doesn't exist, report error and exit.

Reporting error in Rule 5 is normally never used. This is just for safety precaution to prevent an infinite loop.

CHAPTER 3

Navigation Rule (1) addresses Diff (A) by avoiding previously performed (not fresh) action flow. Navigation Rule (2) addresses Diff (B) & (C) by avoiding previously encountered (not fresh) adjacent stage.

By using the Navigation Rules, CBI only needs to keep a temporary pool of the tag names of all the action flows. Attaching a superscript to each of these tags will enable the CBI to determine whether an action flow or a stage is fresh or not. So, the combined effect of the tag names pool plus the superscripts plus the currently received symptom is to indirectly tell CBI where its current location is within a case. After the completion of the case, the temporary pool is discarded.

For example, refer to *case* S_{ID1} in Figure 3.16. With this *case* as the input to CBI, CBI will have the following temporary pool of data initially.

Stages	Tag name	S_1	S_2	S_3	S_4	...	...
	Content						
Action flows	Tag name	$AF_{(1-2)}$	$AF_{(2-3)}$	$AF_{(2-6)}$	$AF_{(3-4)}$	...	...
	Content						

Then after stage S_1 is encountered and action flow $AF_{(1-2)}$ is carried out, their tag names will become S'_1 and $AF'_{(1-2)}$ respectively, with an added " ' " superscript. CBI knows that they are now "not fresh". In this way if latter on, when symptom S_4 is received and S'_1 , S'_2 , S'_3 , $AF'_{(1-2)}$, $AF'_{(2-3)}$, $AF'_{(3-4)}$ are not fresh, then CBI will try to go towards S_5 or S_9 instead of going back to S_3 . Because of these information plus the Navigation Rules, CBI indirectly knows that it reached S_4 from S_3 , not from S_9 . Otherwise without the superscripts to indicate which are the "not fresh" stages and action flows, when symptom S_4 is received CBI is equally likely to choose to go to stage S_5 , S_9 or back to S_3 , which we don't want it to do.

CHAPTER 3

To show the effectiveness and robustness of the Navigation Rules, consider the complete and more complex case in Figure 3.17. Note that S_8 and S_{11} are natural branches, thus $AF_{(2-8)} \neq AF_{(2-3)}$ and $AF_{(4-11)} \neq AF_{(4-5)}$, while S_{14} and S_{16} are forced branches, thus $AF_{(11-14)} == AF_{(11-12)}$ and $AF_{(5-16)} == AF_{(5-6)}$. Also, of course all reverse action flows $AF_{(j-i)} \neq$ forward action flows $AF_{(i-j)}$.

Based on the received symptoms (one by one), customer profile or other conditions, if everything goes smoothly, the CBI would eventually follow the path desired by customer, for example:

$$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_{11} \rightarrow S_{12} \rightarrow S_{13} \rightarrow S_7 \rightarrow S_{ID1}$$

Sometimes there might be problems in some intermediate stages. For example, if a problem is to arise at S_{11} such that S_{12} is unreachable (such as, not enough disk space for maximum installation), then branching is forced to S_{14} (such as, a dialog box that informs about the lack of disk space). Let's say another problem also arise at S_5 and forces branching to S_{16} . By following the set of five Navigation Rules, CBI will eventually follow an alternative path to reach the target goal stage S_{ID1} . The eventual alternative path is:

$$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_{11} \rightarrow S_{14} \rightarrow S_{11} \rightarrow S_4 \rightarrow S_5 \rightarrow S_{16} \rightarrow S_5 \rightarrow S_4 \rightarrow S_3 \rightarrow S_2 \\ \rightarrow S_8 \rightarrow S_9 \rightarrow S_{10} \rightarrow S_6 \rightarrow S_7 \rightarrow S_{ID1}$$

CHAPTER 3

Navigation Rules

- Rule 1: Choose a fresh adjacent action flow. If tie or not exist, then
 Rule 2: Choose a fresh adjacent stage. If tie or not exist, then
 Rule 3: Obey customer preferences. If not exist, then
 Rule 4: Choose any fresh adjacent stage. If not exist, then
 Rule 5: Choose any fresh adjacent action flow. If not exist, then
 report error and exit.

Sequence of action flows:

A(1-2)	Rule 1	
A(2-3)	Rule 1 tie. Rule 2 tie. Rule 3 Ok	(Natural branching)
A(3-4)	Rule 1 tie. Rule 2 Ok	
A(4-11)	Rule 1 tie. Rule 2 tie. Rule 3 Ok	(Natural branching)
A(11-12)	Rule 1 tie. Rule 2 Ok	
A(11-14) = A(11-12)	(Forced branching)	
A(14-11)	Rule 1	
A(11-4)	Rule 1	
A(4-5)	Rule 1 tie. Rule 2 Ok	
A(5-6)	Rule 1 tie. Rule 2 Ok	
A(5-16) = A(5-6)	(Forced branching)	
A(16-5)	Rule 1	
A(5-4)	Rule 1	
A(4-3)	Rule 1	
A(3-2)	Rule 1	
A(2-8)	Rule 1	
A(8-9)	Rule 1	
A(9-10)	Rule 1	
A(10-6)	Rule 1	
A(6-7)	Rule 1	
A(7-ID1)	Rule 1 (Finished)	

Figure 3.17 A complex case

CHAPTER 3

The algorithm to navigate a *case* is shown in below:

Algorithm IV: Navigating a Case ID_x

```

1  do  $k = (\text{number of symptoms in case ID}_x) - 1$ 
2     $\text{SetS}_{\text{ID}_x} = \text{set of all symptoms } S_i \text{ in case ID}_x, \text{ where } 1 \leq i \leq k \text{ or } i = \text{ID}_x.$ 
      All are fresh.
3     $\text{SetAF}_{\text{ID}_x} = \text{set of all action flows } \text{AF}_{(i,j)} \text{ in case ID}_x, \text{ where } 1 \leq i \leq k, j =$ 
      subset(1,2,...,k, or IDx). All are fresh.
4     $S_{\text{CS}} = \text{currently received symptom}$ 
5    set  $S_{\text{CS}}$  as not fresh //by adding a superscript to tag name of  $S_{\text{CS}}$ 
6    while  $S_{\text{CS}}$  is not the same as  $S_{\text{ID}_x}$  //SIDx is goal stage
7      if exist one and only one fresh action flow  $\text{AF}_{\text{adj}}$  adjacent to  $S_{\text{CS}}$ 
8        then do carry out  $\text{AF}_{\text{adj}}$ 
9          set  $\text{AF}_{\text{adj}}$  as “not fresh”
10     else if exist one and only one fresh stage  $S_{\text{adj}}$  adjacent to  $S_{\text{CS}}$ 
11       then do carry out  $\text{AF}_{(\text{CS-adj})}$ 
12         set  $\text{AF}_{(\text{CS-adj})}$  as “not fresh”
13     else if exist customer specified preference parameter
14       then do carry out customer preferred action flow  $\text{AF}_{\text{pre}}$ 
15         set  $\text{AF}_{\text{pre}}$  as “not fresh”
16     else if exist fresh stages  $\{S_{\text{adj}1}, S_{\text{adj}2}, \dots\}$  adjacent to  $S_{\text{CS}}$ 
17       then do carry out any one of  $\text{AF}_{(\text{CS-adj}i)}$ 
18         set  $\text{AF}_{(\text{CS-adj}i)}$  as “not fresh”
19     else if exist fresh action flows  $\{\text{AF}_{\text{adj}1}, \text{AF}_{\text{adj}2}, \dots\}$  adjacent to  $S_{\text{CS}}$ 
20       then do carry out any one of  $\text{AF}_{\text{adj}i}$ 
21         set  $\text{AF}_{\text{adj}i}$  as “not fresh”
22     else do report error and exit
23
24     do retrieve  $S_{\text{CS}} = \text{new symptom}$ 
25     set  $S_{\text{CS}}$  as not “fresh”

```

CHAPTER 3

Whenever a new symptom, not already recorded in the case, is discovered, the human technician is alerted to take over the operation. When the technician is handling the new symptom, the new symptom is recorded and included into the case to make it more knowledgeable. This process is similar to what happens when a new case is created. When a new case is created, symptoms and action flows are added one by one to the new case. When new symptoms and action flows are added to an old case, it is just like continuation of the process to create a new case.

3.3.2.2.2 Case Interpretation

In the Navigation function, action flows retrieved from a case are "varying", meaning they contain variables (for example, %Customer_Name%, **Figure 3.18**) with values not fixed, or contain optional actions that depend on customer preferences (for example, %Preference_Desktop_Shortcut%, **Figure 3.18**), or contain both.

```
<keyboard_input>
  <input_string>%Customer_Name%</input_string>  <!--Comment: variable -->
  . . . . .
  . . . . .
  . . . . .
</keyboard_input>

<start_choice>
  <%Preference_Desktop_Shortcut%>Yes</%Preference_Desktop_Shortcut%>
</start_choice>

. . . . .
. . . . .
. . . . .

<end_choice>
  <%Preference_Desktop_Shortcut%>Yes</%Preference_Desktop_Shortcut%>
</end_choice>
```

Figure 3.18 A fragment of an action flow XML file (taken from **Figure 3.8**)

As previously shown in **Table 3.1**, the 2nd function of CBI is to interpret "varying" action flows and generate customer-specific "absolute" action flows, by collaborating with a customer-profile database maintained by ASP.

CHAPTER 3

To perform the interpretation function, CBI uses two resources and an internal tool. The external resources are: 1) Action Flow XML Schema and 2) external customer profile database. The internal tool is an XML parser.

Action Flow XML Schema pre-defines all the variables in an action flow XML file. Currently, we include some simple customer profile data as the variables in varying action flows. The ASP-maintained customer profile database stores the client-specific preferences and values to be substituted for the action flow variables. The XML parser is used to parse an action flow XML file.

The customer ID, an action flow XML file, and the XML Schema are passed to the parser (customer ID is known when the customer logs in before using the service). Through the parser, all variables (enclosed within %...%) in an action flow XML file can be detected and recognized. Each variable's name (everything between two % symbols) exactly matches the name of a field/column in the customer profile record/row. Thus the CBI can request the customer profile database to retrieve the entry at that particular column of said customer ID. CBI replaces all variables in a varying action flow with corresponding entries from the database.

The process for interpreting optional actions is similar with the process for interpreting variables. The value of customer preference is retrieved from the database in the same way. But instead of replacing the variable in the action flow with the retrieved value, the CBI does a comparison of the retrieved value with the default value called the **key**. A match will unlock the optional action(s) (between the <start_choice> and <end_choice> elements in **Figure 3.18**) and that action(s) is included in the output absolute action flow. Otherwise, the optional action(s) are omitted.

In short, CBI *Case* Interpretation function will transform a "varying" action flow (generated by the *Case* Navigation function) into an "absolute" action flow, which is customer-specific and omits non-preferred actions. An analogy of the varying action flow and absolute action flow would be the class and object in object-oriented programming.

CHAPTER 3

The absolute action flow is then output to the Application Maneuver Mechanisms (AMM) to perform (see Figure 3.15).

Note that the XML parser being used should be a validating parser. This is because we need to specify custom data types and entities by using the XML Schema. A non-validating parser could not understand custom data types and entities since it doesn't read schema files. For the implementation in **Chapter 5**, we use the parser called MSXML.

3.3.2.3 Application Maneuver Mechanisms (AMM)

Application Maneuver Mechanisms (AMM) is the third and final component of AOA to be described. AMM has 2 main responsibilities:

- 1) to perform HWS-symptom capture (Hashed-Window-Symptom), and
- 2) to simulate/carry out the absolute action flow generated by CBI.

For responsibility 1), **Figures 3.6** and **3.7** in **Section 3.3.2.1** have already shown the method of capturing the desktop/window characteristic data. The result is a list of characteristics. In order to generate a HWS-symptom, this list is fed to a RIPEMD-160 hash function to generate a 160bit hash code as output. The algorithm of RIPEMD-160 is listed in (...).

For responsibility 2), AMM uses 3 components:

- 1) an XML parser,
- 2) a toolbox of functions, and
- 3) the Action-Tool Link Information.

The 1st component: the XML parser is the same as the one used by CBI, but in AMM it is used to read the absolute action flow. The 2nd component: a toolbox of functions which can be used singly or in combination, to simulate/carry out the actions specified in the action flow. **Table A.1** of **Appendix A** lists all the currently defined

CHAPTER 3

toolbox functions, from low to high level of abstraction. We denote each toolbox function as a "tool" (albeit a software tool):

The current list of tools is by no means an all-inclusive one. Additional AMM tools can always be written to enable new actions. When the action flow XML Schema file is modified / extended with a new action, then an accompanying new tool or combination of new tools must be written and added to the list of tools. The newly written tools/functions are added to the AMM source code. These are all done by the user of our infrastructure, i.e. the ASP. Note that our proposed infrastructure is intended to be used by ASPs who maintain all the components (including the customer-profile database, XML Schema, AMM tools/functions) of the infrastructure.

For example, other than the 6 fundamental actions (mouse_left_click, keyboard_input, etc), some other higher level actions, such as close_window, terminate_application, run_application, set_checkbox, or even ping_server, can be enabled by writing new tools. Both high level and low level actions are needed.

The 3rd component of AMM is the Action-Tool Link Information. This information is used to logically tie an action with the tool or combination of tools which will simulate the action. The essence is to store the association between the variable tags in the action XML and the parameters in the tools' function definitions. For example, consider the link between the <mouse_drag> action and its tools, in Table A2 of Appendix A. As shown, <mouse_drag> action is simulated by the combination of tools *Get_Item_Location()* & *Mouse_Drag_Displace()*. By referring to this Action-Tool Link Information, the user (ASP) modifies the source codes of AMM to enable it to make use of the two functions to simulate <mouse_drag> action.

Thus, the main issues involved when a new action is defined are to first write the tools to simulate the new action, and then specify the Action-Tool Link Information to associate them together. Action-Tool Link Information is implemented with a GUI. Finally source code of AMM is modified to accommodate the new action and tools.

CHAPTER 3

For example, if <mouse_drag> action is the newly added action, then what needs to be done are:

- 1) Write the codes for the functions (tools) which will be used to simulate the <mouse_drag> action. As in Table A.2, the functions are *int Get_Item_Location (POINT *upperleft, POINT *lowerright; POINT *center; CString caption, CString classname, int width, int height)* and *void Mouse_Drag_Displace(int xl, int yl, int x2, int y2)*.
- 2) Create the Action-Tool Link Information, by using a GUI program (Figure 3.19) .
- 3) Modify source code of AMM to include the newly coded functions and to accommodate the new action.

We believe that the combination of AMM, XML action flow file, and CBI enable a certain degree of automation at client side to manipulate GUI and maneuver applications.

Application Maneuver Mechanisms (AMM) is the third and final component of Application Oriented Automation (AOA). This concludes the description on AOA, the 1st category in our overall proposed infrastructure.

Figure 3.19 GUI program to create Action-Tool Link Information

3.3.3 Supporting Components set

The 2nd category of our overall proposed infrastructure (**Figure 3.1**) is the set of underlying Supporting Components. As shown in **Figure 3.20**, they include the language supports, the 4 common services needed by agents in our model, and an agent shell. We will describe them one by one.

CHAPTER 3

Figure 3.20 Infrastructure's Supporting Components

3.3.3.1 Language Supports

Agents in our infrastructure frequently pass along information, service requests, and replies to each other. This interaction among agents is realized in the form of message exchanges. Language Supports are the group of elements needed by agents, in order to facilitate construction and understanding of messages.

The 1st component of Supporting Components: Language Supports include the message protocol & formats, message schema database, message handling, message parser, and message constructor. **Figure 3.21** shows the elements of Language Support.

Figure 3.21 Language Supports

CHAPTER 3

We do not adopt any standard/recognized agent communication languages (like KQML, FIPA's ACL or KIF), because for agents to communicate effectively both syntactically and semantically, agents must not only share a common language but also must share a common ontology. Since there's still no commonly accepted ontology in the service provision domain and we don't have an ontology developed under this work, therefore we do not see any point in adopting a standard agent communication language because external agents would not be able to communicate effectively with our agents anyway without the ontology. So we turn to XML-based messages as the common language for our agents. A sample message is shown in **Figure 3.22**. (If a widely accepted ontology is formed in the future, our infrastructure would not experience any fatal change. Only the agent messages need to be retooled if that ontology is to be used.)

CHAPTER 3

Our message protocol dictates four compulsory header parts of a message: **1)** message ID, **2)** message nature, **3)** message source, and **4)** message destination. Part 3 and Part 4 indicates the sender agent and intended recipient agent of a message, respectively. The agent_id tag uniquely refers to a specific agent of type agent_type. agent_id is used in case a message must be delivered to a one specific agent out of all the agents that are of the same agent_type. Value of agent_id is assigned by Lord Of The Agents (LOTA, in **Section 33.43**). Part 1 uniquely defines a message sent by a specific agent, by stating the message type and message ID. The format of message ID is conveniently chosen as YYYY_MM_DD_HH_MM_SS_##, easily readable as the ##th message sent in year YYYY, month MM, day DD, hour HH, minute MM, second SS. The combination of message type + message ID will be a unique tag for a certain agent's message. The combination of message type + message ID + message source's agent ID (part 3) will be a unique tag for any message of any agent, for example: "AA03||25_12_2003_17_23_45_01||AID_AA06||". Part 2 indicates the nature of a message, any other message related to this message (example, if it is replying to a previous message), the handling type and priority. Other than the 4 compulsory parts of a message, the actual contents of a message depend on the message type. Every type of message defined has an associated XML Schema file stored in a message schema database.

Those four groups of protocol header information enable messaging handling to be performed. Message handling ensures that messages being passed around the system are processed correctly and are recoverable if lost because of agent malfunctions. Messages being passed always go through an intermediate agent (Coordination agent, described in **Section 3.3.4**) that does message handling. The <message_nature>'s <type> and <initiator_message> provides dialog context information, so that a session built between 2 agents is maintained. By using the header information, a coordination agent enforces a policy to ensure that every message that passes through it is replied to, even if the destination agent somehow hangs. The details are in the discussion on Coordination Agent in **Section 3.3.4**.

All agents need the help of a message parser to read messages. The message parser actually encapsulates an XML parser as the one used in **Section 3.3.2.2 CBI**. When an agent receives a message, it uses the message parser to read the contents in the XML-based message. Conversely, when an agent needs to construct a message of certain type, it uses the help of a message constructor. Based on the type of message to be constructed, the message constructor retrieves the corresponding message schema file from the message schema database. While the message parser only reads a received message and provide the contents to an agent, the message constructor on the other hand will construct a skeleton message (no contents, just tags) based on the message schema file, and then fill in the contents with values provided by the agent. The completed message is then returned to the agent in the form of a long string of character, to be later sent via the communications service. Because all agents need the services of message parser and message constructor, the parser and constructor are implemented as part of a class *MyParserAndConstructor*, so that they can be accessed by creating an object of that class in an agent. The mechanisms for message transport are not grouped together under Language Supports, but are described separately below, as Communications Services.

3.3.3.2 Communications, Registration, and Logging Services

The 2nd, 3rd, and 4th components of Supporting Components are the communications services, registration services, and logging. Since their operation depends on each other, they are implemented together as a *MyCommObject* class. Its header file is listed in **Figure A.2** of **Appendix A**.

This class is inherited from the *CAsyncSocket* class of MFC for windows programming. The communications model uses sockets for transporting stringized form of messages between agents. This class is used by all agents except the core agents (LOTA, Coordination Agents, Usher Agent) because the core agents maintain multiple channels of communication (sockets) at once and involves more complicated management. To use the communications service, an agent must:

- 1) Create a *MyCommObject* object

CHAPTER 3

- 2) Establish a channel with a coordination agent by calling *ConnectToAgent()*
- 3) Register itself with the coordination agent by calling *RegisterAgent()*
- 4) Be implementing an *OnReceive()* function in the agent itself.
- 5) Call *SendMsg()* to send messages
- 6) Call *IAmReady()* when it is ready to process a new message

All agents (except the core agents) are connected to their corresponding coordination agent in our hierarchical model. So when an agent goes online, it must establish a connection with its coordination agent and then registers itself. The registration request will be forwarded by coordination agent to the Lord Of The Agents (LOTA). Based on the agent type, LOTA assigns a unique Agent ID to the new agent. Only after successful registration will the new agent start working i.e. receiving messages from its coordination agent. To receive messages, the agent needs to implement an *OnReceive()* function in itself, because this function will be called by callback function of the agent's *MyCommObject* object when a message arrives. All messages sent and received will be via this object. Whenever the agent is ready to process a new message, it must call *IAmReady()* which will notify the coordination agent to send any next message.

Since sockets are used as communication channels, one coordination agent can handle a maximum of 65536 agents connected to it, as 65536 is the limit for socket ports. If the number of agents ever increases beyond that number, then two (or more) layers of coordination agent are needed to accommodate them.

Once connection and registration have been made, an agent can start recording exceptional events, by sending strings to the *RecordLog()* function. This function appends a simple header "0000" to the original string to be logged, and then uses the *MyCommObject* object to send this specialized message to the coordination agent. The coordination agent immediately recognized this as a log message, not any other messages, and will store it in its log record together with the origin Agent ID and time. We use this specialized message instead of the usual XML messages, to speed up the trivial and frequent logging process. A coordination agent does this for all its "subordinate" agents. Subordinate agents do not store their logs. On the other hand, an agent can also retrieve

CHAPTER 3

its own log record from the coordination agent, by calling the **RetrieveLog()** function. This function requests the coordination agent to send back a message containing a previous logged event. This message also has a "0000" header. The callback function of **MyCommObject** object will not call the **OnReceive()** function but instead calls the **OnEvent()** function of the agent. Thus, to use the logging ability, an agent must:

- 1) Already registered itself
- 2) Use the **RecordLog()** function to stored log records
- 3) Use the **RetrieveLog()** function to retrieve log records
- 4) Be implementing an **OnEvent()** function in the agent itself

Another usage of this logging feature is that subordinate agents can ask their coordination agent to help them store some data for future retrieval.

3.3.3.3 Scheduler Services

The 5th component of Supporting Components is the Scheduler Services. These services enable agents to have more complex usage of the timer functions. For example, Windows programming using MFC only allows 2 direct usage of the timers, i.e. the very basic **SetTimer()** and **KillTimer()** functions. Some agents need more powerful scheduling capabilities, such as an agent which monitors certain parameters of a group of servers every 30 seconds, will need to suspend all its timers when a technician is doing maintenance work on the servers, then resume all its timers after maintenance work are done. That would need more codes to be written besides the two said functions. The Scheduler Services encapsulates those codes and offers agents the convenience of 1) starting a timer, 2) terminating a timer, 3) terminating all timers, 4) suspending a timer, 5) resuming a timer, 6) suspending all timers, 7) resuming all timers, 8) extending a timer, 9) shortening a timer, 10) setting repeating/periodical timers, 11) recalling a timer, 12) setting longer "alarms" other than the usual "milliseconds" timer.

The Scheduler Services are implemented as a **MyScheduler** class. Its header file is listed in **Figure A.3** of **Appendix A**. To use the scheduler services, an agent must:

CHAPTER 3

- 1) Create a *MyScheduler* object
- 2) Be implementing an *OnSchedule()* function in the agent itself.

At every scheduled time/date, *MyScheduler* object's callback function will call the *OnSchedule()* function of the agent. By having a centralized control of the scheduled timers, the Lord Of The Agents can easily cause any agent to suspend its schedule.

3.3.3.4 Agent Shell

The 6th component of the Supporting Components is the Agent Shell. We have prepared an agent template that uses the services described above. Using this agent shell as the foundation, we have written various types of agents for our ASP model (in **Chapter 4** and **Chapter 5**. We show the UML Class view of Agent Shell in **Figure A.4** of **Appendix A**.

The features in our agent shell are:

- 1) Message handling (language) capabilities, by having *MyParserAndConstructor* class header file included and an object of this class created.
- 2) Communications, Registration, and Logging services, by having *MyCommObject* class header file included and an object of this class created.
- 3) Scheduling services, by having *MyScheduler* class header file included and an object of this class created.

Our agent shell has the following functions to be implemented:

- 1) Functions *OnAccept()*, *OnSend()*, *OnConnect()*, *OnClose()*, *OnReceive()*, and *OnEvent()*. These functions will be called by the corresponding call back functions of object *MyCommObject*.
- 2) Function *OnSchedule()*. This function will be called by the call back function of object *MyScheduler*.

CHAPTER 3

- 3) In function ***OnReceive()***, a "***switch***" statement will select the corresponding message handler function to handle each message type.

Writing custom codes for an agent starts by writing the message handler functions for each message type that the agent will receive, and then extending the "***switch***" statement in function ***OnReceive()***.

With that we conclude the description on the 2nd category – Supporting Components – of our overall infrastructure. The next section describes the 3rd and final category.

CHAPTER 3

3.3.4 Core Agents set

All the described components/services in the previous two sections cannot become useful, unless they are being used by agents. In this section, we describe the 3rd and final category of our proposed infrastructure (**Figure 3.1**): The set of three Core Agents. All other agents are collectively denoted as subordinate agents. All subordinate agents in our model (**Chapter 5**) will be written to operate around the Lord Of The Agents (LOTA), Coordination Agents, and Usher Agent.

Figure 3.23 Core Agents of the proposed infrastructure

3.3.4.1 Usher Agent

Usher Agent is the message interchange station. Without it, every agent will be trying to build a communication channel with every other agents, and result in a gigantic mesh of socket connections which means a tightly coupled system. Such a system is not easily scalable. With the Usher Agent, all messages are passed to it, and then routed to the designated targets. For this purpose, Usher Agent examines the `<message-destination>` & `<message_id>` → `<type>` fields in the message by using the

CHAPTER 3

service of Message Parser. Other agents do not need to know the location (address and port) of every other agents, except the ones that are immediately adjacent to them. Referring to **Figure 3.23**, in this way, each subordinate agent (AAA, BBB, CCC, etc) builds only one connection to its coordination agent. Each coordination agent is ready to accept connections from subordinate agents of same type, and also builds only one connection to the Usher Agent. For message passing purpose, all agents (except Usher Agent) need not examine the destination field in the message, and just pass the messages in the same direction until they reach the Usher Agent.

Because of Usher Agent's nature as the central piece, it is also conveniently designated as the entry point to ASP for connections from external agents. Usher Agent receives connection requests from external client agents, but the process for authenticating and approving the requests is delegated to another agent type (Authentication Agent with access to customer-profile, **Chapter 4**). We believe that having Usher Agent as the entry point to ASP not cause major bottleneck problems because Usher Agent itself does not perform any information processing, only routing of messages essentially.

In short, the responsibilities of Usher Agent are to: 1) route messages, 2) be the entry point for external agents' connections, 3) receive external connection requests and then utilize lower level agents to authenticate the requests, 4) receive registration requests from newly on-line coordination agents and then forward to LOTA.

The benefits of Usher Agent are more organized communication channels and less tightly coupled components for easier scaling.

The Usher Agent is not as proactive as any other agents, but in the infrastructure it is still designated as an agent and utilizes the Agent Shell to be built. This is because we wish to maintain the uniformity of an agent community where agents usually communicate with other agents.

CHAPTER 3

3.3.4.2 Coordination Agents

A Coordination Agent is the coordinator of a group of subordinate agents of the same type. There could be more than one Coordination Agents in the infrastructure, one each for every type of subordinate agents. Every time a new type of agents is implemented, a corresponding coordination agent must also be implemented for that agent type. But actually, all coordination agents are functionally the same, except that they handle different message types depending on their subordinate agents.

The basic purpose of a Coordination Agent is to evenly distribute jobs (messages to be processed) to its group of subordinate agents. A Coordination Agent manages a message queue on behalf of its agents. When it receives a "I_Am_Ready" message type from a subordinate agent, it will send one message (if any) from the message queue to the ready agent. But in reality, a Coordination Agent does not simply distribute the messages one by one to every subordinate agent. Message handling is the more important contribution of Coordination Agents.

For message handling purposes, a Coordination Agent uses the message header information as described in **Section 3.3.3.1 Language Supports**. A portion of the header information shown in **Figure 3.22** is reproduced here:

CHAPTER 3

For ease of description, we affix a sign to each element in the header above.

Element (E) (values: 0-5) enables more urgent messages to jump the queue in Coordination Agent. Element (F) (values: "All", "Any", "Specific") specifies whether this message should be passed to ALL subordinate agents (such as instruction from LOTA for all agents to suspend schedule), ANY subordinate agents (normal message), or a SPECIFIC subordinate agent (to continue a series of message exchanges).

Elements (A), (B), (C), and (D) are used for message recovery from error and for maintaining a dialog session. Other than maintaining a message queue, a Coordination Agent also maintains a message buffer for each of its subordinate agents. Every message is copied into the buffer before being sent to a subordinate agent. For a message in buffer that needs a replying message (B) but doesn't receive a specific reply (C) & (D) within a certain timeout, that message in buffer is resent to another subordinate agent. If a reply is received, the Coordination Agent forwards the reply (C) % (D) onwards to the Usher Agent, then deletes the replied to message (A) from the buffer. More complex dialog maintenance and reproduction can be implemented by utilizing these header information.

CHAPTER 3

This adds robustness to the communication services, and allows subordinate agents to be run in less stable machines as compared to the core agents.

Other than message handling, there are other benefits of having Coordination Agents. Without Coordination Agents, all messages queues will be piled up at the Usher Agent, thus causing a entry point bottle neck problem. Thus Coordination Agents take the queues away from Usher Agent. Even though the addition of a layer of Coordination Agents means the introduction of an extra communications channel between subordinate agents and Usher Agent, the effect on efficiency is not significant because the communications are localized (within a server, or within a LAN). We believe that with the flexibility and services offered by Coordination Agents, the minor effect on efficiency can be overlooked.

Also, by setting a hierarchical arrangement of subordinate agents, Coordination Agents, Usher Agent, and LOTA as the infrastructure backbone, the resulting model is very scalable and flexible. Any agent of a new type can be added easily just by having its new Coordination Agent register with Usher Agent. Any new agent of old type can be added easily just by registering with the corresponding existing Coordination Agent. All registration requests ultimately reach the LOTA. All other agents are not affected. New agent registration process is shown in Figure 3.24.

CHAPTER 3

Figure 3.24 Adding and registering a new agent

With this scalability and flexibility, increase in system load (messages to be processed) can be quickly handled and distributed by the easy addition of duplicate subordinate agents. Newly added agents could be either on the same hardware or on newly added hardware, as long as they register themselves. Almost like hot-pluggable. But a more appropriate term should be hot-registerable. The scalable and flexible feature is very suitable for ASPs, which have high concern for maintaining system performance. Removal of a subordinate agent is equally easy by calling the *UnRegisterAgent()* function of *MyCommObject* object for a smooth unregister, or by simply terminating the subordinate agent. The corresponding Coordination Agent will detect the broken socket connection and inform LOTA that an agent has gone off-line. LOTA will delete that agent's ID record.

In short, the responsibilities of a Coordination Agent are: 1) jobs distribution, 2) maintaining message queue, 3) message handling and recovery, 4) receive connection and registrations requests of new agents and then forward to LOTA.

CHAPTER 3

The most significant benefits of Coordination Agents are robustness of communications, scalable and flexible system, and software-based load distribution.

3.3.4.3 Lord Of The Agents (LOTA)

LOTA is the one highest level agent in the infrastructure. The purpose of having LOTA is to have a means to provide ASP personnel a unified view and control of all agents in the system. It happens to be one of two agents (the other one is Technician Agent, **Chapter 4**) at ASP-side that needs a well-built GUI for interaction with humans.

The basic purpose of LOTA is to show a complete listing of all agents' records, including the IP address, port, assigned Agent ID, type and live cycle of all agents. Additional information can be obtained by sending a message to poll all or certain group of agents. For example, LOTA can poll a group of subordinate agents about their scheduled tasks. The retrieved information (by messages) will be displayed in the GUI. Since the messages are in XML format, LOTA can easily build an appropriate display.

Through LOTA, orders can be sent: 1) to an agent to terminate itself, 2) to a coordination agent to disconnect, suspend, or resume a specific subordinate agent, 3) to a coordination agent to terminate itself, 4) to Usher Agent to disconnect, suspend, or resume a specific coordination agent.

The third responsibility of LOTA is to accept and approve registration requests of new internal agents (subordinate agents, Coordination Agents, and Usher Agent). Approved registration will be replied with a unique Agent ID. Therefore, LOTA is the first agent that needs to be run before any other agents, followed by a registration request from Usher Agent, registration request from a Coordination Agent, and then other agents.

LOTA's poll and control messages are exclusive, and are not allowed to be sent by others. To prevent other agents (especially external agents) from sending LOTA exclusive messages with malicious intent, Usher Agent is the regulator. As described earlier in **Section 33.4.1**, before routing a message, Usher Agent will read its `<message_destination>` & `<message_id>→<type>` fields. If the message type is a known

CHAPTER 3

LOTA exclusive message type but is not from the connection from LOTA, then it is not routed. A notification message containing the pretender's IP address, port and Agent ID will be sent to LOTA.

In short, the responsibilities of LOTA are: 1) to provide a GUI, 2) to display information about all other agents, 3) to control all other agents, 4) to receive and approve registration requests from all other agents.

With that we conclude the description on the 3rd and final category – Core Agents – of our overall infrastructure.

Chapter 4 Agent-Based ASP Model

In this chapter, we identify the necessary functionalities required of an agent-based ASP model. We then propose an agent-based ASP model, on top of the infrastructure proposed in **Chapter 3**. The proposed model consists of various components and agents that cater to the needs of users and the ASP. Finally we conclude this chapter with an example of a use case of the ASP model.

4.1 Functionalities in the Agent-Based ASP Model

Every ASP runs various backroom functions to streamline their operations. The fundamental backroom functions are

- 1) **Customer authentication and account management.** The identity claimed by a customer who connects through internet must be authenticated to prevent abuse of service usage. Also, every registered customer has a personal account with confidential personal data.
- 2) **Directory services.** There must be an organized listing of all the applications provided by the ASP, to which customers can gain access to browse for the provided services.
- 3) **Technical support.** As explained in previous chapters, prompt technical support is one of the most crucial elements of a good ASP company. Customers expect to have convenient access to technical help or assistance of technical personnel whenever the need arises.
- 4) **Provisioning.** ASPs usually adopt either the Remote Services or the Network Computer (NC) technologies to provide hosted applications to customers. Applications are hosted in large server farms in order to cater to a large number of simultaneous users.

CHAPTER 4

- 5) **Security.** An ASP, being an online business, has to ensure that stringent network security measures (firewalls, anti-virus utilities, etc) are in place in order to minimize disturbances to service provision. In addition, physical security is equally important to protect the servers from physical dangers such as natural disasters, fire, flood, or even theft.
- 6) **Billing and payment collection.** ASPs usually collect monthly fees from clients. The fees are either a fixed amount or dependent on service usage. For the latter, ASPs will need to have reliable means to record clients' usage of hosted applications. For payment, even though manual cash payment (mail the bill, then wait for the cheque) is a possible method, an automatic payment arrangement is sometimes viewed as more convenient. ASPs may interface with external professional payment systems, such as a payment gateway built by a legal professional organization (for example, credit card company) or clients' banks that support payment on web services.

As explained earlier, we believe that software agents can play a significant role in providing prompt technical support to ASPs' clients. It is the main functionality focused on by of our proposed agent-based ASP model. In addition, in view of the characteristics of other useful backroom fbnctions (customer authentication, directory services, application provisioning, information reporting) as being regularly running and repeating, we decide to implement those functions as agent based functions. With the backroom functions implemented as agents, we have an overall community of interacting software agents that not only takes care of technical support for clients, but also functioning together as a working system to provide a complete service provision experience for clients. Whenever a task is to be performed, it will need the cooperation of several agents. However, our proposed agent-based ASP model at this moment does not include the functionalities for security and payment. This is because those issues involve complicated implementation in view of the numerous real life security threats faced by the internet community. Therefore, security measures are out of the scope of this work.

Thus, in the following we will describe several core functionalities that are to be included in our proposed overall agent-based **ASP** model. The functionalities are listed in **Figure 4.1**.

Figure 4.1 Core Functionalities of Agent-based ASP Model

- a) **User identification.** Naturally, the first step of every customer connection to **ASP** is user identification. This is for future billing purposes and for recording each customer's personal information, preferences and application usage history. Customers' personal data and preferences could be stored using one of three methods: i) the cookies strategy, ii) server-side user data, or iii) a hybrid strategy [60]. A cookie is a small piece of user-related data saved in persistent storage on the client side for future server-side reference. The second method stores all user data at the server side. A hybrid strategy combines the two earlier strategies. We favor a hybrid strategy that is very heavily inclined towards having server-side user data. The argument is that in the **ASP** model, customer data, especially payment and login information, is highly sensitive. In a working environment, it is possible that more than one customer will have access to the same PC. Therefore, it is not secure for this customer data to be stored as plain-text cookies at the client side and then frequently transmitted over the Internet. Thus, having server-side user data is the safer strategy. Also,

CHAPTER 4

with user data being stored at the ASP server, our model presents the access-point-independent convenience. Customers can use any machine at any location to access the same applications at ASP, while continue having access to their usual preferences stored at ASP. At the client local machine, only application session data (if any) are stored. For example, if a user A goes to a PC not used by A before, A just needs to download a generic client agent from the ASP website. After providing the correct username and password to the newly downloaded client agent, the client agent will be able to access A's personal data or preferences stored at ASP and then start serving A.

b) One personal agent for each user. Even though the ASP model, at the highest level, is a one-to-many relationship, customers should feel that they are receiving one-to-one service in a personalized manner. An agent of user should be used. This agent acts on behalf of the user, and serves as an intermediary between the user and the ASP. This agent of user, called the client agent, serves one individual user. What a user needs to do is to download (one time) a generic client agent from the ASP website. Then the user provides username and password to the newly downloaded client agent, which enables the agent to access that particular user's personal data and preferences stored at ASP. Thereafter, the agent will provide one-to-one service to the user [61]. The client agent will remain at the machine, waiting to be executed by the user next time. From the ASP'S point of view, each client agent represents a user. From the user's point of view, his client agent is the sole portal for all interactions with the ASP.

c) Provision of application by remote hosting and downloading. The common ASP paradigm dictates that applications are hosted at servers and to be accessed by clients, without having to download the applications. However, we believe that, for several reasons, clients will occasionally prefer to download and install certain applications locally on their PCs. One reason is the data privacy and security factor. Also, when a client knows that he is going to use an application frequently for a specific period, he would prefer to download and install it (provided that the application is not too complex), rather than

having to remotely access the application every time. Therefore, an ASP model should provide both server-based application hosting and desktop-based application downloading services to clients, so as to cater for different application usage situations. In this way, clients enjoy a certain degree of freedom in choosing between either utilizing the ASP server resources by remotely accessing applications (ala client-server model), or the alternative option of applications downloaded to be run locally. Client Agent also has the potential to function as an interface for users to gain access to various web-based services. In our model, all ASP services are accessed by the user via the client agent. Thus, we believe the Client Agent has the potential to play the role of combining various web-based services (hosted at various servers), and subsequently presenting new functionalities and information to the user, in a standardized way that is easy for the user to comprehend.

- d) **Application directory.** ASP keeps tab on the usage of applications by its customer base. The ASP model should have a provision for maintaining a record of application usage history, installation, deletion, upgrades and application versions used by customers.
- e) **Reporting.** All users should be informed and kept up to date with the situation at ASP side. News such as addition of new software, software upgrades, bugs, patches, monthly bills, emails etc are regularly forwarded to all users, with their consent. Thus an automatic reporting mechanism should be in place.
- f) **Automation and technical support.** The main reason that users adopt ASPs is for the IT support and expertise. Clients view the improved IT support and rapid IT implementation as the most important factor for deciding on using ASPs [3], [10]. Therefore, when we are designing an ASP model, a main functionality to be considered is on providing prompt and efficient technical support. As discussed in **Chapter 1**, many works done by technicians are frequently repeated, on different users' PCs. We believe that some of these regular works/procedures can be recorded and then replayed at a later time, in an automated fashion. Thus an agent-based approach to the ASP model should encapsulate automation for handling these procedures transparently. To enable

CHAPTER 4

those agent-assisted procedures, a knowledge-based approach is adopted, for keeping a record of encountered problems and solution methods. The knowledge base is used by agents in future reference and automatic performing of tasks. The stored knowledge serves as the condition-action rules for agents.

Based on these necessary functionalities, we propose an agent-based **ASP** model in this chapter. In a multi-agent system like this, the model's facilities can be dynamically modified, added or removed easily by replacing the corresponding agents [62].

Also, we decided to use static agents instead of mobile agents. While the usage of mobile agent seems tempting, it is not exactly suitable or necessary for all agent-based applications. There is almost nothing that can be done by mobile agents that cannot be done by other means [63]. The strength of mobile agents is seen primarily in applications where local processing of data need to be done at various distributed sites, in a multi-hop manner where the mobile agent has an itinerary to follow. The mobile agent will migrate from one site to another site in order to complete different aspects of its tasks, before finally returning to its origin. For our **ASP** model, that is not the case. The major part of the agent-based **ASP** infrastructure is internal, that is, at the server side. Only the client agents are external, that is, at the client side. While this client agent could be given mobility capabilities, it is simpler to implement it as a packaged static agent that is dispatched to the client and installed on the client **PC**. No further migration of the agent is ever needed from then onwards. Thus, agents in an **ASP** model do not need to perform multi-hop migration. By omitting the mobility support, security and control issues associated with mobile agents are non-issue in this case [64].

4.2 Overall Agent-based ASP Model

The proposed agent-based ASP model is illustrated in **Figure 4.2**. There are two sides in this model, i.e. the client-side component, and the ASP-side components. There are a total of 11 types of agents in the overall model:

- i) Seven types of subordinate agents at the ASP-side: ASP Server Pool Agents, Information Dissemination Agents, Authentication Agents, Service Directory Agents, FTP Agents, Knowledge Base Agents, and Technician Agents.
- ii) Three types of Core Agents (from **Section 3.3.4**) at the ASP-side: a LOTA Agent, an Usher Agent, and Coordination Agents (seven of them, one for each type of subordinate agents).
- iii) One type of external agent at the client-side: Client Agents (one for each user).

LOTA and Client Agent are implemented independent of the agent shell, because both these agents have very unique tasks compared to the other agents. The other agents are extended from the agent shell.

As defined in **Section 2.2.1**, software agents are programs to which one can delegate tasks, and which will carry out their delegated task autonomously. A further classification of the attributes of agents includes

- 1) autonomy,
- 2) persistence,
- 3) socialability,
- 4) reactivity,
- 5) proactivity,
- 6) personality, and
- 7) mobility.

In regards to the above attributes, for our agent based model:

CHAPTER 4

- 1) The Client Agent together with the Technician Agent behaves in an autonomous manner for providing technical support. Basing on a set of rules (the case base), the agents work together to dynamically form sequence of instructions (action flows) according to the external situation (symptoms). The Knowledge Base Agent also contributes to this task. Whereas for the other agents in the model, they have weak autonomy, in that they perform their tasks according to fixed sequence of instructions when responding to requests from other agents.
- 2) All the agents handle a predefined set of tasks without human intervention, and can be continuously running over long periods of time.
- 3) All the agents are sociable because they exist together is an agent community, inter-communicate, and requires cooperation from each other in order to complete their assigned tasks.
- 4) While most of our agents are reactive in that they respond to messages received from other agents, the Client Agent also proactively monitors the condition at client's PC.
- 5) Also, each agent has a unique identifier assigned by the Lord Of The Agent, such that even identical agents of the same type are distinguishable from each other,
- 6) However, our agents do not have the mobility capability, as explained in the previous section.

All 11 types of agents will be described in detail in the following section.

CHAPTER 4

Figure 4.2 Agent-based ASP Model. Double sided arrow $\longleftrightarrow$ means two way communications.

CHAPTER 4**4.3 Components of the Model**

For each agent in our proposed model, we specify the following features:

- i) Classification: either Core Agent, subordinate agent, or external agent
- ii) Quantity: how many duplicates of the same agent type are active simultaneously in our model
- iii) GUI: any GUI generated
- iv) Communications: direct communications with which agent(s) and/or with non-agent component(s)
- v) Resources: any external resources used
- vi) Information: what information needed and maintained by the agent to fulfill its tasks
- vii) Responsibilities: actions performed by the agent

4.3.1 Lord Of The Agents (LOTA)

Features	Details
Classification	-Core Agent
Quantity	1
GUI	-Yes. -Displays a unified view of the status of other agents in the system. Through the GUI, a human administrator can send commands/messages to control the other agents.
Communications	-With the Usher Agent
Resources	-None
Information	-Information about all agents, including the IP addresses, port, assigned Agent ID, agent type, and live cycle.
Responsibilities	-To provide GUI to administrator -To display information about all other agents -To poll other agents for information -To control/command all other agents

CHAPTER 4

	-To receive and approve registration requests from all other agents -To assign a unique Agent ID to every registered agent
--	---

Table 4.1 Features of Lord Of The Agents (LOTA)

As described in **Section 3.3.4.3** of the previous chapter, LOTA is the highest level agent in our model. The purpose of having LOTA is to have a means to provide ASP personnel a unified view and control of all agents in the system. It is the only agent at ASP-side that has a well-built GUI for interaction with humans. All newly activated agents at the ASP-side need to send a registration request message to LOTA. If approved, LOTA will assign a unique Agent ID to each agent. Therefore, LOTA is the first agent that needs to be executed before other agents are.

As stated earlier, LOTA is uniquely implemented, independent of the agent shell.

4.3.2 Authentication Agent

Features	Details
Classification	-Subordinate Agent
Quantity	≥ 1
GUI	-No
Communications	-With its Coordination Agent -With Authentication Server
Resources	-Authentication Server
Information	-IP address and port number of its Coordination Agent -Connection with Authentication Server
Responsibilities	-To maintain user profiles in the database at Authentication Server -To store and retrieve information from the user profile database -To help Usher Agent to authenticate connection requests from Client Agents

Table 4.2 Features of Authentication Agent

CHAPTER 4

The main purpose of the Authentication Agent is to maintain user profiles. The types of user information stored in the database are: personal information, payment information, log in information, history of application usage, and preferences (e.g. "Preferred application download directory" or "Preferred installation path"). The stored information is sometimes retrieved upon request from other agents, notably the Client Agents and Information Dissemination Agents. The Authentication Agent also helps Usher Agent to authenticate connection requests from Client Agents, by comparing login information provided by the Client Agents with originally stored login information.

A prototype Authentication Agent is implemented by extending the agent shell. A new *AuthenticationAgent* class is created by inheriting the *AgentShellDlg* class (refer to **Figure A.4** of **Appendix A**). The *OnReceive()* function of *AgentShellDlg* class is overwritten and its "switch" statement is extended to include new message handler functions. See **Figure A.5** in **Appendix A**.

4.3.3 Service Directory Agent

Features	Details
Classification	-Subordinate Agent
Quantity	≥ 1
GUI	-No
Communications	-With its Coordination Agent -With Service Directory Server
Resources	-Service Directory Server
Information	-IP address and port number of its Coordination Agent -Connection with Service Directory Server
Responsibilities	-To maintain application profiles in the database at Service Directory Server -To store and retrieve information from the application profile database

Table 4 3 Features of Services Directory Agent

CHAPTER 4

The main purpose of the Service Directory Agent is to maintain application profiles. The types of application information stored in the database are: the usual application parameters (name, developer, version, year, system requirements, etc), locations of applications at FTP Server and ASP Server Pool, password to FTP Server, and application usage history. Application usage history keeps a record of which users have accessed an application, the frequency of access and any comment given. The stored information in application profiles is sometimes retrieved upon request from other agents, notably the Client Agents.

A prototype Service Directory Agent is implemented by extending the agent shell. A new *ServiceDirAgent* class is created by inheriting the *AgentShellDlg* class. See **Figure A.6** in **Appendix A**.

4.3.4 FTP Agent

Features	Details
Classification	-Subordinate Agent
Quantity	≥ 1
GUI	-No
Communications	-With its Coordination Agent -With FTP Server
Resources	-FTP Server
Information	-IP address and port number of its Coordination Agent -Connection with FTP Server -Location of available applications and passwords
Responsibilities	-To provide application downloading through FTP protocol -To manage applications stored in the FTP Server

Table 4.4 Features of FTP Agent

The main purpose of the FTP Agent is to provide application downloading through FTP protocol. It maintains information regarding the location of available applications and the passwords required to download them. The FTP Agent itself also

CHAPTER 4

embeds the FTP Server program, developed using the Nexgen Server Software Development Kit [65].

A prototype FTP Agent is implemented by extending the agent shell. A new *FTPAgent* class is created by inheriting the *AgentShellDlg* class. See **Figure A.7** in **Appendix A**.

4.3.5 Knowledge Base Agent

Features	Details
Classification	-Subordinate Agent
Quantity	≥ 1
GUI	-No
Communications	-With its Coordination Agent -With Knowledge Base Server
Resources	-Knowledge Base Server
Information	-IP address and port number of its Coordination Agent -Connection with Knowledge Base Server
Responsibilities	-To maintain a Knowledge Base of <i>cases</i> in the Knowledge Base Server -To update and retrieve <i>cases</i> from the Knowledge Base

Table 4.5 Features of Knowledge Base Agent

The Knowledge Base Agent implements the functions of the Knowledge Base Subsystem (KBS), of which the details have been described in **Section 3.3.2.1**. Essentially, the KB Agent manages the KB of stored *cases*, combine *cases* near to each other, update the **KB** with newly create *cases*, and tries to find matching *cases* to respond to symptoms observed by Client Agent. If no matching *case* is found, the Technician Agent is alerted.

CHAPTER 4

A prototype Knowledge Base Agent is implemented by extending the agent shell. A new *KnowledgeBaseAgent* class is created by inheriting the *AgentShellDlg* class. See Figure A.8 in Appendix A.

4.3.6 Technician Agent

Features	Details
Classification	-Subordinate Agent
Quantity	≥ 1
GUI	-Yes
Communications	-With its Coordination Agent
Resources	-None
Information	-IP address and port number of its Coordination Agent
Responsibilities	-To display new symptoms for human technician to handle -To record actions performed by human technician and then send to Knowledge Base Agent for storage -To request existing <i>cases</i> from the Knowledge Base Agent, provide a graphical view of the retrieved <i>cases</i> , and allow editing

Table 4.6 Features of Technician Agent

The Technician Agent collaborates closely with the Knowledge Base Agent. It provides a GUI interface for human technicians to respond to help requests (symptoms) from Client Agents and thus generate new *case*. When a Client Agent sends a symptom to the Knowledge Base Agent but a matching *case* cannot be found from the KB Server, the Client Agent will turn to the Technician Agent for help. The technician agent alerts a human technician. An actual screenshot of the symptom / user PC screen is sent by the client agent to the Technician Agent. Through the Technician Agent GUI, the human technician views the screenshot, and performs the necessary mouse/keyboard actions. Those actions are automatically recorded and sent to the KB Agent. A new entry is created in the KB Server table to record this new *case*. Finally, this technician

CHAPTER 4

recommended solution is sent to the Client Agent for execution on the user PC. The combination of KB Agent and Technician Agent allows remote task automation / technical support to be performed by the Client Agent, which save a lot of time and human traveling.

Also, the Technician Agent allows human technician to view existing *cases* in the KB Server and perform editing.

A prototype Technician Agent is implemented by extending the agent shell. A new *TechnicianAgent* class is created by inheriting the *AgentShellDlg* class. See **Figure A.9** in **Appendix A**.

4.3.7 ASP Server Pool Agent

Features	Details
Classification	-Subordinate Agent
Quantity	≥ 1
GUI	-No
Communications	-With its Coordination Agent -With ASP Server Pool
Resources	-No
Information	-IP address and port number of its Coordination Agent -Connection with ASP Server Pool
Responsibilities	-To provide information about the ASP Server Pool to other agents To indirectly manage applications on the ASP Server Pool

Table 4.7 Features of ASP Sewer Pool Agent

The ASP Server Pool Agent acts as the interface between the agent community and the ASP Server Pool. The ASP Server Pool is usually managed by an access infrastructure commercial product, such as the Citrix MetaFrame Access Suite [66]. The Citrix access infrastructure simplifies the deployment and administration of even hundreds of applications on the server pool, and delivers them to users who run the Citrix

CHAPTER 4

ICA thin client software. Thus, the ASP Server Pool Agent embeds the necessary APIs (application programming interface) to interface with the access infrastructure on the Server Pool. The ASP Server Pool Agent will provide information about the status of the Server Pool (e.g. availability of certain applications, load conditions, etc) to other agents, and will also provide an indirect way for control commands to be sent to the access infrastructure.

4.3.8 Coordination Agent

Features	Details
Classification	-Core Agent
Quantity	7, one for each type of subordinate agents
GUI	-No
Communications	-With its own subordinate agents -With the Usher Agent
Resources	-None
Information	-IP address and port number of Usher Agent -IP address, port number, and Agent ID of its subordinate agents
Responsibilities	-Distribute jobs to its subordinate agents -To maintain message queue and message buffers -Message handling and recovery

Table 4.8 Features of Coordination Agent

A Coordination Agent is the coordinator of a group of subordinate agents of the same type. One Coordination Agent is implemented for every type of subordinate agent. Its main task is to evenly distribute jobs (messages to be handled) to its subordinate agents. It also maintains a message queue, thus avoids pilling up of messages at the Usher Agent which may cause an ASP entry point bottleneck problem. As described in **Section 3.3.4.2** of the previous chapter, a Coordination Agent makes use of message buffers and message header information to perform message recovery and allows for more robust

CHAPTER 4

inter-agent communications. The Coordination Agents shield the whole model from any execution problems of subordinate agents.

A prototype Coordination Agent for Authentication Agent is implemented by extending the agent shell. A new *CoorAuthAgent* class is created by inheriting the *AgentShellDlg* class. See **Figure A.10** in **Appendix A**.

4.3.9 Usher Agent

Features	Details
Classification	-Core Agent
Quantity	1
GUI	-No
Communications	-With the LOTA Agent -With all Coordination Agents -With all Client Agents
Resources	-None
Information	-IP address and port number of all Coordination Agents -IP address and port number of LOTA Agent -IP address, port number, and Agent ID of all Client Agents
Responsibilities	-To route all messages exchanged between agents -As the entry point of connections from Client Agents -To authenticate connection requests from Client Agents, by collaborating with the Authentication Agent(s)

Table 4.9 Features of Usher Agent

As described in **Section 33.4.1** of the previous chapter, Usher Agent is the message interchange station. All messages to be exchanged between different agent types are first passed to Usher Agent, and then routed to the designated targets. Usher Agent is also the entry point to the ASP. It receives connection requests from every Client Agent, and calls upon the service of the Authentication Agent(s) to authenticate the identity claimed by the user who is represented by that Client Agent. By having the Usher Agent,

CHAPTER 4

communication channels are more organized because it prevents a gigantic mesh of socket connections from each agent to every other agent. The whole model is less tightly coupled, thus having better scalability.

A prototype Usher Agent is implemented by extending the agent shell. A new *UsherAgent* class is created by inheriting the *AgentShellDlg* class. See **Figure A.11** in **Appendix A**.

4.3.10 Information Dissemination Agent

Features	Details
Classification	-Subordinate Agent
Quantity	≥ 1
GUI	-No
Communications	-With its Coordination Agent
Resources	-None
Information	-IP address and port number of its Coordination Agent
Responsibilities	-To provide automatic notification services to all users -To perform collaborative filtering to recommend applications to users with similar characteristics

Table 4.10 Features of Information Dissemination Agent

The role of the Information Dissemination agent is for providing automatic notification services to all users. This is essentially generation of emails for informing users about billing, application usage dateline, reminder for deleting downloaded application after dateline, application updates and patches, etc. But, users have the option of choosing (recorded under preferences in user profiles) whether this notification process is an information push style or pull style. This is because some users may not appreciate receiving update notifications frequently. Also, a recommender system is deployed, whereby new or previously-unused applications are recommended to users. Collaborative filtering mechanism is used in the recommendation [67]. The principle is that, applications that are used and highly-rated by a group of users with similar user

CHAPTER 4

characteristics (occupation, age, history of applications used, etc), are recommended to another similar user who has never tried that application before. Thus, application recommendations can be broadcast to those what will be potentially interested in it, without annoying other users [68].

An Information Dissemination Agent can be implemented by extending the agent shell. However, we currently do not implement this agent.

4.3.11 Client Agent

Features	Details
Classification	-External Agent
Quantity	≥ 1 , one for each human user
GUI	-Yes -Allows human user to interact with the ASP-side
Communications	-With the Usher Agent
Resources	-None
Information	-IP address and port number of Usher Agent
Responsibilities	-To establish connection with ASP-side -To display list of available services/applications for user to select -To initiate service downloading process -To initiate Remote Services (e.g. Citrix ICA) thin client session -To carry out automated tasks (using AMM & CBI, Chapter 3) -To detect symptoms at client side and notify Knowledge-base Agent at ASP-side -To query users about personal information and preferences

Table 4.11 Features of Client Agent

Client Agent is the only type of component at client-side and its main purpose is to assist its user. It can be freely downloaded from the ASP's website and run. Client Agent is one of the most important components, because from the user's point of view, it is the only way a user can interact with and obtain ASP services. Thus it has a GUI for

CHAPTER 4

user interaction and for displaying any information sent from the ASP, including a list of available services/applications for user to select.

Services selected by the users could be either downloadable or accessed through a Remote Service (such as Citrix ICA) Thin Client session. To initiate the former, Client Agent will collaborate with the FTP Agent at ASP-side. To initiate the latter, Client Agent will collaborate with the ASP Server Pool Agent at ASP-side and then launch the thin client connection software at client-side computer.

An important function of Client Agent is to carry out automated task / technical support procedures, by making use of the embedded Case Base Interpreter (CBI, **Section 3.2.2.2**) and Application Maneuver Mechanisms (AMM, **Section 3.2.2.3**). For this purpose, the Client Agent detects symptoms at the user's PC and notifies the Knowledge-base Agent. If any matching case is retrieved, the Client Agent will navigate and interpret the *case*, generate client-specific action flows, and then carry out the list of action flows on the user's PC.

Also, occasionally the Client Agent will ask the user to provide his/her preferences and personal information. The collected information is sent to the Authentication Agent at ASP-side for storage.

As stated earlier, Client Agent is uniquely implemented, independent of the agent shell. Prototype implementation of Client Agent is shown in **Chapter 5**.

4.4 A Use Case Example

To better illustrate the interactions between different agent types in the proposed Agent-based ASP model, we will present a sample use case in this section to show the steps involved in selecting and installing an application from the ASP to the user PC. A demo system that actually depicts these steps are shown in **Chapter 5**.

CHAPTER 4

Figure 4.3 Sample Use Case

Figure 4.3 shows a sample use case, where the user chooses an application for downloading and followed by installation. There are 4 steps.

In step 1, connection request is sent from Client Agent to Usher Agent. After successful authentication with the Authentication Agent, the connection request is approved and set up between Client Agent and Usher Agent.

In step 2, the user wishes to choose an application. Client Agent sends a message requesting for a list of available applications to Service Directory Agent, routed through Usher Agent. The list is returned to Client Agent via Usher Agent.

CHAPTER 4

In step 3, user selects an application **AA** for downloading. Client Agent sends a download request to FTP Agent, routed through Usher Agent. The installation program of the requested application **AA** is downloaded via FTP protocol, bypassing Usher Agent (since it is not a message).

In step 4, Client Agent installs application **AA** by running the downloaded installation program. A hashed-window-symptom (see Section 3.3.2.1) of the first dialog box of installation program is captured and sent to the Knowledge Base Agent, routed through Usher Agent. A matching *case* is retrieve from **KB** Server by the Knowledge Base Agent, and sent back to the Client Agent via Usher Agent. By navigating the case for "application **AA** installation", Client Agent can automate the whole process of running the **AA** installation program.

Chapter 5 Prototype Implementation

In this chapter, we show our prototype system. We demonstrate an ASP scenario being addressed by our model, where a user chooses an application provided by ASP. The chosen application is subsequently downloaded and installed on the user's PC with the help of client agent. The emphasis is on the roles of technician agent, *case* base, and client agent.

First, we show and explain the user interfaces of technician agent and client agent. Then we show a series of screen shots depicting the process of creating a *case* for a certain application's installation, by using the "Create *case*" function of technician agent. Finally we show another series of screen shots depicting the process whereby client agent downloads and installs said application automatically (utilizing the *case* created earlier).

5.1 *User Interfaces of Technician Agent and Client Agent*

5.1 .1 Technician Agent

The GUI of our prototype implementation of Technician Agent is shown in **Figure 5.1(a)**. It has the basic functions of:

- 1) Create a *case*
- 2) Retrieve a *case*

When button <Create Case> is clicked, the TechnicianAgent window is minimized to a tray icon (**Figure 5.1(b)**). Right-clicking the tray icon will bring up a menu to start creating a case (**Figure 5.1(c)**).The menu items are listed below:

- i) Start Creating Case
- ii) Restart Creating Case
- iii) Capture Symptom
- iv) Recapture Symptom

CHAPTER 5

- v) Start Recording Action Flow
- vi) Save Action Flow
- vii) **Clear** Action Flow
- viii) Save Case
- ix) End

Figure 5.1 Technician Agent GUI – Create case

The purposes of the menu items are quite self-explanatory by the name of the menuitems. Section 5.2 **Using Technician Agent to Create a Case** will show how the menu is used.

CHAPTER 5

On the other hand, when button <Retrieve Case> of TechnicianAgent window is clicked (**Figure 5.2(a)**), a "Retrieve Case" dialog box pops up (**Figure 5.2(b)**).

After specifying the parameterized name of a case in the "Retrieve Case" dialog box, button <Retrieve> is clicked. The retrieved case will be shown in a dialog box with a list of its symptoms and action flows (**Figure 5.2(c)**). Clicking button <Edit> next to the action flow's name will cause program Notepad to be launched for editing the action flow.

CHAPTER 5

Figure 5.2 Technician Agent GUI – Retrieve Case

CHAPTER 5**5.1.2 Client Agent**

The GUI of our prototype implementation of Client Agent is shown in **Figure 5.3**. First, user needs to log on to ASP, by keying in the username and password then clicking button <Log on to ASP> (**Figure 5.3(a)**).

After successful log on, user can choose to invoke one of the services offered by Client Agent. In our prototype, service "Install Application" is implemented. User chooses this service and then click button <Launch service> (**Figure 5.3(b)**).

An "Install Application" dialog box pops up for user to choose the desired application. Then button <OK> is clicked (**Figure 5.3(c)**). An FTP connection will be setup with the FTP Agent at ASP site to download the chosen application (**Figure 5.3(d)**). After finish downloading, the appropriate case for automatically installing this application will be retrieved from Knowledge Base Agent at ASP site. Client Agent will subsequently carry out the case as will be shown in **Section 5.3 Client Agent to Perform Automation**.

CHAPTER 5

(b)

(c)

Figure 5 3 Client Agent GUI – Install Application

5.2 Using Technician Agent to Create a Case

This section shows how we use the Technician Agent to create a *case*. The *case* to be created will be named: <My type.Me.Centurion.Version 1.Program-Install>. This is a *case* that, when carried out by Client Agent, will automatically run the installation program of application Centurion from start to finish. Application Centurion is actually an artificial application created by us. Centurion doesn't do anything, other than having an installation program for itself.

As described earlier in **Section 5.1.1** and in **Figure 5.1**, in order to create a case, button <Create Case> of Technician Agent is clicked to bring up a tray icon. Then, installation program of application Centurion is run. Right clicking the tray icon brings us to the screenshot in **Figure 5.4(a)**.

CHAPTER 5

CHAPTER 5

CHAPTER 5

Figure 5.4 To create a case

In screen of **Figure 5.4(a)**, human technician will click menu item <Start Creating Case> to start creating a *case*. In Technician Agent a *case* variable is created. Menu item <Start Creating Case> is disabled while menu items <Restart Creating Case> and <Capture Symptom> are enabled. See **Figure 5.4(b)**. To save paper space, subsequent screenshots shown in **Figure 5.4(c)** to **5.4(e)** will have their top and left-hand side parts trimmed off.

CHAPTER 5

In screen of **Figure 5.4(b)**, human technician will click menu item <Capture Symptom> to capture the first stage of this case. A Hash-Window-Symptom (as described by **Section 3.3.2.1.2**) of the current dialog box of Centurion Installation Program is captured by Technician Agent. Menu item <Capture Symptom> is disabled while menu items <Recapture Symptom> and <Start Recording Action Flow> are enabled. See **Figure 5.4(c)**.

In screen of **Figure 5.4(c)**, human technician will click menu item <Start Recording Action Flow> to start recording all mouse and keyboard actions. Implementation wise, Technician Agent actually loads a DLL file (Dynamic-Link Library) which sets a system-wide mouse hook and a system-wide keyboard hook in order to record all mouse and keyboard actions performed by human technician. Menu item <Start Recording Action Flow> is disabled while menu items <Save Action Flow> and <Clear Action Flow> are enabled, and then the menu is hidden. See **Figure 5.4(d)**.

In screen of **Figure 5.4(d)**, human technician will perform all the necessary actions. In this case, the necessary action is just simply a left mouse click on the <Next> button. Doing this will cause the second dialog box (second stage) to come out. Before doing anything, the action flow for first dialog box (first stage) must be saved. For that, human technician must right-click the tray icon to bring up the tray icon menu again. See **Figure 5.4(e)**.

In screen of **Figure 5.4(e)**, human technician will click menu item <Save Action Flow> to have Technician Agent save the recorded action flow for the previous symptom. Menu items <Save Action Flow> and <Clear Action Flow> are disabled while menu item <Capture Symptom> is enabled. See **Figure 5.4(f)**. The menu will look almost exactly like the menu in **Figure 5.4(b)**, except that menu item <Save Case> is enabled.

There are four remaining stages (dialog boxes), as shown in **Figures 5.4(g) – 5.4(j)**. Subsequent rounds of capturing symptom and recording action flow like in **Figures 5.4(b) – 5.4(f)** are repeated for each of those remaining stages. Only difference is

CHAPTER 5

that in these subsequent rounds, menu item <Save Case> will also be enabled, unlike in the first round where menu item <Save Case> was never enabled.

Figure 5.5 Saving a case

In the end, after the action flow for the sixth (last dialog box, **Figure 5.5(a)**) is performed, the "Centurion Installation Program" exits. This last action flow is recorded (**Figure 5.5(b)**). Then human technician will click on menu item <Save Case> (**Figure**

CHAPTER 5

5.5(c)). A dialog box will pop up for human technician to enter the parameterized name for this newly created *case*. See **Figure 5.5(d)**.

Among the *case* creation rounds, there are two scenarios worth noting.

Scenario one, for the third dialog box (**Figure 5.6(a)**), the string entered by human technician into the "Name" field is "%Customer_Name%", so that when the *case* is carried out next time, the actual customer's name is fetch from customer profile (Authentication Agent) and keyed in. The action flow looks somewhat like in **Figure 5.6 (b)**.

Scenario two, for the fourth dialog box (**Figure 5.7(a)**), options are involved. As described previously in **Figure 3.13** of **Section 3**, the action flow for this symptom must be enclosed with a pair of <start_choice> and <end_choice> elements. Currently this pair of elements is added manually by human technician after the *case* is finished created, so that the action flow looks somewhat like in **Figure 5.7(b)**.

CHAPTER 5

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<actions>
  ...
  ...
  ...
  <keyboard_input>
    <input_string>%Customer Name%</input_string>
    <window_caption>Name</window_caption>
    <class name>Edit</class name>
    <width>182</width>
    <height>23</height>
    <window_style>2011234304</window_style>
  </keyboard_input>
  ...
  ...
  ...
</actions>
```

(b)

Figure 5.6 Having variable value in action flow

(a)

CHAPTER 5

```

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<actions>

  <start_choice>
    <%Preference_Installation_Size%>Typical</%Preference_Installation_Size%>
    <!--Comment: Please choose preferred installation type -->
  </start_choice>

  ...
  ...
  ...
  ...

  <end_choice>
    <%Preference_Installation_Size%>Typical</%Preference_Installation_Size%>
  </end_choice>

</actions>

```

(b)

Figure 5.7 A stage with choice

The created *case* can be represented as in **Figure 5.8(a)**, where **ID1** = <My type.Me.Centurion.Version 1.Program-Install>. If two more *cases* are created for the same installation program, but this time with installation options "Minimum" and "Maximum" respectively selected for the fourth dialog box (**Figure 5.4(h)**) instead of selecting "Typical", then we have two more *cases* almost similar to **Figure 5.8(a)**, except with different **AF**(4,5), and **S5** for installation sizes "Minimum" and "Maximum". Combination of these three *cases* will produce the *case* in **Figure 5.8(b)**.

(a)

(b)

Figure 5.8 Case ID1 = <My type.Me.Centurion.Version 1.Program-Install>

5.3 Client Agent to Perform Automation

Now we show how the *case* created in the previous section is utilized.

Let's consider a user who wants to acquire application Centurion from ASP, but the user's PC display has different resolution (800x600 pixels) and color from the display of human technician's PC (1024x768 pixels) when the *case* was created.

The user's first move is to invoke Client Agent and then to log on to the ASP, as shown in **Figure 5.9(a)**. In doing this, Authentication Agent of ASP is involved (refer to **Figure 4.3**).

Second move, user chooses the service wanted (which is "Install Application") and then launch it (**Figure 5.9(b)**). Next, user chooses the application wanted (**Figure 5.9(c)**). In doing this, Service Directory Agent of ASP is involved.

Subsequently, Client Agent will proceed to download the chosen application's installation program (**Figure 5.9(d)**). In doing this, FTP Agent of ASP is involved. Upon finished downloading, Client Agent will communicate with Knowledge Base Agent of ASP in order to retrieve *case* `<My type.Me.Centurion.Version 1.Program-Install>` which was created previously.

Next, Client Agent will run the downloaded installation program. Client Agent's window is minimized (**Figure 5.9(e)**). Notice the differences between the current resolution and color in **Figure 5.9(e)** and the resolution and color in **Figure 5.4(b)** when the *case* was created. These differences will not prevent the *case* from being carried out successfully.

The first (**Figure 5.9(e)**) and second (**Figure 5.9(f)**) dialog boxes (i.e. stages) are handled by the corresponding actions flows from the *case*.

CHAPTER 5

(a)

(b)

CHAPTER 5

(c)

(d)

CHAPTER 5

(e)

(f)

CHAPTER 5

CHAPTER 5

(i)

(j)

CHAPTER 5

(k)

Figure 5.9 Client Agent making use of created **case**

For the third dialog box (Figure 5.9(g)), Client Agent will retrieve the user's name from ASP (Authentication Agent) to substitute for the %Customer_Name% keyboard input variable and it is keyed in. For this user, the name is "Mengke Batee".

For the fourth dialog box (Figure 5.9(h) & Figure 5.9(i)), choice is involved, either "Typical", "Minimum", or "Maximum" installation. If the user's choice is not readily known (not yet stored in user profile), we need to acquire the user's choice value for this first time. In our prototype implementation, we choose to have Client Agent prompt user by popping up a dialog box with a question/request (taken directly from the "Comment" tag of action flow, see **Figure 5.7(b)**). User conveys his/her choice by selecting from the drop-down list box. This choice is stored into user profile, to be used in future occurrence of a similar choice condition.

CHAPTER 5

Then, the rest of the dialog boxes (Figures 5.9(j) & (k)) are handled by the corresponding actions flows from the case.

This prototype implementation shows that by utilizing our non-static case, we can automate the installation of a program on user's computer. Differences in resolution, color, and window positions do not affect the effectiveness of the case. Even if the installation process takes a long time, minimal or no user intervention is required.

Chapter 6 Discussion and Conclusion

6.1 Discussion

The previous chapter depicted a proof-of-concept implementation that accomplished the following: i) creation of a case, ii) retrieval and execution of a case, and iii) reusability of a case under different environment. Specifically, a software issue (setting up an application) was handled successfully with a recorded case. If we can define suitable methods of abstracting and representing the symptoms of a certain condition / process and the solution procedures, it is possible to develop a mechanism that detects a previously seen condition and reproduce the corresponding recorded solution.

In Chapter 4, it was shown that an arrangement of various types of software agents that cater to the requirements of an ASP scenario can be implemented by making use of the proposed Supporting Components set and Core Agents set. In this combination of an ASP setting (being consisted of repetitive and routine backroom functions) with software agents, we realized various operational functions of ASP by defining instances of communication messages and subroutines to handle the tasks in a certain function. Implementation was rather straightforward by making use of the common agent functionalities already provided by the Supporting Components set.

A quantitative proof (such as a well defined cost function) would be the most compelling argument to demonstrate the efficiency of our proposal and its advantages. But such data is simply not yet available, unless the proposal can be tested in an actual ASP. This is because we would require data such as: what portion of overall technical solutions can be recorded as cases, what is the frequency of recurring problems faced by technicians, and also the actual client composition of a certain ASP (how many clients subscribe to a certain service/application). Given this situation, we instead provide a qualitative analysis of the efficiency of our system

By the word 'efficient', we refer to a system that is able to contribute towards the reduction of time, manpower, and / or money spent by an ASP. The combined

effect of the roles of client agents and a KBS is that new problems and solutions can be captured and stored as cases. Capturing and storing a case requires a one-time effort by a technical staff. A knowledge base system (server & storage resources) needs to be procured, plus the service of a knowledge base expert to manage the KBS. On the other hand, stored cases can be reused for multiple times. Every time a case is invoked and reused, time is saved because the associated problem is handled in a prompt manner with minimal human technician involvement. For this particular problem, the client need not initiate a conversation or request for service from technician. Technician also need not spend time to personally travel to client's site to view the problem and provide the solution. Thus the overall process from discovery of problem to application of the corresponding solution is faster. Manpower is saved because the technician is freed to engage other tasks which may require a higher degree of human attention. Money is saved because no traveling cost is incurred. The combined amount of manpower saved translates to a reduced number of technical staff to hire. Overall, an ASP can save its operational costs.

We believe that the aggregated savings from our proposal is far greater than the additional costs (KBS hardware and KBS expert). To put things in perspective, our system is more efficient in the long term under such circumstances: i) a larger portion of overall technical solutions can be represented as cases, ii) a higher frequency of recurring problems faced by technicians, and iii) a larger number of clients subscribe to a certain service /application. If these circumstances are met, then the reuse rate of a case is high, i.e. the case is more valuable. Moreover, by combining and consolidating individual cases into compound cases in the KBS, individual cases learn new knowledge that adds to their effectiveness.

In regards to the scalability of our proposal, in Chapter 4 it was described that our agent-based ASP model consists of various agents that are still loosely coupled to each other. Each agent needs to know about its immediate adjacent agents only. When there is an increase in system load (more clients to be served), additional multiple agents of a particular type can be added into the overall system to handle the service requests, without affecting the operation of other agents. The sole Usher Agent does not pose a great danger as a bottleneck because it has a very limited amount of data

processing role. If a rare need arises it is not impossible to have more than one Usher Agent.

A valid concern is the issue of security. Under usual circumstances, our system should operate within the boundaries of secured connections that already exist between an ASP and each of its clients. Otherwise, there remain two significant aspects of security. One is the process involved in transferring personal data from user's PC to ASP center. Obviously, an adequate level of encryption is needed to protect user's privacy in case the connection is compromised or eavesdropped. The other aspect is how to prove the authenticity of a case that is received by a client PC. Before executing the case, it must be established that the case is genuine, i.e. it indeed came from the ASP in an unaltered manner and is not fabricated by external parties who may have malicious intentions. A normal practice is to have digital signature procedures in place.

6.2 Conclusion

This thesis proposes an efficient and scalable infrastructure for an agent-based ASP model, with emphasis on ASP side task automation and client-side automation/technical support. The proposed infrastructure enables ASP backroom functions to be easily implemented as agents are incorporated into the overall model, which is highly scalable and modular. Also, by using a "Knowledge Base plus Agent" approach, client-side task automation and technical support are realized in a client-PC-adaptive manner. We have defined cases that are small in size, highly reusable, non-static, and extensible (able to include newly acquired knowledge), for storing captured knowledge. Essentially, a case is used to store the steps/procedures taken while performing a newly encountered task, so that the case can be retrieved for automatically handling the same task at another PC or at a latter time. Overall, the proposed infrastructure enables the savings of human manpower and time, by capitalizing on the fact that many ASP tasks are often repeated.

The proposed infrastructure improves on current task automation methods by addressing some of their shortcomings. The shortcomings include:

1. Lack of a mechanism to properly classify, group, and distribute automation tasks that are created.
2. Automation tasks created with existing software tools are not versatile / dynamic enough to be utilized for handling different users.
3. Created tasks are very vulnerable to changes in application.
4. The way to identify a window is not effective.

In view of those shortcomings, our proposed infrastructure systematically classify, group and distribute automation tasks (denoted as "cases"), plus invoke the stored tasks for future usage. We also defined generic format for tasks, which include links to other sources such as to customer profile database. Furthermore, in our proposal, created tasks are much less vulnerable to changes in application layout, whereby in our approach the proper execution of an automation task is much less dependent on the GUI layout/positioning of window components (buttons, text boxes, etc). Thus our approach is comparatively more robust. (Though, major GUI changes, such as from Windows 98 to Windows XP, cannot be accounted for.) We have also proposed an approach to capture the context of a window by gathering information regarding all the components in a window. By this way, windows are much more distinguishable from each other.

Based on the proposed infrastructure of agent-based ASP model, we have developed a prototype system. The prototype implementation shows that by utilizing our case, we can automate the installation of a program on user's computer. Differences in resolution, color, and window positions do not affect the effectiveness of the case. Even if the installation process takes a long time, minimal or no user intervention is required. The result is encouraging as it is a proof of concept that our proposal is viable in a real-life situation.

There are several areas of the proposed infrastructure that offer possibility of future augmentation and enhancement. One area is refinement of the defined structure for a case. Currently, it has weakness in accounting for looping conditions, whereby a stage might need to be repeatedly visited until a condition is satisfied. For example, at a certain stage in a case the client agent might need to query the PC or ask the user to provide (enter) several set of data consecutively. In a real world, not all *cases* will run

from start to finish in a rigid or 100% uninterrupted manner. It is desirable to have provisions to handle these situations.

Secondly, further research can be done on how to determine which elements (components) of a stage should be included as the defining elements of a captured symptom. Certain elements are actually not part of the distinguishing factors. For example, for *cases* that deal with Microsoft Word, many symptoms will include a scroll bar and menu bar, which are generic elements. If information that pinpoints the distinguishing elements of a symptom can be extracted, then a symptom will be much more immune against major changes in the generic elements.

Also, our implementation of the proposed infrastructure is mainly Microsoft Windows based, even though the underlying concepts of the infrastructure (application-oriented automation set, supporting components set, core agents set) are not limited to any specific platform. It would be interesting to explore on how viable is the infrastructure on other platforms (such as Linux), specifically the impact on the current definition of our symptoms.

References

- [1]Richard Comerford, "The New Software Paladins", IEEE Spectrum, pages 56-61, June 2000.
- [2]Susan J. Shepard, "IT Shops Take Stock of Application Service Providers", IEEE IT Pro, pages 12-16, March/April, 2000.
- [3]Anjana Susarla, Anitesh **Barua**, Andrew B. Whinston, "Myths about Outsourcing to Application Service Providers", IEEE IT Pro, pages 32-35, May/June 2001.
- [4]Chad Vawter, Ed Roman, "J2EE vs. Microsoft .NET", The MIDDLEWARE Company, June 2001.
<http://www.theserverside.com/articles/article.tss?l=J2EE-vs-DOTNET>
- [5]Jon Surmacz, "Alive And Kicking", CIO.com, 18th July 2001.
<http://www2.cio.com/metrics/2001/metric237.htm>
- [6]Dan Muse, "The Future Looks Bright for ASPs", 27thMarch 2002.
<http://www.aspnews.com/news/weekly/article.php/999251>
- [7]Bruce McCracken, "Small to Mid-Sized Businesses Turn to ASPs; Larger Enterprises Follow Their Lead", Outsourcing Journal, August 2004.
<http://www.outsourcing-journal.com/aug2004-asp.html?email005913>
- [8]ASPscope.com, "Top 5 ASPs", 1st February 2005.
<http://aspscope.com/pages/top25/top5.htm>
- [9]Renee Boucher Ferguson, "IBM to Acquire Application Service Provider Corio", 25th January 2005.
<http://www.eweek.com/article2/0,1759,1754514,00.asp>

References

- [10] Lixin Tao, "Shifting Paradigms with the Application Service Provider Model", IEEE Computer 34(10), pages 32-39, 2001.
- [11] Anjana Susarla, Anitesh Barua, Andrew B. Whinston, "Making the Most out of an ASP Relationship", IEEE IT Pro, pages 63-67, November/December, 2001.
- [12] Hahn Tram, "The ASP Model for Energy Delivery Information Systems", Proceedings of the 2001 Transmission and Distribution Conference and Exposition, (2001 IEEE/PES), pages 754-758, 2001.
- [13] David Kirkpatrick, "The End Of Software?", FORTUNE, 21st May 2002.
http://www.fortune.com/indexw.jhtml?channel=artcol.jhtml&doc_id=208052
- [14] Michael Wooldridge, Nicholas R. Jennings, "Intelligent Agents: Theory and Practice", The Knowledge Engineering Review 10(2), pages 115-152, 1995.
- [15] Kenneth R. Walsh, "Analyzing the Application ASP Concept: Technologies, Economies, and Strategies", Communications of the ACM 46(8), pages 103-107, August 2003.
- [16] Phil Wainewright, "Industry Basics: Inside Infrastructure Service Providers", ASPnews.com, 10th January 2002.
http://www.aspnews.com/strategies/asp_basics/article.php/715761
- [17] Nicole Pleasants Lane, "Developing a Spend Management Strategy for Maximum supply Chain Savings", USInternetworking, 2003.
http://www.usi.com/documents/Spend_Management_FINAL.pdf
- [18] Michael Berger, Bernhard Bauer, Michael Watzke, "Towards an Agent-Based Infrastructure for Distributed Virtual Organizations", Proceedings of the Tenth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pages 354-355, 2001.

- [19]Nicholas R. Jennings, "An Agent-based Approach for Building Complex Software Systems", Communications of the ACM 44(4), pages 35-41, April 2001.
- [20]Huinan Xu, Phil Seltsikas, "Evolving the ASP Business Model: Web Service Provision in the Grid Era", Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing (P2P' 02), pages152-159,2002.
- [21]"Server-based Computing White Paper", Citrix Systems, Inc., 1999.
<http://www-tus.csx.cam.ac.uk/techlink/workshops/atlantic/atlantic7.pdf>
- [22]Borko Furht, Chris Phoenix, John Yin, Zijad Aganovic, "An Innovative Internet Architecture for Application Service Providers", Proceedings of the 33rdIEEE Hawaii International Conference on System Sciences, pages 1-10,2000.
- [23]Tyson Gill, "Developing and Deploying Applications Using Remote Services", informIT.com, 6thDecember 2002.
[http://www.informit.com/isapi/product_id~\(F956E0AI-68D9-4F28-962E-72304B17DD1F\)/session_id~\(DA9E32FC-30B7-4137-B55C-7C322ABB57F8\)/content/index.asp](http://www.informit.com/isapi/product_id~(F956E0AI-68D9-4F28-962E-72304B17DD1F)/session_id~(DA9E32FC-30B7-4137-B55C-7C322ABB57F8)/content/index.asp)
- [24]David Friedlander, "Market Overview: Server-Based Computing", Forrester Research, 11th February 2002.
<http://www.forrester.com/go?docid=25631>
- [25]Piroz Mohseni., "The Waite Group's NC Guide: Development Strategies for Network Computers", Waite Group Press, Corte Medera, CA, 1997.
- [26]Jose M. Vidal, Paul A. Buhler, Michael N. Huhns, "Inside an Agent", IEEE Internet Computing, pages 82-86, Jan/Feb 2001.
- [27]David Kotz, Robert S. Gray, "Mobile Agents and the Future of the Internet", ACM Operating Systems Review, pages 7-13, August 1999.

References

- [28]Mike P. Papazoglou, "Agent-Oriented Technology in Support of E-Business", Communications of the ACM 44(4), pages 71-77, April 2001.
- [29]Nathan J. Muller, "Improving Network Operations with Intelligent Agents", International Journal of Network Management, Vol. 7, pages 116-126, 1997.
- [30]Roberto A. Flores-Mendez, "Towards a Standardization of Multi-Agent System Frameworks", ACM Crossroads, 1999.
<http://www.acm.org/crossroads/xrds5-4/multiagent.html>
- [31]Gaku Yamamoto, Hideki Tai, "Agent Server Technology for Next Generation of Web Applications", Proceedings of the 4th IEEE International Conference on Computational Intelligence and Multimedia Applications (ICCIMA), pages 82-86, 2001.
- [32]Danny B. Lange, Mitsuru Oshima, "Seven Good Reasons for Mobile Agents", Communications of the ACM 42(3), pages 88-89, March 1999.
- [33]Sun Yubing, Peng Honghui, Lin Zuoquan, Hu Dekun, "A Software Agent System for News Information Delivery on Internet", Proceedings of the 23th International Conference on Information Technology Interfaces (ITI 2001), pages 151-158, June 2001.
- [34]Paulo Marques, Paulo Simoes, Luis Silva, Fernando Boavida, Joao Silva, "Providing Applications with Mobile Agent Technology", Proceedings of the 4th IEEE Conference on Open Architecture and Network Programming (OPENARCH 2001), pages 129-136, April 2001.
- [35]Ouahiba Fouial, Nikos Houssos, Nadia Boukhatem, "Software Downloading Solutions for Mobile Value-Added Service Provision", CiteSeer Scientific Literature Digital Library, 2000.
<http://citeseer.nj.nec.com/fouial00software.html>

- [36]Eduardo Mena, Arantza Illarramendi, Alfredo Goni, "Customizable Software Retrieval Facility for Mobile Computers using Agents", Proceedings of the Seventh IEEE International Conference on Parallel and Distributed Systems (ICPADS'2000),pages 479-484, July 2000.
- [37]Eduardo Mena, Arantza Illarramendi, Alfiedo Goni, "A Software Retrieval Service based on Knowledge-Driven Agents", Proceedings of Fifth IFCIS International Conference on Cooperative Information Systems (CoopIS'2000), pages 174-185, September 2000.
- [38]Alfredo Goni, Arantza Illarramendi, Eduardo Mena, Yolanda Villate, J. Rodriguez, "Antarctica: a Multiagent System for Internet Data Services in a Wireless Computing Framework", NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Springer series of Lecture Notes in Computer Science (LNCS 253812002) pages 119-135, October 2002.
- [39]Matthias Klusch, Stefano Lodi, Gianluca Moro, "Agent-Based Distributed Data Mining: The KDEC Scheme", Springer series of Lecture Notes in Computer Science (LNCS 258612003) pages 104-122, July 2003.
- [40]Hillol Kargupta, Byung-Hoon Park, Daryl E. Hershberger, Erik L. Johnson, "Collective Data Mining: A New Perspective Towards Distributed Data Mining", Advances In Distributed and Parallel Knowledge Discovery, AAAI/MIT Press, pages 131-178,2000.
- [41]Hillol Kargupta, Ilker Hamzaoglu, Brian Stafford, "Scalable, Distributed Data Mining Using an Agent-Based Architecture", Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD-97), pages 211-214, 1997.
- [42]Salvatore J. Stolfo, Andreas L. Prodrromidis, Shelly Tselepis, Wenke Lee, Dave W. Fan, Phillip K. Chan, "JAM: Java Agents for Meta-Learning Over Distributed Database", Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD-97), pages 74-81,1997.

- [43]Stuart Bailey, Robert Grossman, Sivakumar, H., Andrei Turinsky, "Papyrus: A System for Data Mining over Local and Wide Area Clusters and Super-Clusters", Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, (CDROM), article 63, 1999.
- [44]S. Krishnaswamy, A. Zaslavsky, S.W. Loke, "An Architecture to Support Distributed Data Mining Services in E-Commerce Environment", 2nd IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS 2000), pages 239-246, June 2000.
- [45]Stavros Papastavrou, George Samaras, Evaggelia Pitoura, "Mobile Agents for World Wide Web Distributed Database Access", IEEE Transactions on Knowledge and Data Engineering 12(5), pages 802-820, September/October 2000.
- [46]Yolanda Villate, David Gil, Alfredo Goni, Arantza Illarramendi, "Mobile Agents for Providing Mobile Computers with Data Services", Proceedings of the Ninth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 98), October 1998.
<http://citeseer.ist.psu.edu/villate98mobile.html>
- [47]Boris Libermann, Frank Griffel, Michael Merz, Winfried Lamersdorf, "Java-Based Mobile Agents – How to Migrate, Persist, and Interact on Electronic Service Markets", Proceedings of the 1st International Workshop on Mobile Agents (MA97), pages 27-38, April 1997.
- [48]Thomas Megadanz, Kurt Rotherlem, Sven **Kruse**, "Intelligent Agents: an Emerging Technology for Next Generation Telecommunications", Proceedings 15th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM 1996), pages 464-472, 1996.

- [49]Vicenza Carchiolo, Michele Malgeri, Giuseppe Mangioni, "An Agent Based Platform for a Service Provider", Proceedings of the 2nd IMACS International Conference on Circuits, Systems and Computers (CSC'98), October 1998.
<http://citeseer.nj.nec.com/91085.html>
- [50]Tsung-Chuan Huang, Chu-Sing Yang, Sheng-Wen Bai, Szu-Hsuan Wang, "An Agent and Profile Management System for Mobile Users and Service Providers", Proceedings of the 17th IEEE International Conference on Advanced Information Networking and Applications (AINA '03), pages 574-577, March 2003.
- [51]Takahiro Kawamura, Tetsuo Hasegawa, Akihiko Ohsuga, "Rental Application to Rental Service Development of Advanced ASP Framework", Proceedings of the 4th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2000), pages 58-64, September 2000.
- [52]Gautam Kar, Alexander Keller, Seraphin B. Calo, "Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis", Proceedings of the Seventh IEEE/IFIP Network Operations and Management Symposium (NOMS 2000), pages 61-74, April 2000.
- [53]Gautam Kar, Alexandar Keller, "An Architecture for Managing Application Services over Global Networks", Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM 2001), pages 1020-1027, April 2001.
- [54]Automize <http://www.hiteksoftware.com>
- [55]AutoIt <http://www.hiddensoft.com/AutoIt>
- [56]EMac <http://www.evansprogramming.com/emacs.asp>
- [57]Macro Express <http://www.macroexpress.com>
- [58]AutoMate <http://www.unisyn.com/automate>

References

- [59]Robert C. Martin, "UML Tutorial: Finite State Machines", ObjectMentor.com, Engineering Notebook Column, June 1998.
<http://www.objectmentor.com/resources/articles/umlfsm.pdf>
- [60]Weiguang Shao, Wei-Tek Tsai, Sanjai Rayadurgam, Robert Lai, "An Agent Architecture for Supporting Individualized Services in Internet Applications", Proceedings of the 10th IEEE International Conference on Tools with Artificial Intelligence, pages 140-147, November 1998.
- [61]Takeshi Okuda, Makoto Takano, Koichi Tajima, Shinji Shimojo, Suguru Yamaguchi, Hideo Miyahara, "Realizing Continuous and Transparent Service Using Agents", Proceedings of the 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, (PACRIM'01), Vol. 2, pages 736-739,2001.
- [62]Yoshimichi Tanizawa, Ichiro Satoh, Yuichiro Anzai, "A Mobile Agent-Based Framework to Support Personal Mobility", Proceedings of 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/ Distributed Computing, International Association for Computer and Information Series, pages 144-151,2001.
- [63]Detlef Schoder, Torsten Eymann, "The Real Challenges of Mobile Agents", Communications of the ACM, Vol. 43, No. 6, pages 111-112, June 2000.
- [64]Paolo Bellavista, Antonio Corradi, Cesare Stefanelli, "Monitor and Control of Mobile Agent Applications", ACM OOPSLA Workshop on Experiences with Autonomous Mobile Objects and Agent Based Systems", Oct 2000.
- [65]NexgenServer Software Development Kit <http://www.nexgenserver.com>
- [66]Citrix MetaFrame Access Suite <http://www.citrix.com>

- [67]Pattie Maes, Robert H. Guttman, Alexandros G. Moukas, "Agents That Buy and Sell", Communications of the ACM, Vol. 42, No. 3, pages 81-91, March 1999.
- [68]Simon Case, Nader Azarmi, Marcus Thint, Takeshi Ohtani, "Enhancing E-Communities with Agent-Based Systems", IEEE Computer, Vol. 34, Issue 7, pages 64-69, July 2001.

Appendix A

This appendix consists of some figures and tables for Chapter 3.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.ntuccs.com"
  xmlns="http://www.ntuccs.com"
  elementFormDefault="qualified">

  <xs:element name="actions">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:choice>

          <xs:element name="mouse_left_click">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="window_caption" type="xs:string" />
                <xs:element name="class_name" type="xs:string" />
                <xs:element name="width" type="xs:integer" />
                <xs:element name="height" type="xs:integer" />
                <xs:element name="window_style" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>

          <xs:element name="mouse_right_click">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="window_caption" type="xs:string" />
                <xs:element name="class_name" type="xs:string" />
                <xs:element name="width" type="xs:integer" />
                <xs:element name="height" type="xs:integer" />
                <xs:element name="window_style" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>

          <xs:element name="mouse_double_left_click">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="window_caption" type="xs:string" />
                <xs:element name="class_name" type="xs:string" />
                <xs:element name="width" type="xs:integer" />
                <xs:element name="height" type="xs:integer" />
                <xs:element name="window_style" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>

          <xs:element name="keyboard_input">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="input_string" type="xs:string" />
                <xs:element name="window_caption" type="xs:string" />
                <xs:element name="class_name" type="xs:string" />
                <xs:element name="width" type="xs:integer" />
                <xs:element name="height" type="xs:integer" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Appendix A

```

        <xs:element name="window_style" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="mouse_drag">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="x_axis_movement" type="xs:integer" />
        <xs:element name="y_axis_movement" type="xs:integer" />
        <xs:element name="window_caption" type="xs:string" />
        <xs:element name="class_name" type="xs:string" />
        <xs:element name="width" type="xs:integer" />
        <xs:element name="height" type="xs:integer" />
        <xs:element name="window_style" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="mouse_point">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="window_caption" type="xs:string" />
        <xs:element name="class_name" type="xs:string" />
        <xs:element name="width" type="xs:integer" />
        <xs:element name="height" type="xs:integer" />
        <xs:element name="window_style" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="wait_milliseconds" type="xs:integer" />

  <xs:element name="start_choice">
    <xs:complexType>
      <xs:element name="%Preference_Desktop_Shortcut%" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="end_choice">
    <xs:complexType>
      <xs:element name="%Preference_Desktop_Shortcut%" type="xs:string" />
    </xs:complexType>
  </xs:element>

  </xs:choice>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

Figure A.1 XML Schema for the action flow XML file

Tools	Description
<code>void Block_Input(int yesno);</code>	To block or enable all user mouse and keyboard input.
<code>void Wait_Milliseconds(int x);</code>	Pause for <i>x</i> milliseconds.
<code>void Mouse_Button_Down(CString button);</code>	Mouse button <i>button</i> pressed down.
<code>void Mouse_Button_Up(CString button);</code>	Mouse button <i>button</i> released.
<code>void Mouse_Click(CString button, int x=100);</code>	Mouse button <i>button</i> clicked, clicking speed <i>x</i> milliseconds.
<code>void Mouse_Double_Click(int double=200, int single=100);</code>	Mouse left double click, double click speed <i>double</i> milliseconds, single click speed <i>single</i> milliseconds.
<code>int Mouse_Get_X();</code>	Get the current x coordinate of pointer.
<code>int Mouse_Get_Y();</code>	Get the current y coordinate of pointer.
<code>void Mouse_Displace(int x, int y);</code>	Move mouse pointer by <i>x</i> horizontally, and <i>y</i> vertically.
<code>void Mouse_Move_To(int x, int y);</code>	Move mouse pointer to coordinate (<i>x</i> , <i>y</i>).
<code>void Mouse_Point(int x, int y);</code>	Point mouse pointer at coordinate (<i>x</i> , <i>y</i>).
<code>void Mouse_Drag_To(int x₁, int y₁, int x₂, int y₂);</code>	Mouse drag from coordinate (<i>x</i>₁ , <i>y</i>₁) to (<i>x</i>₂ , <i>y</i>₂).
<code>void Mouse_Drag_Displace(int x₁, int y₁, int x₂, int y₂);</code>	Mouse drag from coordinate (<i>x</i>₁ , <i>y</i>₁) by <i>x</i>₂ horizontally, and <i>y</i>₂ vertically.
<code>void Key_Down(int x);</code>	Press keyboard key <i>x</i> , where <i>x</i> is a virtual key code.
<code>void Key_Up(int x);</code>	Release keyboard key <i>x</i> , where <i>x</i> is a virtual key code.

Appendix A

<i>void Key_In_String(CString string, int x);</i>	Key in a <i>string</i> of characters, with delay <i>x</i> milliseconds between subsequent characters.
<i>int Mouse_Click_Item(CString button, CString caption, CString classname, int width, int height);</i>	Mouse button <i>button</i> to click at item with caption <i>caption</i> , class name <i>classname</i> , width <i>width</i> , and height <i>height</i> .
<i>int Get_Item_Location(POINT *upperleft, POINT *lowerright; POINT *center; CString caption, CString classname, int width, int height);</i>	Get the coordinates of <i>center</i> , <i>upperleft</i> , and <i>lowerright</i> corners of item with caption <i>caption</i> , class name <i>classname</i> , width <i>width</i> , and height <i>height</i> .

Table A.1 AMM's Toolbox Functions

<pre> <zs:element name="mouse_drag"> <zs:complexType> <zs:sequence> <zs:element name="x_axis_movement" type="xs:integer" /> <zs:element name="y_axis_movement" type="xs:integer" /> <zs:element name="window_caption" type="xs:string" /> <zs:element name="class_name" type="xs:string" /> <zs:element name="width" type="xs:integer" /> <zs:element name="height" type="xs:integer" /> <zs:element name="window_style" type="xs:string" /> </zs:sequence> </zs:complexType> </zs:element> </pre>		
Action: <mouse_drag>		
Tools	Parameters	Value
int Get_Item_Location(<i>POINT *upperleft, POINT *lowerright; POINT *center; CString caption, CString classname, int width, int height</i>)	POINT *upperleft	
	POINT *lowerright	
	POINT *center	
	CString caption	<window_caption>
	CString classname	<class_name>
	int width	<width>
	int height	<height>
void Mouse_Drag_Displace(<i>int x1, int y1, int x2, int y2</i>);	int x1	center -> x
	int y1	center -> y
	int x2	<x_axis_movement>
	int y2	<y_axis_movement>

Table A.2 Link between <mouse_drag> action and its tools

```

#ifndef AFX_MYCOMMOBJECT_H_A4C9BBF7_8CBE_4E8C_9C99_36ED6A21E8BA__INCLUDED_
#define AFX_MYCOMMOBJECT_H_A4C9BBF7_8CBE_4E8C_9C99_36ED6A21E8BA__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MyCommObject.h : header file

////////////////////

// MyCommObject command target

class MyCommObject : public CAsyncSocket
{
// Attributes
public:

// Operations
public:
    MyCommObject();
    virtual ~MyCommObject();

// Implementation
protected:
    int ConnectToAgent(CWnd* parentAgent, UINT targetPort, LPCTSTR targetAddress);
    //parentAgent == pointer to current agent

    CString RegisterAgent();
    int UnRegisterAgent();
    void SetAgentID(CString);
    CString GetAgentID(CString);
    int SendMsg(CString message);
    int IAmReady();
    int RecordLog(CString logEntry, CTime eventTime);
    CString RetrieveLog(CTime eventTime);
    int CloseComm();

    //overrides callback function of CAsyncSocket
    virtual void OnSend(int nErrorCode);
    virtual void OnReceive(int nErrorCode);
    virtual void OnClose(int nErrorCode);
    virtual void OnConnect(int nErrorCode);
    virtual void OnAccept(int nErrorCode);

private:
    CWnd* pParentAgent;
    CString agentID;
};

#endif
// !defined(AFX_MYCOMMOBJECT_H_A4C9BBF7_8CBE_4E8C_9C99_36ED6A21E8BA__INCLUDED_)

```

Figure A.2 Header file for *MyCommObject* class


```

#ifndef AFX_MYSCHEDULER_H_D0AA1B29_A774_4683_88B8_75760225933B__INCLUDED_
#define AFX_MYSCHEDULER_H_D0AA1B29_A774_4683_88B8_75760225933B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class MyScheduler
{
public:
    MyScheduler();
    virtual ~MyScheduler();

protected:
    void SetParent(CWnd* pParentAgent);
    int StartTimer(CString timerName, UINT timeout, UINT repetition = 0);
    int TerminateTimer(CString timerName);
    int TerminateAllTimer();
    int SuspendTimer(CString timerName);
    int ResumeTimer(CString timerName);
    int SuspendAllTimer();
    int ResumeAllTimer();
    int ExtendTimer(CString timerName, UINT extensionTime);
    int ShortenTimer(CString timerName, UINT shortenTime);
    int RecallTimeout(CString timerName);
    int RecallTimeLeft(CString timerName);
    int TimerIsSuspended(CString timerName);
    int RecallRepetition(CString timerName);
    int StartAlarm(CString alarmName, CTime alarmTime);
    int TerminateAlarm(CString alarmName);
    int TerminateAllAlarm();
    int SuspendAlarm(CString alarmName);
    int ResumeAlarm(CString alarmName);
    int SuspendAllAlarm();
    int ResumeAllAlarm();
    int RecallAlarmTime(CString alarmName);
    int AlarmIsSuspended(CString alarmName);
    CString* GetAllTimerName();
    CString* GetAllAlarmName();

private:
    CWnd* pParentAgent;
    CTime timeT;
    struct{
        int timerID;
        CString timerName;
        int timeout;
        CTime startTime;
        int suspend;
        int repetition;
    }TimerItem[MAX];

    struct{
        int alarmID;
        CString alarmName;
        CTime alarmTime;
        CTime startTime;
        int suspend;
    }AlarmItem[MAX];
};

#endif // !defined(AFX_MYSCHEDULER_H_D0AA1B29_A774_4683_88B8_75760225933B__INCLUDED_)

```

Figure A.3 Header file for *MyScheduler* class

Appendix A

Figure A.4 UML Class view of Agent Shell

Figure A.5 UML Class view of Authentication Agent

Appendix A

Figure A.6 UML Class view of Service Directory Agent

Figure A.7 UML Class view of FTP Agent

Appendix A

Figure A.8 UML Class view of Knowledge Base Agent

Figure A.9 UML Class view of Technician Agent

Appendix A

Figure A.10 UML Class view of Coordination Agent

Figure A.11 UML Class view of Usher Agent