

Developing Efficient Mobile Agent Systems using Aspects

by

Yang Max Wang

Supervised by

Prof. John Grundy

Prof. John Hosking

Dr. Santokh Singh

A Master of Science Thesis in Computer Science

The University of Auckland

Abstract

Mobile agents are emerging as a complementary technology for supporting distributed computing and remote services that grant the agents both automaticity and mobility. In this thesis, we introduce a novel aspect oriented approach called the Aspect Oriented Mobile Agent system (AOMA) based on the development and reengineering of an exemplar mobile agent system in the form of a meeting scheduling tool. Our approach is adapted from our component-based development methodology Aspect-Oriented Component Engineering (AOCE) that uses aspects to better categorize and reason about provided and required services of individual components in software systems. Our approach uses new extended Aspect-Oriented UML notations to effectively and efficiently describe, capture and manage both early aspects and late aspects with the support of AOCE.

Acknowledgement

I would like to thank my supervisors Prof. John Grundy, Prof. John Hosking and Dr Santokh Singh much for their invaluable suggestions and guidance throughout the thesis research. I also appreciate their support and encouragement during my hard times in the year. I also thank my parents for their encouragement, support and great family love. Also, I would like to thank my relatives and college friends who have either directly or indirectly helped me out during the year.

Table of Contents

| | | |
|-------------------------|--|----|
| Abstract | I | |
| Acknowledgement | II | |
| Table of Contents | III | |
| Chapter 1 | Introduction | 1 |
| 1.1 | Motivation | 2 |
| 1.2 | Objectives | 2 |
| 1.3 | Our Approach | 3 |
| 1.4 | Thesis Overview | 4 |
| Chapter 2 | Research Background and Related Work | 5 |
| 2.1 | Distributed Computing Techniques | 5 |
| 2.1.1 | Remote Objects | 5 |
| 2.1.2 | CORBA | 7 |
| 2.1.3 | Web Services | 8 |
| 2.2 | Mobile Agents | 9 |
| 2.2.1 | Aglet | 11 |
| 2.2.2 | Jade | 11 |
| 2.2.3 | Ether Yatri | 12 |
| 2.2.4 | Agent Development Kit (ADK) | 13 |
| 2.3 | Software Development Lifecycle | 13 |
| 2.4 | Unified Modeling Language (UML) | 14 |
| 2.5 | Aspect Oriented Component Engineering | 15 |
| 2.5.1 | AOUML | 16 |
| 2.5.2 | Developing Web Services with AOCE | 17 |
| 2.6 | Aspect Oriented Programming | 17 |
| 2.6.1 | Aspect J | 18 |
| 2.6.2 | Spring | 18 |
| 2.6.3 | PostSharp | 19 |
| 2.6.4 | Aspect# | 20 |
| 2.6.5 | AspectDNG | 20 |
| 2.7 | Summary | 21 |
| Chapter 3 | Development of Meeting Scheduler without Aspects | 22 |
| 3.1 | Requirements | 22 |
| 3.2 | Requirement Engineering | 23 |
| 3.2.1 | Identification and Analysis of Stack Holders | 23 |
| 3.2.2 | System Requirement Analysis | 24 |
| 3.2.3 | Service Role Analysis | 25 |
| 3.3 | Architectural Design | 27 |
| 3.3.1 | Use Case Analysis with Sequence Diagrams | 27 |
| 3.3.2 | Identification of Task Agents | 31 |
| 3.4 | System Design | 33 |
| 3.5 | System Development | 36 |
| 3.6 | Summary | 39 |
| Chapter 4 | Analysis of Meeting Scheduler and AOMA | 40 |
| 4.1 | Cross-Cutting Requirement Analysis | 40 |
| 4.2 | Architecture Design Analysis | 43 |
| 4.3 | Domain Analysis | 44 |
| 4.3.1 | Mobile Agent Systems | 45 |
| 4.3.2 | Hybrid Web Application | 45 |
| 4.3.3 | Security | 46 |
| 4.4 | AOMA Scheduling Analysis | 47 |
| 4.4.1 | Design Pattern | 48 |
| 4.4.2 | Implementation | 49 |
| 4.4.3 | AOMAS Aspects | 50 |

| | | |
|------------|---|----|
| 4.5 | Summary | 51 |
| Chapter 5 | Re-engineering Meeting Scheduler with Aspects | 52 |
| 5.1 | AO-Requirement Engineering..... | 52 |
| 5.1.1 | Delegate Components | 52 |
| 5.1.2 | Service Components..... | 54 |
| 5.2 | AO Architecture Design | 55 |
| 5.3 | AO System Design..... | 57 |
| 5.4 | Implementation | 59 |
| 5.4.1 | Basic Aspects | 60 |
| 5.4.2 | Composite Aspects..... | 63 |
| 5.4.3 | Security | 65 |
| 5.4.4 | Planning..... | 66 |
| 5.5 | Summary | 69 |
| Chapter 6 | Evaluation of the AOMA System..... | 70 |
| 6.1 | Evaluation Criteria | 70 |
| 6.2 | Evaluation Scenarios | 71 |
| 6.2.1 | Scenario One | 71 |
| 6.2.2 | Scenario Two..... | 74 |
| 6.2.3 | Scenario Three..... | 77 |
| 6.2.4 | Scenario Four | 78 |
| 6.2.5 | Scenario Five..... | 79 |
| 6.3 | Evaluation Result | 83 |
| 6.4 | Summary | 85 |
| Chapter 7 | Discussion and Conclusion | 86 |
| 7.1 | Contributions..... | 86 |
| 7.2 | Future Research..... | 87 |
| 7.3 | Conclusion | 87 |
| References | | 89 |
| Appendix | | 92 |
| 9.1 | The Extended AO-XML Schema | 92 |
| 9.2 | MASDL..... | 95 |

Chapter 1 Introduction

With the emergence of large scale systems, the traditional approaches that build software from scratch have been replaced by newer approaches called “Component-Based Software Development” methodologies (CBSD). These focus on assembling software systems from parts or “components” [1][2]. CBSD significantly changes the way we develop software systems by focusing on selection of software Components Off-The-Shelf (COTS, also referred to as “commercial off the shelf”) and the assembly of those components within an appropriate software architecture [2][3]. Furthermore, development of large scale component-based systems always involves a variety of distributed technologies, which connect different parts or components over networks and make them seamlessly work together [9]. Widely-used approaches for distributed computing include web services [15], COBRA [16], remote objects [18][19] and mobile agents [14]. In contrast to other distributed computing techniques, mobile agents are still a brand new concept. Several mobile agent frameworks and toolkits have been developed or are under development by the Java and .Net communities [14]. Several conceptual modeling methodologies have also been introduced. Most of them are adapted from traditional agent modeling methodologies, and their major concern is to model the mobility of mobile agents [13].

Aspect oriented software development (AOSD) [20][21] has become an important new approach to software engineering. AOSD addresses the problem of overlapping, horizontal cross-cutting concerns across multiple modules such as classes and components that exist in software development. The idea of this approach is to use aspects to represent concerns that cut across modules, and implement those aspects at a programming level separately from the modules. This enables aspects to be easily managed and controlled since they are isolated from the modules, and once defined, modules can be reconfigured by weaving aspect code in.

Up until the last few years, most work using aspects has been limited to the implementation phase of software development, i.e. finding cross-cutting concerns that implementation units have in common and factoring those out as aspects. Many current applications of aspects, such as in Aspect-Oriented Programming (AOP) [22][23], mainly concentrate on the implementation phase of the life cycle. These aspects are actually code blocks that can specify different concerns in modules. Such aspects are known as “*late aspects*”. However, much recent work has tried to generalize the concept and apply it to different phases of the life cycle. A new direction of AOSD is to identify and categorize aspects, called “*early aspects*”, in early phases of the life cycle, and convert them into programming level aspects for modules in the implementation phase. Identifying aspects at an early stage helps to achieve separation of crosscutting concerns in the initial system analysis instead of deferring such decisions to later stages of design and code, hence avoiding costly redesigning and refactoring. Several approaches have been introduced to try to assist in the identification of early aspects for AOSD, these includes Theme [24], Early-AIM [26], and EA-Miner [25]. However, most such techniques are still in initial stages of research. Our whole of lifecycle AOCE methodology [8] is one such technique.

In contrast to traditional software systems, component-based systems offer great potential for better reuse of existing or third party components, compositional system development, and dynamic end user reconfiguration. However, CBSD methodologies also encounter several common problems, similar to those present traditional software systems. These include the common issue of cross-cutting concerns and the interleaving of common code [4]. In CBSD, this problem may also lead to poor or inaccurate descriptions of the assembled component's requirements, and cause difficulties for developers and other components to make design decisions [5][6]. To address these issues, we have developed a novel CBSD approach called Aspect-Oriented Component Engineering (AOCE). Our new approach uses a horizontal modulization unit called an "aspect" to capture, depict, and handle cross-cutting characterizations in various views of components throughout the whole software development lifecycle [7][8]. In later stages, AOCE also promises full support for implementation of aspect-oriented designs with specific AOP frameworks or toolkits. In this thesis, we aim to apply AOCE to a new application domain: mobile agent based systems.

1.1 Motivation

While much emphasis has been placed on the development of mobile agent toolkits and platforms, little attention has focussed on the conceptual modelling of mobile agent applications. Although several conceptual modeling methodologies have been developed, their major concern is to model the mobility of mobile agents [10][11][12][13]. None of them have addressed the issues of cross-cutting interactions or communication between various components within mobile agent systems, which we believe to be crucial for elegant design of mobile agent applications due to the variety of distributed techniques and design patterns involved and the massive agent behaviour interactions engendered.

In our previous work, we investigated the application of AOCE to web service development, and proposed several novel techniques to support aspect oriented web services [27]. We were motivated by the success of this approach to explore the application of AOCE to mobile agents, another emerging distributed application technology, for more adaptable and robust mobile agent systems. We have also developed a visual modeling tool called "Aspect-Oriented UML" (AO-UML) for the AOCE methodology [4]. AO-UML provides comprehensive modeling capabilities for the AOCE techniques to support the whole life cycle development. We are further motivated to explore the possibility of providing tool support for AOCE-based specification of mobile agent system through adaptation of our Aspect-Oriented UML tool so that we have a tool that has a wide-ranging ability of handling the design and development of mobile agent systems based on capturing, considering, implementing and utilizing aspects.

1.2 Objectives

The objectives of this thesis are listed below:

- Create a comprehensive and elegant solution to address issues and problems caused by cross-cutting concerns and support aspect oriented system development in each life cycle

phase of mobile agent specification with AOCE.

- Model and develop an exemplar mobile agent system with traditional object oriented techniques for practical knowledge base of this thesis
- Enhance AOURL with additional functions, notations and view types to support the use of AOCE-based mobile agent specification.
- Reengineer the exemplar system using an AOCE-based approach and implement the aspect oriented design with a COTS AOP toolkit.
- Evaluate the effectiveness of our AOCE-based multi-agent system specification from several perspectives.

1.3 Our Approach

This section discusses our approach to achieve the objectives of this thesis. The process includes four essential steps as explained in the following:

- 1) We require a practical study on mobile agent system development for the sake of analysis and investigation of aspects. Using the EtherYatri mobile agent toolkit [31] and the traditional object-oriented approach, we first develop a typical mobile agent system from scratch. Our primary goal is not for a perfect solution in a specific domain but to develop an exemplar application for comprehensive analysis of mobile agent systems. Thus, the system should be as simple as possible, but cover all essential mobile agent system characteristics on the other hand.
- 2) The next step is to address and investigate the cross-cutting issues appearing during the development of the prototype. In terms of early aspects, we focus our research on problems and issues in requirement engineering, architectural design, domain analysis and system design, and propose an early aspect solution. From this, we develop a methodology for Aspect-Oriented Mobile Agent design (AOMA).
- 3) Afterwards, we re-engineer the original prototype with the techniques developed in the previous step, and then implement the aspect-oriented design with the PostSharp toolkit [43] to formalize a late aspect solution. We also use AOURL to support the redevelopment and enhance its function, notations, capabilities and view-types to suit for AOMA. The late aspect solution emphasises conversion and adaptation of early aspects to implementation aspects as well as the techniques for aspect-oriented programming of the mobile agent system.
- 4) The last step is to evaluate the AOMA approach from several perspectives to prove its feasibility and superiority. Thus, we draft two plans for the evaluation. The first one is to apply several identical design changes to both versions of the exemplar application, and compare several measures of them such as code impacted, difficulty to address issues and error proneness in order to evaluate the adaptability of AOMA applications. Then, in the second

plan, we develop another typical mobile agent system with AOMA to estimate its usability, facility and understandability.

1.4 Thesis Overview

This thesis is composed of seven chapters as the outline shown in the following:

Chapter 1 – Introduction This chapter briefly introduces this thesis and describes the motivation, objectives and our approach for this project, so readers can capture a basic understanding of the entire structure of the thesis.

Chapter 2 – Background and Related Work The technologies and research work involved are amply explained in this chapter. It contains various widely-used techniques and studies carried out in several areas including distributed computing, Mobile Agent System Development (MASD), Aspect-Oriented Software Development (AOSD) and Aspect-Oriented Programming (AOP).

Chapter 3 – Development of Meeting Scheduler without Aspects This chapter describes the process of developing a meeting scheduling system using traditional object-oriented techniques. The techniques and modeling methodologies used in each life cycle phase are explained in details here.

Chapter 4 – Analysis and AOMA In this chapter, we address and analyse the cross-cutting issues in early development phases. We also introduce the AOMA methodology and discuss details about how the problems can be effectively solved with early aspects as the early solution of AOMA. Some of the AOMA techniques are intercepted and developed with our AOURL modeling tool.

Chapter 5 – Reengineering Meeting Scheduler with Aspects For proof of concept, with the AOURL modeling tool and the PostSharp toolkit, the meeting scheduler prototype is reengineered using AOMA. We also discuss the problems and issues encountered during the reengineering process.

Chapter 6 – Evaluation In this chapter, we designed several scenarios to evaluate the AOMA methodology with a set of criteria. Then, we develop another typical mobile agent system with AOMA to its estimate reusability, facility and understandability. We also discuss the evaluation results at the end.

Chapter 7 – Discussion and Conclusions This chapter discuss future work and potential improvements of AOMA, and describe the research contribution of this thesis.

Chapter 2 Research Background and Related Work

This project uses a variety of techniques from the areas of Component-Based Software Development (CBSD), Distributed Computing and Aspect-Oriented Software Development (AOSD). In this chapter, we review the state of the art in each of these areas based on our background study and literature review.

2.1 Distributed Computing Techniques

Distributed computing is a method of computer processing in which different parts of a program run simultaneously on two or more computers that are communicating with each other over a network [32]. Distributed computing techniques refer to the frameworks or toolkits that support distributed computing, and can be categorized as platform independent or platform dependent. Platform-independent techniques, such as web services and CORBA, normally provide unique communication protocols to support cross-platform integration for distributed systems. Conversely, platform-dependent techniques rely on the platforms that they are built upon, and they have their own architectures, communication layers and object models. Such techniques include .Net Remoting (superseded by WCF in .Net 3.0), Java RMI and mobile agents. In this section, we describe the architectures and the rationales of several common distributed computing techniques.

2.1.1 Remote Objects

Remote object techniques enable an object on a virtual machine to invoke methods on an object in another virtual machine. The two most common approaches for remote objects are Remoting for .Net [18] and RMI for Java [19].

.Net Remoting

.Net Remoting is a Microsoft application programming interface for remote object communication. .Net Remoting uses the concept of a common language runtime (CLR) Application Domain (AppDomain) to isolate the executed applications from one another. A process can cross multiple application domains but an application domain can only exist in one process. .Net Remoting is the only way to move objects between application domains. An object is considered local if it is in the same AppDomain as the caller or invoker. If the object is in a different AppDomain, then it is designated as remote even if it resides on the same machine as the caller's.

.NET Remoting Architecture

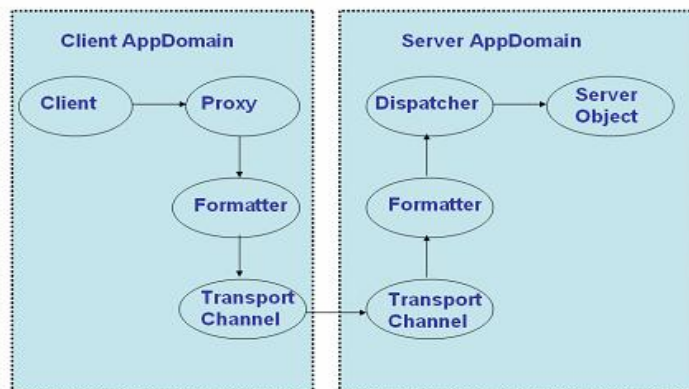


Figure 2.1: .Net Remoting Architecture from <http://www.developer.com>

.Net Remoting enables remote access of objects across application domains using proxy objects. All public methods of a remote object are available to be invoked across AppDomains. As shown in figure 2.1, when a client class calls a remote method of a proxy, the proxy assembles the call with an appropriate formatter and then sends it over an appropriate transport channel. Afterwards, a listening channel on the server domain picks up the message and forwards it to the server system. The server system then dispatches the call with the same formatter and invokes the method of the corresponding object. Finally, the server system returns the results along the same pathway that the call originated from [34].

Java RMI

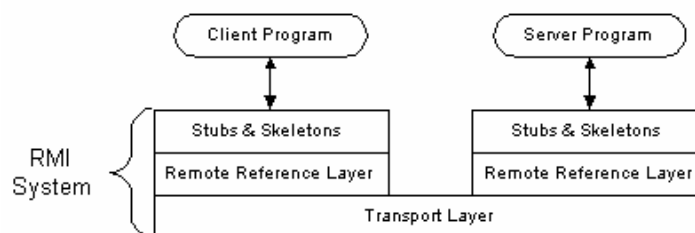


Figure 2.2: Java RMI Architecture from <http://java.sun.com/>

Java's Remote Method Invocation (**Java RMI**) enables programmers to create distributed Java-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines. The RMI architecture is based on one important principle: the definition of behaviour and the implementation of that behaviour are separate concepts. In RMI, the definition of a remote service is coded using a Java interface as a skeleton as shown in figure 2.2. The implementation of the remote service is coded in a class as a stub. A skeleton is a helper class that is generated for RMI to use. It understands how to communicate with the stub across the RMI link.

In JDK2.0, the stub class is superseded by reflection for the connection to the remote service object. The remote reference layer defines and supports the invocation semantics of the RMI connection. This layer provides a RemoteRef object that represents the link to the remote service implementation object. The transport layer makes the connection between JVMs. All network connections are stream-based using TCP/IP [19][35].

Discussion

Remoting and RMI are the two distributed computing technologies for .Net and Java respectively, the two most important computer platforms. They implement remote object communication across an executive boundary which is either a CLR application domain or a JVM. In terms of distributed computing, they are used for development of programs that are required to connect to another applDomain or JVM on a different machine. Although Remoting and RMI have similar concepts, they are platform dependant. This means they cannot interoperate with each other because one platform is unable to interpret the object structure and communication standard of another the other without specific techniques.

2.1.2 CORBA

Common Object Request Broker Architecture (CORBA) is an open, platform-independent architecture and infrastructure defined by the Object Management Group (OMG) for computer applications to work together over networks. CORBA makes cross-platform object invocation feasible.

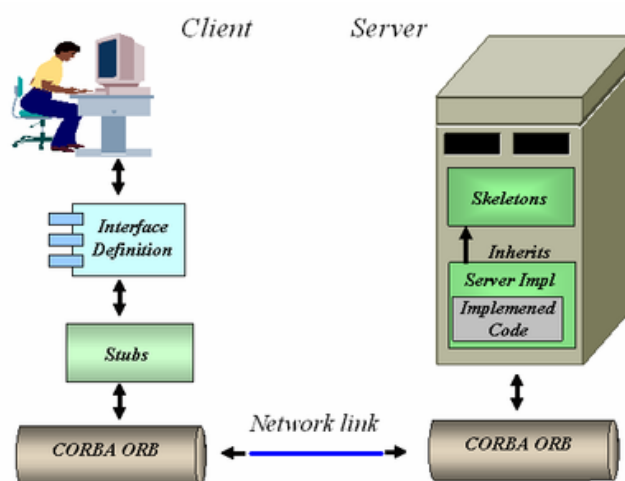


Figure 2.3: The CORBA Architecture from <http://en.wikipedia.org/wiki/CORBA>

Similar to RMI, CORBA uses stubs and skeletons to represent its object behaviours and implementations. CORBA uses an interface definition language (IDL) to interpret the interfaces of program objects (or CORBA objects). CORBA then specifies a “mapping” from IDL to a specific implementation language like C, C++, C#, Perl, Visual Basic, Java and Python. Any program

implemented by the CORBA architecture is capable of interpreting CORBA messages from other CORBA based programs no matter what platforms they run on.

CORBA uses the standard protocol IIOP (Internet Inter-Orb Protocol) as its communication standard. Using the IIOP protocol, programs implemented the CORBA architecture by any platforms from almost any computers can seamlessly interoperate with each other. CORBA has been implemented on both Java and .Net by the communities in the form of RMI-IIOP [33] and IIOP .Net [61].

2.1.3 Web Services

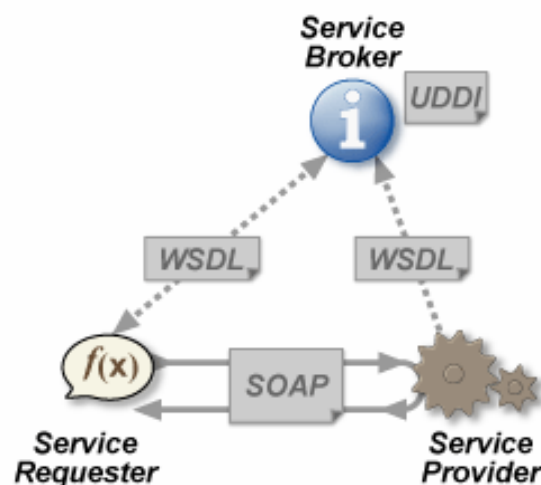


Figure 2.4: The Web Service Architecture from http://en.wikipedia.org/wiki/Web_service

Web services are software systems designed to support interoperable machine to machine interaction over a network. Web services define an application programming interface (API) that can be accessed over a network, and executed on a remote system hosting the requested services [49]. Web services use SOAP-formatted XML documents to exchange information. (see figure 2.4) Interfaces of web services are described by Web Service Description Language (WSDL) which is an XML standard [50]. SOAP is an XML-based envelop format that can be embedded into any web communication protocols such as HTTP, SMTP and XMPP [51]. Web services can be registered and published using UDDI which is a protocol for web services inquiry and discovery [52].

Unlike remote object techniques which are object-oriented, web services are based on the concepts of Service-Oriented Architecture (SOA) that uses a message as the basic unit of communication. SOA is a design for linking business logic and computational resources on demand to achieve the desired results for service consumers such end users or other services. Web services have been widely applied to enterprise system development because of their high usability, facility and platform independence. However, in contrast to remote object techniques, the performance (communication speed) and reliability (fault tolerance) are quite poor in some cases.

In conclusion, both remote objects and web services are powerful technologies for distributed applications. For applications that require interoperability and must function over public networks, web services are probably the best choice. For those that require communications with other remote objects or components and where performance is a key priority, remote objects have superiority. In some architectural scenarios, developers might also be able to use remote objects in conjunction with web services and take advantage of the best of both worlds [34].

2.2 Mobile Agents

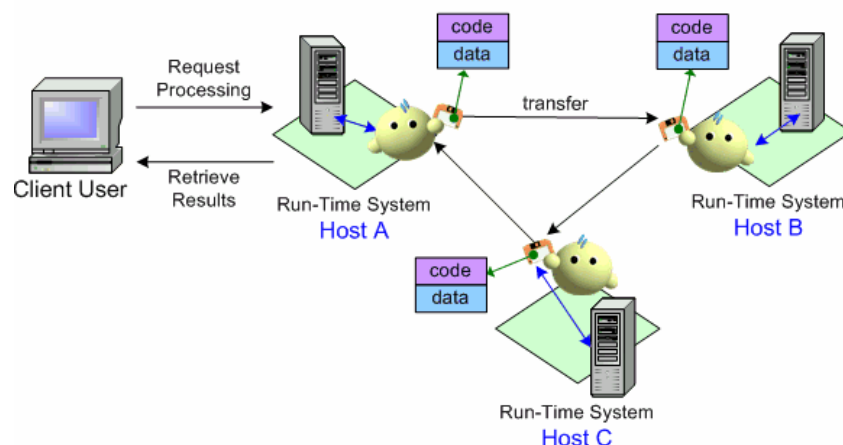


Figure 2.5: A Conceptual Diagram of A Mobile Agent from http://en.wikipedia.org/wiki/Mobile_agent

Mobile agents have been developed as a new technology for distributed application development. Mobile agents are programs that can move from machine to machine. Mobile agent features include their autonomy, social ability and mobility. In contrast to remote object technologies which enable programs to access objects across certain boundaries, mobile agent technology can completely move an entire program from one virtual machine to another. Thus, mobile agent technology is also platform dependant. More importantly, a mobile agent can save its own state and transport this saved state to the next host and resume execution from the saved state. As shown in figure 2.5, mobile agents are autonomous. They can decide their own behaviours such as clone, dispatch, retraction and dispose.

Mobile agents can reduce network load by moving computation closer to data, and support asynchronous operations on heterogeneous hosts, dynamic adaptation, tolerance to network faults and flexible maintenance. The common uses of mobile agents include resource discovery, monitoring, information retrieval, network management, offline autonomous processing, remote socializing and dynamic software deployment [14]. We also note that mobile agent applications do not purely use mobile agents, but also make use of one or more other distributed technologies such as the conjunction of web services and remote object as discussed above. For example, a mobile agent system may use EJB [42] for persistency services, and web services for communication with

other platforms. We term these types of system “Hybrid Web Systems” being web systems developed with a variety of distributed technologies that can interact with each other.

Several mobile agent frameworks and toolkits have been developed or are under development by the Java and .Net communities, including Aglets, Jade and ADK for Java, and Ether Yatri for .Net. Based on our investigation, all of the agent frameworks or toolkits have very similar architectures no matter what platforms they are built upon. Thus, we conclude a common structure from them. This structure consists of three layers the Infrastructure Layer, the Agent Hosting Layer and the Agent Layer as shown in figure 2.6.

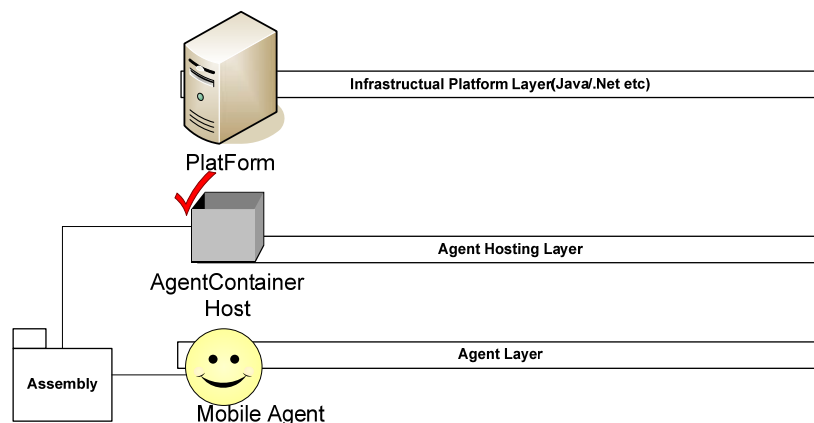


Figure 2.6: The common structure of mobile agent frameworks

Infrastructure Layer represents the development platform such as Java and .Net on which agent frameworks are built upon.

Agent Hosting Layer allows agent hosts to communicate with each other for interoperation. Usually, a host application is needed for every computer to send or receive agents as well as managing them.

```
(request
:sender (:name dominicagent@whitestein.com:8080)
:receiver (:name rexhotel@tcp://hotelrex.com:6600)
:ontology personaltravelassistant
:language FIPASL
:protocol fiparequest
:content
(action movenpickhotel@tcp://movenpick.com:6600
(bookhotel(:arrival 25/11/2000) (:departure 05/12/2000) ...)))
```

Figure 2.7: An ACL Message Example

Agent Layer is the place where agents interact with each other. Those activities dominated by agents themselves should be considered in this layer. Mobile agents cannot be communicated with as common components are. Just as SOAP is used as the communication protocol for web services, mobile agents also have their own communication standards. The most common approach is to use Agent Communication Language (ACL) messages the structure of which is defined by The Foundation for Intelligent Physical Agents (FIPA) [36]. Many mobile agent frameworks and

toolkits such as Jade and ADK support ACL communication. Figure 2.7 shows an ACL message example from Jade's Tutorials [29].

2.2.1 Aglet

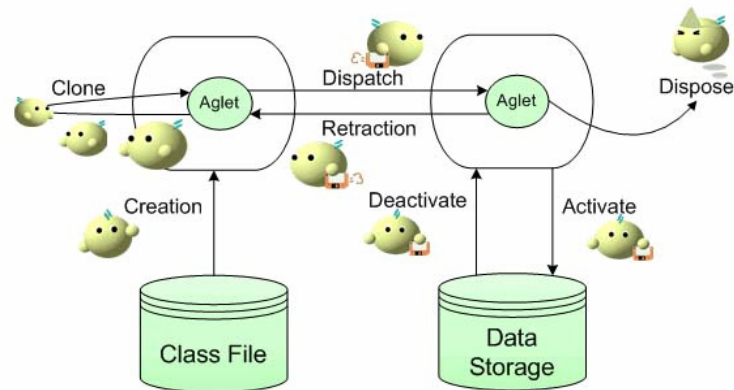


Figure 2.8: The behavior life cycle of an aglet from http://en.wikipedia.org/wiki/Mobile_agent

Aglet is a java based mobile agent platform and library for building mobile agents based applications. It was the first mobile agent development toolkits and was originally developed by IBM Tokyo research laboratory. However, the project is now hosted at SourceForge as an open source project under the IBM Public License [28]. JRE or J2SK are required to run the aglet platform and to start aglet's default server (Tahiti server), which allows Aglets (mobile agents) to freely migrate and communicate with each other. Aglet was appreciated for its clear and easy to use API, good modularity and design, and good security management. However, the Aglet is not stable sometimes when it receives considerable numbers of agents. Garbage collection is another issue of Aglet because of some problems in updating bytecodes of aglets.

2.2.2 Jade

JADE is a Java-Based middleware developed by TILAB for the development of distributed multi-agent applications based on a peer-to-peer communication architecture. Communication between the peers, regardless of whether they are running in the wireless or wired network, is completely symmetric with each peer being able to play both the initiator and the responder role as shown in figure 2.9. JADE includes both the libraries (i.e. the Java classes) required to develop application agents and the run-time environment. Each instance of the JADE run-time is called a "container" [29].

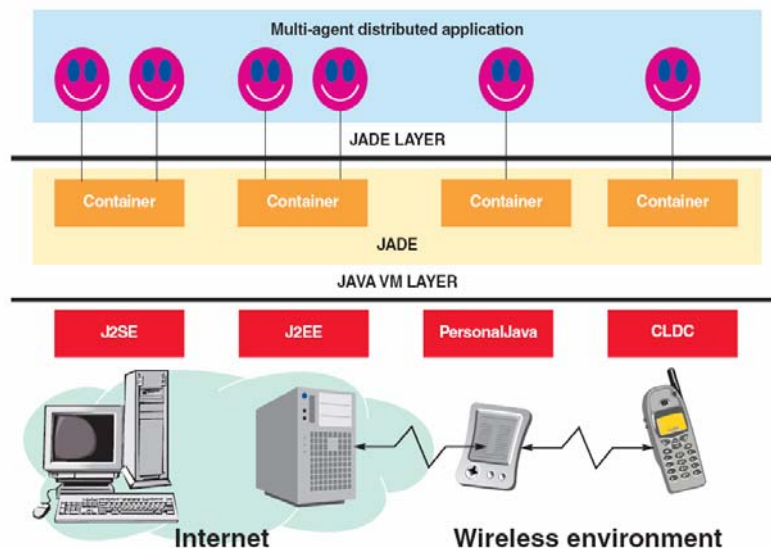


Figure 2.9: The Jade Architecture from Jade's White Paper [29]

Jade is advanced in its support of java mobile environment programming with a technique called Jade-Leap. It simplifies the implementation through a middleware that compiles with the FIPA specifications, which also makes it interpretable with any FIPA based agents such as ADK agents through ACL languages. However, Jade does not support the overloading of methods of operators because its original intention is for development of database applications. One notable problem is the lack of parameterized constructors in Jade, which may cause some serious consequences since it is not possible to tell if an object has been properly initialized [37].

2.2.3 Ether Yatri

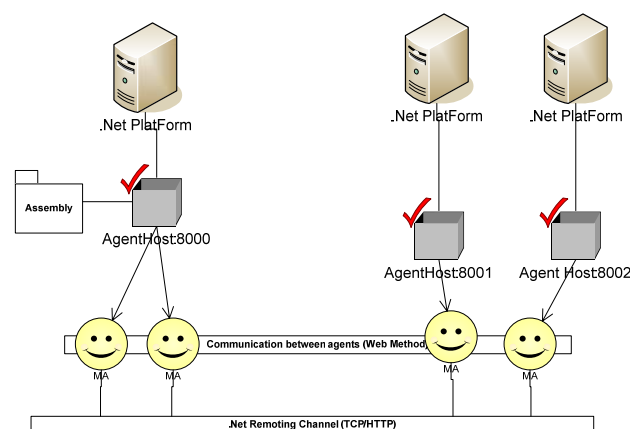


Figure 2.10: The Architecture of EtherYatri

Ether Yatri is a developer toolkit for building mobile agents on the Microsoft .NET Platform. In EtherYatri, an AgentHost is registered on a specific port using .Net's Remoting registration service, so it can be managed as a remote object. The AgentHost manages all kinds of agent activities

including those cannot be done by agents themselves. All agent objects are sent and received by AgentHost through the TCP/HTTP channel as data streams. (See figure 2.10) Unlike Jade and ADK, Ether Yatri uses a different technique called “Agent Web Method” for agent communication. This technique is similar to web services, where the methods of an agent can be marked as web methods which can then be accessed through service calls from agent hosts.

Ether Yatri is the first toolkit that blends the power and versatility of the Microsoft .NET Platform with the flexibility of the mobile agent paradigm. However, EtherYatri doesn't support .Net Compact Framework, which means it doesn't work in a wireless environment. Although agent web methods are easy to use, some issues appear because of the poor reliability of the approach and complexity of the invocation. Another major drawback of EtherYatri is that it does not compile with the FIPA standard, which makes it non-interoperable with other FIPA based agents.

2.2.4 Agent Development Kit (ADK)

The Tyrllian Agent Development Kit (ADK) provides a rich set of functions and standard tasks that allow developers to create mobile agents and to program their behaviour. Agent Hosts in ADK are called “Habitats”. They provide several actions for agents, such as mobility and persistency, and also provide a relatively secure environment.

ADK also compiles with the FIPA standard, so ADK agents can communicate with other FIPA agents through Agent Communication Language (ACL). Furthermore, ADK can work on a range of platforms from IBM zSeries to mobile phone. One of the innovations of the ADK is task-oriented programming. When building an ADK application, the developers can define a series of tasks, which are high level active building blocks that can be combined together in an intuitive and manageable way to create complex logic. However, this approach has fatal drawbacks when processing a large numbers of tasks. For example, if one of the tasks fails, the whole process corrupts. Also, tasks scheduling is very complicated and inefficient through this approach.

2.3 Software Development Lifecycle

The SDLC literature has introduced and defines various phase models [1] [38]. After considering the current state of the art for software engineering, we have identified eight core phases that define an SDLC based on the descriptions in these papers; these eight phases are listed in figure 2.11. For each phase of SDLC, many techniques and models have been developed for object oriented software development. The development of the exemplar application in this thesis work, including both the without aspects stage and the with aspects stage, will use the definition of SDLC form figure 1 as the guidelines. In addition, the first two phases in the table are normally defined as “early phases”, i.e., cross-cutting concerns appearing in those phases are classified as “early aspects”. During design and development of the software, early aspects can be handled programmatically using AOSD techniques and AOP frameworks and toolkits, so the adapted aspects are known as “late aspects” or implementation aspects, or could be handled via conventional coding.

| Phase | Explanation |
|-------------------------|--|
| Requirement Engineering | Requirements engineering is mainly concerned with reasoning about the problem domain and formulating an effective understanding of the stakeholders' needs. Such an understanding leads to the emergence of a requirements specification that forms a bridge between the problem domain and the solution domain which is further divided into system architecture, design and implementation later on. |
| Architecture Design | Software architecture forms one of the key artifacts in the entire software development life cycle since it embodies the earliest design decisions and includes the gross-level components that directly impact the subsequent design and implementation. Accordingly, it is important that the architecture design supports the software qualities required by the various stakeholders. |
| Software Design | The design activity of a software development process gives a designer an opportunity to reason about a required software system as defined by a set of requirements. This process of reasoning about the system entails consideration of the behaviour necessary for the system to achieve its goals, and a corresponding structure to support that behaviour. |
| Software Development | The development phase involves converting design specifications into executable programs. Primary procedural programming activities include the creation and testing of source code and the refinement and finalization of test plans. |
| Testing | This phase requires the completion of various tests to ensure the accuracy of programmed code, the inclusion of expected functionality and the interoperability of applications and other network components. Thorough testing is critical to ensure systems meet organizational and end-user requirements. |
| Maintenance | This phase mainly involves making changes to hardware, software and documentation to support its operational effectiveness. Primary activities include bug fixing, functional enhancement, security enhancement and addressing user requirements. |

Figure 2.11: The important phases of System Development Life Cycle

2.4 Unified Modeling Language (UML)

The **Unified Modeling Language (UML)** [39] is a standard visual language for visualizing,

specifying, constructing, and documenting the artifacts of a software-intensive system. UML provides a variety of visual notations, diagram models to represent, analyze and design software systems. UML can be used to model the structure of object-oriented software and software development processes. Moreover, UML is a platform independent structural language, so it is not restricted by any specific programming language. In the field of software engineering, UML has played an important role for the SDLC model since the release of version 1.0. The UML graph to the SDLC model can best be described by five interlocking views; the use case view, architecture, sequence, design and implementation views. All these view types have been effectively visualized by corresponding UML models including the use case, UML architecture, UML sequence, UML class and deployment diagrams.

2.5 Aspect Oriented Component Engineering

Aspect oriented software development (AOSD) [21] has become an important new concept to software engineering. This approach addresses the problem of overlapping, horizontal cross-cutting concerns across multiple modules. The fundamental idea here is to use aspects to represent concerns that cut modules, and implement aspects at a programming level separately from the modules. Based on the concept of AOSD, we developed the Aspect Oriented Component Engineering (AOCE) approach to support component system development. This approach makes extensive use of both “early” and “late” aspects throughout the software development lifecycle [6][8][27]. AOCE uses component “aspects” to categorise systematic properties that components provide functions for or require functions from other components. Examples of component aspects include user interface, distribution, security and persistency. Properties under a specific aspect category are described as “aspect details” that some components provide or that others require. For example, one component may provide a view (user interface aspect detail, see figure) that another component may require to display data to end users. Each aspect detail can be further described by a set of “aspect detail properties” which store useful information about the aspect detail.

AOCE supports identification, description and reasoning about a component’s high-level functional and non-functional requirements grouped by different types of aspects. AOCE component requirements are refined into design-level software component services implementing these aspects but which are also characterized by more detailed design and implementation-level aspects. Theoretically, Aspect-oriented component designs can be deployed using any component frameworks. Components are implemented using aspect characterizations to support dynamic component description, discovery, adaptation, reconfiguration and deployment-time testing. Aspect information in component implementations allows developers, end users and other components to access high-level knowledge about a component’s capabilities, and to perform basic configuration validity checks. Furthermore, using a proper aspect oriented programming framework, early aspects can be converted into or adapted to implementation aspects. After components are developed, they can be automatically indexed by their early aspects, and users can formulate high-level queries concerning a component’s capabilities.

2.5.1 AO-UML

The AO-UML IDE project is aimed at investigating the capability of the AOCE techniques to support whole lifecycle aspect-oriented component engineering in an efficient manner [4]. For developers who wish to explore AOCE, the AO-UML tool provides a systematic and end-user oriented way for them to learn about AOCE techniques and apply them in their own scenarios. The AO-UML modeling tool is implemented with the Marama meta-CASE tool which was used to specify and generate the AO-UML IDE.

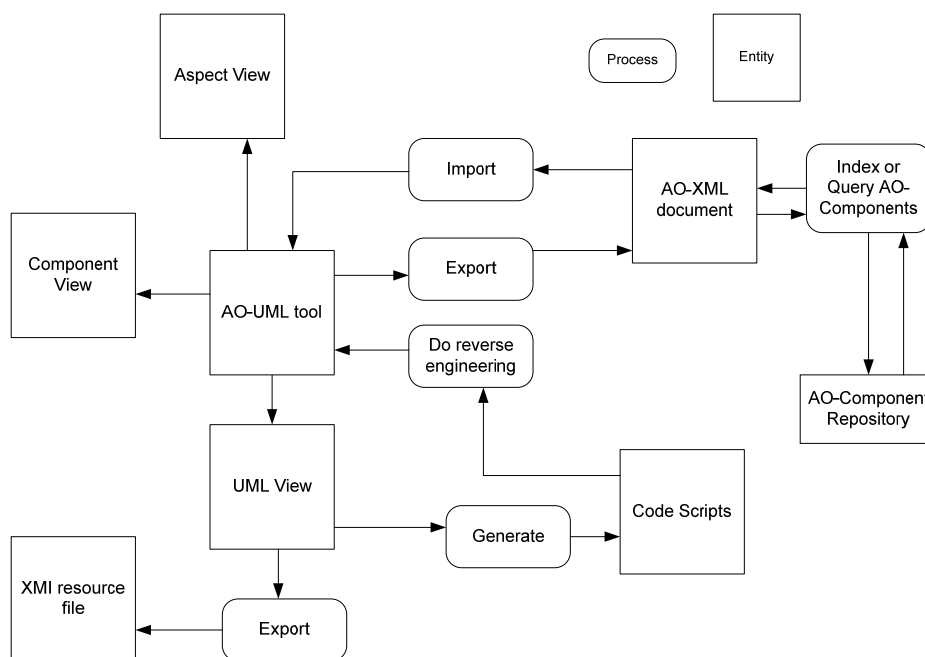


Figure 2.12: The overall architectural design of AO-UML

The architecture of AO-UML, shown in Figure 2.12, consists of three main parts, the Graphical User Interface (GUI), a repository and an implementation factory. The GUI of AO-UML is composed of three view types, which can be switched between: the component, aspect and UML views. Each view type has its own visual symbols and semantics for describing the design in multiple perspectives. The aspect view manages information concerning component aspects and their mapping. The component view is used to gather information about components and their aspects. The UML view co-operates with the component view by importing the aspects from the component views. Developers can use additional notations provided by the UML view to complete the aspect-oriented UML designs of each component.

An aspect-oriented component repository is used by AO-UML users to store and share information concerning all AO-components developed so far. This supports fast search and retrieval of these components using aspect-enhanced queries. In order to communicate with the repository, a novel schema called “AO-XML” was developed to define the grammar for exchanging aspectual information [40]. This schema defines an XML document type which stores information related to the components and their aspects in a well-structured way.

The final part of the architectural design is an implementation factory. This provides tool support for the implementation phase such as transferring the design into a platform-readable form, implementing reverse engineering and exporting XMI files. Based on the UML design diagrams drawn, AO-UML can be used to generate code scripts for various platforms to minimize the developer's effort in writing code.

2.5.2 Developing Web Services with AOCE

We have applied AOCE to the design and development of web services-based distributed systems. Using this approach, developers can analyse, design, develop and maintain web services components based on their functional and non-functional concerns (aspects). Information of these concerns is captured by AO-WSDL documents which are extended from original web service description language (WSDL). Such information can be indexed and inquired by an AO-UDDI registry. This approach supports richer, clearer and more superior web services systems as compared to those built without this technique [27].

2.6 Aspect Oriented Programming

Aspect-Oriented Programming (AOP) complements OOP by providing a horizontal way to analyze and develop programs. AOP aims to increase comprehensibility, adaptability and reusability by introducing a new modular unit called "aspect". Whereas OO presents programs as a vertical hierarchy of objects, AOP decomposes programs into aspects or concerns. This enables the modularization of concerns that would otherwise cut across multiple objects. In this section, we discuss several AOP techniques that are widely used in the community. Before we start, there are several central AOP concepts that must be understood [53]:

- **Aspect** is a modularization of a concern for which the implementation might cut across multiple objects
- **Joinpoint** represents a life point during execution of a program, such as a method invocation or when a particular exception is thrown.
- **Advice** is action taken by the AOP framework at a particular joinpoint. Different types of advice include "around," "before" and "throws" advice.
- **Pointcut** is a set of joinpoints specifying when an advice should fire.
- **Introduction** is a set of methods or fields added to an advised class.
- **Weaving** means assembling aspects to create an advised object. This can be done at compile time (e.g. PostSharp), or at runtime (e.g. AspectJ and Spring).

2.6.1 Aspect J

```

Aspect
    aspect VisitAspect {
        void Point.acceptVisitor(Visitor v) {
            v.visit(this); } }

Pointcuts
    set: pointcut set() : execution(* set*(..)) && this(Point);

Advice
    after () : set() { Display.update(); }

```

Figure 2.13: Example of aspects, pointcuts and advices in AspectJ

AspectJ, the first AOP language, is implemented as an Eclipse plug-in to support aspect-oriented Java Programming. It enables clean modularization of crosscutting concerns, such as error checking and handling, synchronization, context sensitive behaviour, performance optimizations, monitoring and logging, debugging support and multi-object protocols. AspectJ allows all valid Java programs to define special constructions – “aspects” [44]. An AspectJ aspect contains several entities including inter-type declarations, pointcuts and advice as shown in figure 2.13.

AspectJ has become very widely-used and is effectively a de-facto standard for AOP. It is considered the most comprehensive Java AOP framework because of its full support of pre-compilation weaving and runtime weaving. AspectJ has been designed to be implemented in many ways, including source- or bytecode-weaving and directly in the VM. However, on the other hand, the great strength of AspectJ is also its main weakness because AspectJ can obliviously and globally change the behaviour of the base system very easily. Furthermore, AspectJ still have the common problems for AOP such as binding interference, object-type interference and change of inheritance relations.

2.6.2 Spring

Spring is an application framework focused on helping build enterprise applications. It provides a wide range of functionality such as Dependency Injection, Aspect Oriented Programming (AOP), data access abstractions, and remote services integration. Spring has been implemented in both Java (Spring Java) [45] and .Net (Spring.Net) [46] with its core concepts and values.

```

ProxyFactory factory = new ProxyFactory(myBusinessInterfaceImpl);
factory.AddAdvice(myMethodInterceptor);
IBusinessInterface tb = (IBusinessInterface) factory.GetProxy();
tb.CallMethodFromBuissnessInterface();  <-- Advices are executed as well

```

Figure 2.14: An AOP example of Spring.Net

Spring supports aspect oriented programming as part of its business solution. It uses a ProxyFactory class to control ordering and application of the pointcuts and advices. Spring

performs code weaving at run-time as shown in figure 2.14, and a weaved object must have an interface for code injection. The AOP mechanism of Spring is build upon Spring's dependency injection functionalities, which brings both advantages and drawbacks. By using IoC (Inversion of Control) containers, aspects can be weaved to objects at run-time. Implementation of weaving is rather simple because of the spring wrapper. However, the major limitation of Spring is also caused by its proxy-based weaving which requires that a weaved object must inherit a certain interface. Hence, only methods of the interface are eligible to be weaved, which greatly reduces the flexibility of spring's aspect weaving because not all objects are necessary or able to inherit interfaces. The situation is worse when Spring deals with third party components which have all of their constructs fixed.

2.6.3 PostSharp

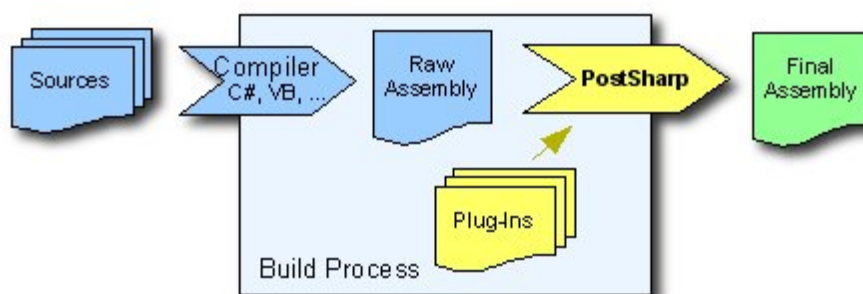


Figure 2.15: The rationale of PostSharp

PostSharp [43] is a platform to transform or analyze .net assemblies after compilation, it reads binary assemblies into memory and interprets them as MSIL codes as described in figure 2.15. Then, it executes a suite of plug-ins and writes back the modified code. PostSharp is still under development as an ongoing project, but it already demonstrates several novel AOP concepts such as scalable multicasting pointcuts using attributes, custom advices with aspectual properties defined in attributes and conditional weaving with aspectual requirements. PostSharp is the most usable C# AOP framework that we have found. It supports capture of global and local pointcut, editing of aspect detail properties and efficient management of aspects. More importantly, PostSharp is compatible with EtherYatri, the mobile agent framework we have used in our research. However, the major drawback of PostSharp is the lacks of runtime weaving support, and common problem of .Net AOP frameworks. Also, the implementation of PostSharp aspects is rather complex and requires extensive experience in CLI or MSIL, the lowest level human-readable programming language in the .Net framework before compilation. As a result, the newer version provides a PostSharp.Laos package which is a lightweight aspect-oriented wrapper to facilitate the weaving in PostSharp.

2.6.4 Aspect#

```

Config File:
aspect SecurityAspect for RSSContentProvider
    include Mixins.SecurityResourceImpl in MyMixinsAssembly
    pointcut method(* RetrieveContent(*))
        advice(SecurityCheckInterceptor) ← Advice is weaved in here.
    end
end

Weaving:
StreamReader reader = new StreamReader( configfile );
AspectEngineBuilder builder = new AspectLanguageEngineBuilder(reader);
AspectEngine engine = builder.Build();
RSSContentProvider provider = engine.Wrap( new RSSContentProvider() );

```

Figure 2.16: A weaving example of Aspect#

Aspect# [48] is a .Net AOP framework based on the common language infrastructure (CLI). It relies on the DynamicProxy, a lightweight proxy generator for interfaces and concrete classes, and offers a built-in language to declare and configure aspects. The weaving approach used in Aspect# is depicted in figure 2.16. A configure file is used to define the weaving details including the packages imported, aspect details, weaved advices and mapping for pointcuts and advices. After a config file is created, Aspect# uses an aspect engine to handle the weaving. The engine injects advices to the target object based on policies defined in that config file, and produce the final output with aspects weaved in. Aspect# has clear definitions for the AOP concepts such as aspect, joinpoint, pointcut and advice, so it is very easy to learn and use. Also, the use of config files makes the weaving process more flexible and reusable. The major drawback of Aspect# is its lack of advice types. There is only one type of advice available in Aspect#, which is the method interceptor advice aka before body advice. Also, weaving with the aspect engine is very inconvenient for developers, and only allows one object to be weaved at a time.

2.6.5 AspectDNG

```

Advice:
[AroundCall("* Sample::Test(*)")]
public static object YourMethodCallInterceptor(JoinPoint jp) {
    Console.WriteLine("Code before calls to '.. Sample.Test(..)");
    object result = jp.Proceed();
    Console.WriteLine("Code after calls to '.. Sample.Test(..)");
    return result;
}

Weaving:
csc /out:mySample.exe /r:aspectdng.exe Sample.cs Aspects.cs
aspectdng.exe mySample.exe
mySample.exe

```

Figure 2.17: Example of advices and weaving in AspectDNG

AspectDNG is a simple free aspect weaver for .Net application. As with other AOP frameworks, it defines a set of advices types for joinpoints such as `AroundBody`, `AroundCall`, and `AroundField`. Other weaving features of AspectDNG include `insert` (insert fields, methods and types), `warning`, `trace`, `error`, `SetBaseType` and `ImplementInterface`(change inheritance relations). Figure 2.17 shows how AspectDNG weaves aspects with an external weaver. The weaving has to be archived manually by using the `aspectdng.exe` weaver to specify the raw class and the aspect class. AspectDNG provides a nice solution for aspect injection because of its clear definition and good variety of advices. However, its weaving strategy is obviously a big limitation for its use, and makes it very inconvenient to use. Moreover, AspectDNG doesn't support aspect customization, meaning defining and editing aspectual properties is not possible.

2.7 Summary

In this chapter, we have summarized and discussed the background technologies related this thesis project including CBSD, Distributed Computing, AOSD and AOP. We also presented several representative techniques in each of these fields such as remote objects, web services and mobile agents for distributed computing, the AOCE approach for AOSD and several toolsets for AOP. In addition, we have described the state of the art of mobile agent technology as well as several widely-used toolkits. In the next chapter, we will describe a practical study of the development of an exemplar mobile agent system by a traditional object oriented approach.

Chapter 3 Development of Meeting Scheduler without Aspects

For the SDLC model, traditional OO techniques supporting analysis and design in each phase have been finely elaborated, and tend to be mature. With these techniques, developers are promised to come out with a system which has been fully decoupled and completely modularised. Based on the previous background study, we have selected techniques which are widely used by the community for the development of our exemplar program. In this project, we mainly concentrate on the first five phases of development and aim to produce a deployable prototype for the program. In this chapter, we discuss the details of the first stage development of the scheduling tool.

3.1 Requirements

The exemplar system for this project is called “Meeting Scheduler”. It is a tool to support a group of users for scheduling meetings on the web. This tool can be plugged into Outlook, which means it can schedule meetings based on users’ Outlook Calendars. Conforming to our fundamental purpose for developing this application, Meeting Scheduler (MS) covers most essential features of mobile agent systems. In addition, it is also designed to involve other distributed techniques such as web services. Therefore, we assume the following requirements in order to achieve the most desirable result for the project.

First of all, users are classified as extra member, group leader, manager or administrator. Login is required for users to enter the system. After users successfully log into the system, they can start synchronizing information with Meeting Scheduler. Synchronized information consists of proposed meetings, invitations, confirmed meetings and contacts. With enough meeting information, including a meeting name, duration, description and participants, a meeting can be initialized by a group leader, manager or administrator. Possible participants are any users who have registered accounts in the system.

Meetings are recorded on the system after being proposed. Participants can see their own invitations when they come online. They can choose to either accept or decline an invitation. Therefore, invitations have three states: pending, accepted and declined. Proposers can always check the states of the invitations. Although the ideal case is to schedule a meeting when all invitees have replied, proposers can always perform scheduling whenever they consider that necessary conditions are fulfilled. To schedule a meeting, the system first works out all possible meeting times with available attendees in each time slot. Furthermore, only participants who accept the invitations are drawn into the scheduling pool. Afterwards, the proposer can choose to either confirm the meeting with the most suitable time or defer making a decision. Each working session is given a 10 minutes time out. If a session expires, the user is required to renew the session.

3.2 Requirement Engineering

3.2.1 Identification and Analysis of Stake Holders

As the starting point of requirement analysis, the stake holders of this system are identified from the requirement specifications:

Meeting Proposer

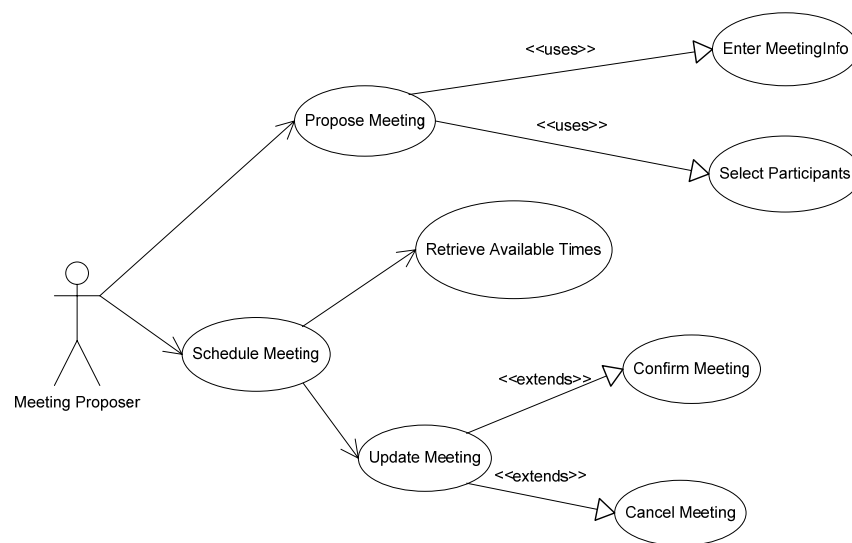


Figure 3.1: The use case diagram of Meeting Proposer

Meeting proposers are the ones who initialize meetings with the system. Only a group leader, manager or administrator can act this role. Meeting proposers are also the ones allowed to schedule the meetings that they proposed. (See Figure 3.1)

Meeting Participant

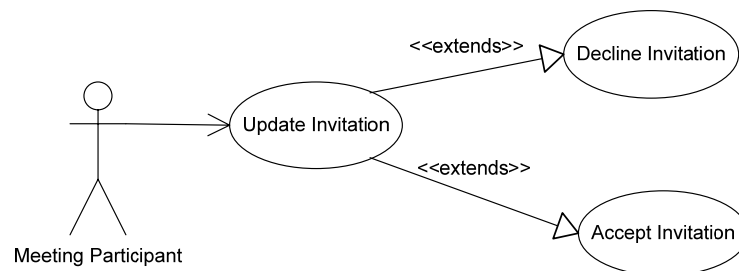


Figure 3.2: The use case diagram of Meeting Participant

A meeting participant is whoever is invited to a specific meeting. They could be participants for several meetings at a time. The only task for participants is to reply to invitations to allow the system to scheduling meetings. (See Figure 3.2)

User

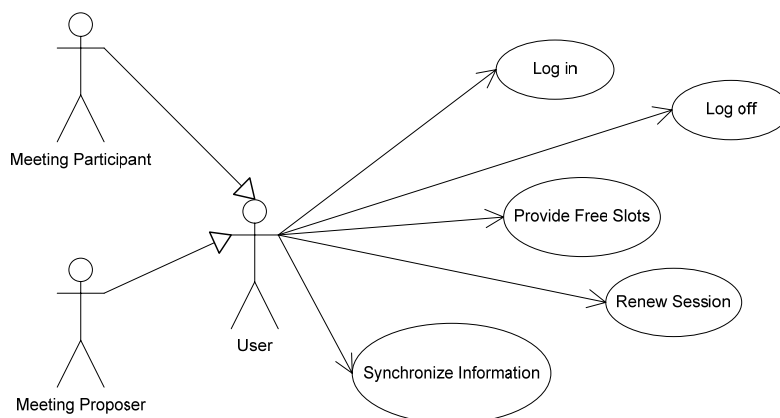


Figure 3.3: The use case diagram of User

User is any person using this system. All users have registered accounts in the system. Moreover, users can synchronize information and update their available time slots for scheduling meetings. (See Figure 3.3)

3.2.2 System Requirement Analysis

Based on the requirements of the stack holders, we can roughly work out what functions the system must provide. (See Figure 3.4) First of all, the system authenticates users with their account information or sessions. The system also takes charge of data persistency by recording information and handling updates from users. For scheduling meetings, it fetches the free slots from Outlook Calendars or directly from users. Based on the times from proposer and participants, it calculates possible meeting slots and records the available attendees in each slot. Managing users' time slots is another concern for the system since such information must be stored off the local client as some kind of data model. Besides that, it is required to provide some GUIs to display the information.

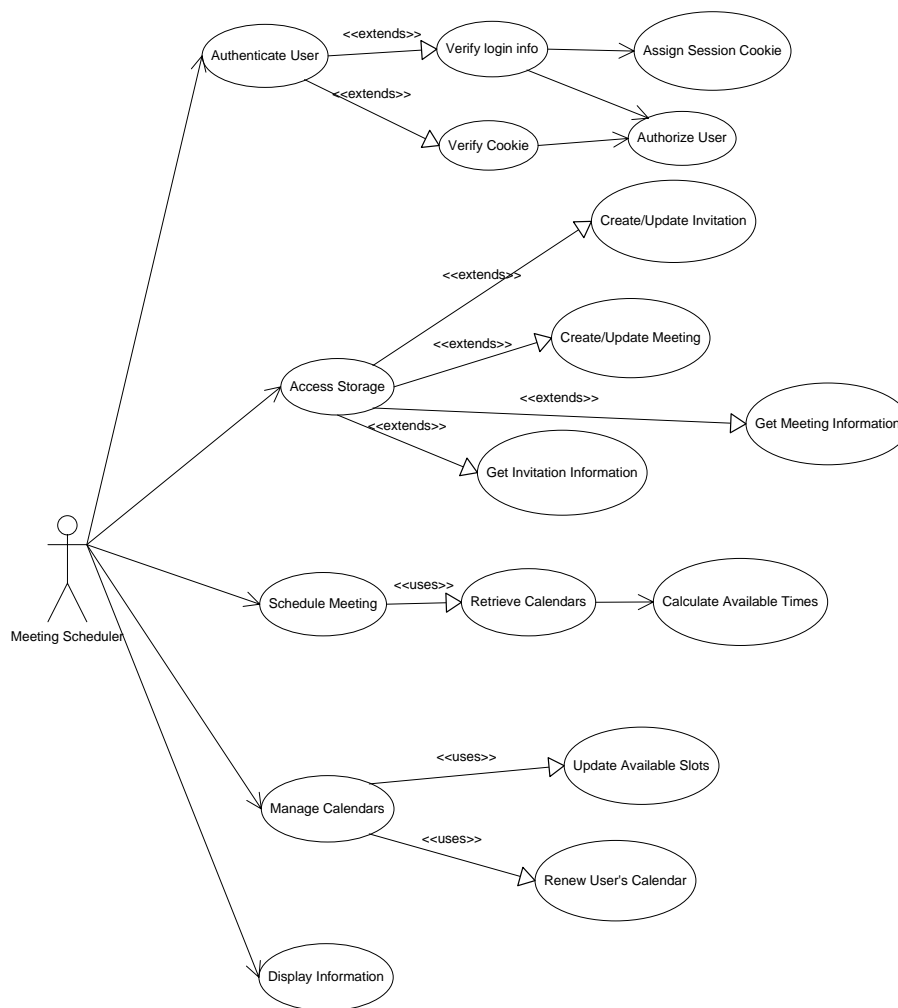


Figure 3.4: The use case diagram of the system

3.2.3 Service Role Analysis

This stage involves activities for identification of the roles of all potential service providers and analysis of their requirements and functionalities. As discussed in the previous chapter, a hybrid web application could use more than one distributed computing technology, meaning that a service provider can be implemented using any of the technologies based on scenarios. For instance, Contact Manager shown in figure 3.6 could be implemented as web service, service agent or EJB. Other than distributed components, the system also consists of traditional components such as those in figure 3.5.

| Traditional Components | |
|------------------------|--|
| <u>Client</u> | This component mainly provides services for users to interact with the system. It also manages various mobile agents. |
| <u>Outlook</u> | This is a data model provided by Outlook for third party softwares to manage its data such as appointments, tasks etc. |

| | |
|----------------------|---|
| <u>DatabaseProxy</u> | This developed component provides all necessary functions for the communication with a SQL server which can be used by any .Net applications. |
|----------------------|---|

Figure 3.5: The traditional components used by the system

| Distributed Components | |
|-------------------------------|---|
| <u>Contact Manager</u> | This role provides various services for contact management such as adding/removing contacts, updating contact information or retrieving contact information, nevertheless this project only requires services for retrieving contact information. |
| <u>Meeting Manager</u> | This role provides services for the management of meetings and invitations. |
| <u>Secretary</u> | Secretary manages its owner's schedule, and negotiates with other users on behalf of its owner. Furthermore, it is quite unlikely to have all users online at the same time, so secretaries are settled on remote hosts to support asynchronized scheduling activities. |
| <u>Securer</u> | Securer agent is in charge of logging, authentication and authorization of the users before they can access any system services. |

Figure 3.6: The distributed components for this system

When candidate components are designed or retrieved from a component repository, we discover that candidate agents can be classified into the following three categories:

- **Static service agents** - these are agents that provide services in their original host, and it is not necessary for them to move around.
- **Mobile service agents** - these agents provide services on behalf of their creators such as common components or other agents. In contrast to static service agents, they are required to migrate to other hosts and provide services there.
- **Task agents** – these are the most commonly used type of mobile agents, they move computation of their creators to remote hosts, and operate there with local data.

Service agents including static service agents and mobile service agents are proxies supporting communications with system services. Therefore, services provided by such agents are actually services available on their hosts. Normally, mobility is not a necessary attribute for service agents. However, some service agents are required to move around and provide services to other users or the system. This kind of service agent is called “mobile service agent. The secretary agent from our scheduling tool is one example because it is created on the client and moves off to provide calendar information to other contacts. Task agents are composed of tasks, destinations and itineraries which can be customized by users. We usually don't consider and identify task agents at this stage since the architecture design is still unknown. In reality, a large scale distributed system might integrate with existing third party services or distributed components such as web services and service agents. Therefore, we leave the decision of identifying and using task agents until the architectural design is worked out.

3.3 Architectural Design

To explore issues of architectural design, as described below, we add several extra architecture requirements for Meeting Scheduler. All services involved in contact management must be provided as web services since these services will also be required by other applications in the future. Another concern is the memory consumption by the agents, especially for those systems with numerous users since users can send their agents to the server such as secretary agents, broker agents and so on. As a solution, we can use some agent habitat servers to host user agents for alleviation of the heavy memory duty of the service host. Furthermore, we choose the EtherYatri toolkit for development of mobile agents since the system is developed in .Net.

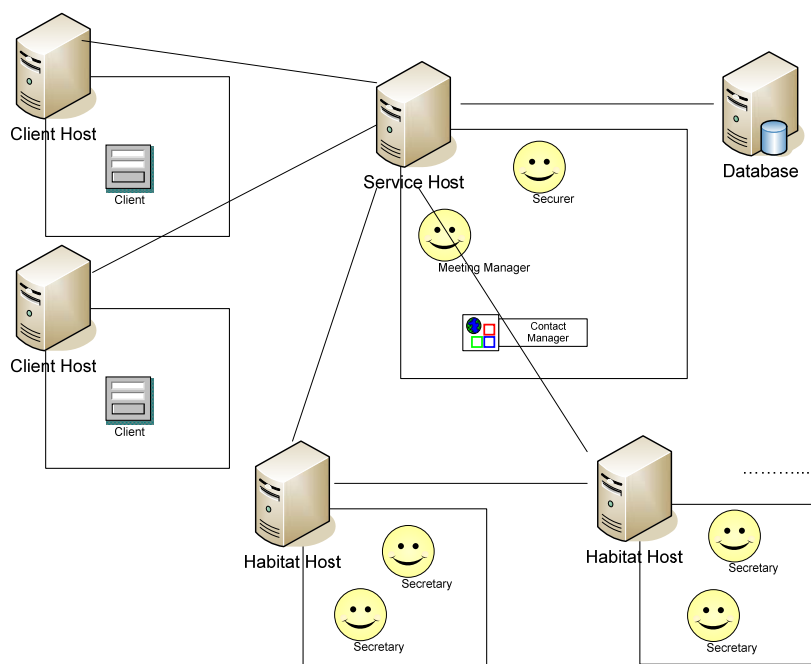


Figure 3.7: The Architectural Design of Meeting Scheduler

The architectural design of Meeting Scheduler consists of three types of hosts including User Host, Service Host and Agent Habitat. (see figure 3.7) A Client Host is the place where users can access the system. It provides GUIs to the users, hosts mobile agents and handles remote invocation of system services such as agent services and web services. The service host is like a system service manager hosting all service providers and it also emerges as the data storage by integrating with a database. A habitat host is used to settle user agents from client hosts. The location information of these agents is managed by the meeting manager from the service host.

3.3.1 Use Case Analysis with Sequence Diagrams

A sequence diagram shows a series of events that occurs within a particular system. A sequence diagram can be extremely useful for design and analysis of task agents, since it visually shows logic flows on vertical time scales [41]. All major events or procedures can be identified according to

the use cases of the stake holders. (See Section 3.2.1) With sequence diagrams, the operation cost of each use case, the main criteria to identify task agents, can be nicely exposed to developers. Initially, assuming task agents do not exist in the system and all remote objects are allowed to be accessed directly, we identify a series of events for each use case with the remote service invocation strategy.

Logging

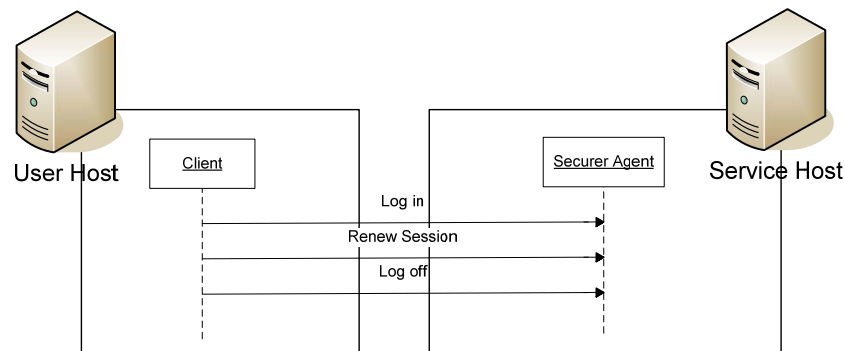


Figure 3.8: The Sequence Diagram of Logging

A user is required to log into the system to get access the system and log off when access is completed. Once users invoke the login service successfully, they receive sessions issued by the securer agent. The session is the key for remote service calls which is used to authenticate the user. Sessions expire after a certain period, and users are required to relog for session renewal. (See Figure 3.8)

Synchronisation

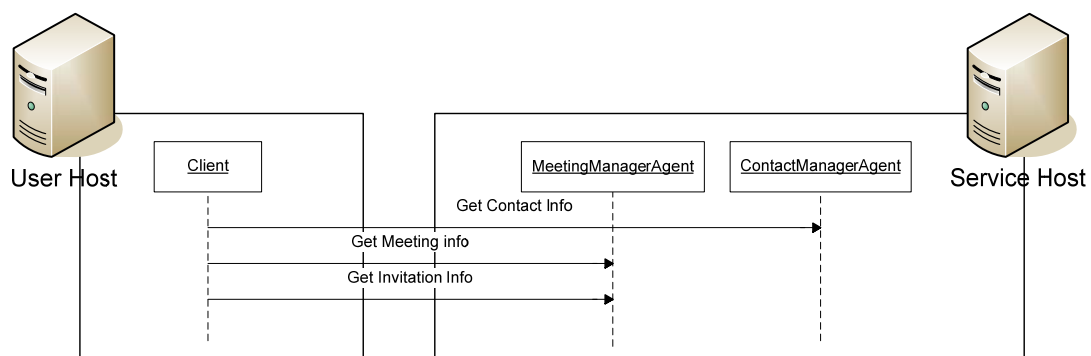


Figure 3.9: The Sequence Diagram of Synchronization

Contacts, meetings and invitations are the three types of information synchronized from the service host. The contact information can be retrieved from the contact manager agent, and meeting

information and invitation information can be found from the meeting manager agent (See Figure 3.9).

Meeting Initialization

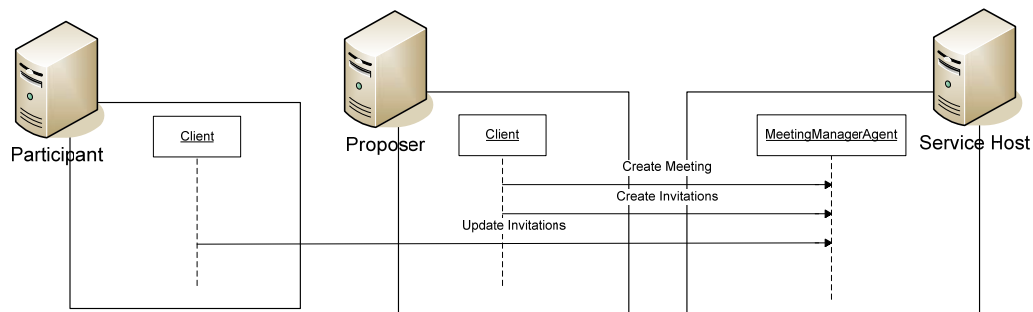


Figure 3.10: The Sequence Diagram of Proposing Meeting and Updating Invitations

To propose a meeting, the proposer can use the client to call the meeting manager and create a meeting instance in the database. Then, based upon participants selected, invitations are also created in the database. Later on, participants can see the invitations and reply to them when they come online (see Figure 3.10).

Migration of Secretary Agent

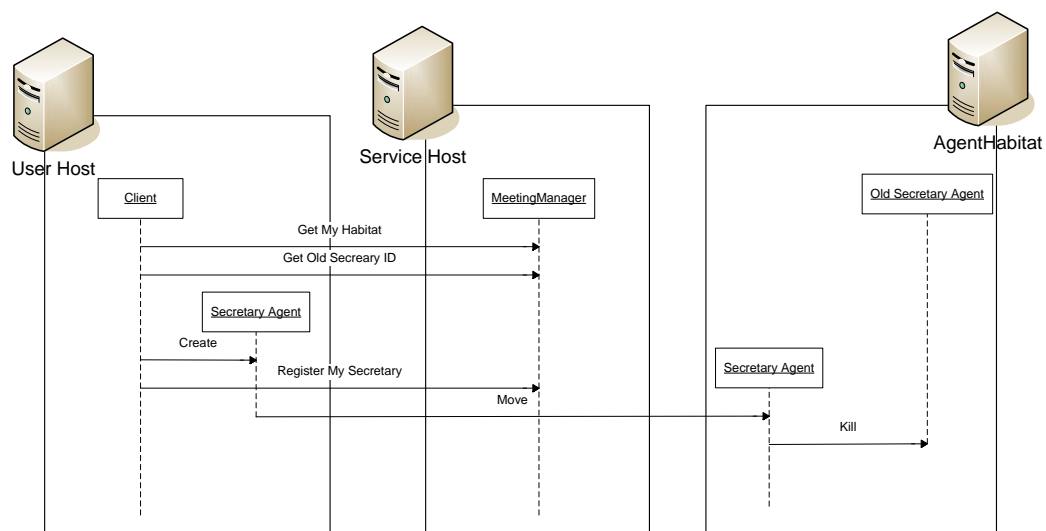


Figure 3.11: The Sequence Diagram of Migrating Secretary Agent

Secretary agents, classified as mobile service agents, are used to manage their owners' schedules and negotiate with other secretaries for scheduling meetings. Before migrating to a habitat, a secretary agent needs to find out the location of the habitat arranged for it. Such information can be fetched from the meeting manager agent. If a secretary agent is already working for the same

owner, its id needs to be found out in order to replace it with the new one. Then, the client creates a new secretary agent, and registers it with the meeting manager. Finally, the new secretary agent migrates to the habitat and replaces the old one. (See Figure 3.11)

Meeting Scheduling

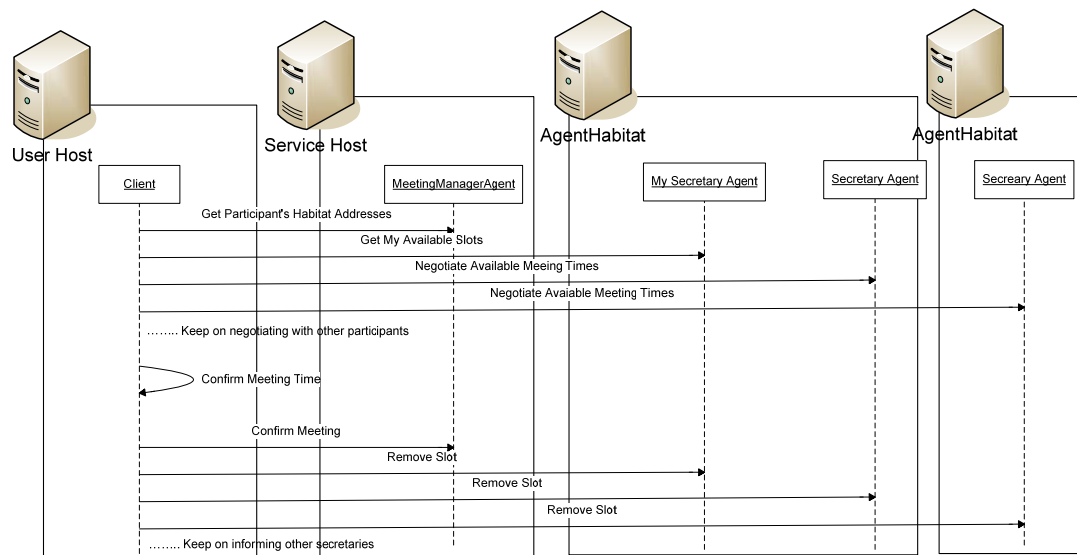


Figure 3.12: The Sequence Diagram of Scheduling Meeting

Scheduling meetings could be complicated if numerous users are invited. First of all, when a user requests to schedule a meeting, the client collects the addresses of the proposer's habitat and the participants' habitats. Then, it queries the proposer's secretary about the time slots available for this meeting. Finally, Based on the proposer's available times, it negotiates with each participant's secretary for its owner's available meeting times. Here is a sample of the negotiation result:

| <u>Date</u> | <u>Time</u> | <u>Attendees</u> |
|----------------------|---------------|-------------------|
| Thursday, 7 December | 10:00 – 10:30 | David, Paul |
| Thursday, 7 December | 12:30 – 13:00 | Paul |
| Friday, 8 December | 11:00 – 11:30 | John, David, Paul |

In this sample, the meeting duration is 30 minutes, and the proposer's secretary provides three available slots for this meeting. The three participants are David, Paul and John. As we can see on the table, the most suitable slot for this meeting is from 11:00 to 11:30 on Friday although the proposer can select any slot from the list. Then, the proposer can confirm the meeting by selecting a preferred time. Then, the client tells the meeting manager that the meeting has been confirmed, and asks it to update this meeting to the database accordingly. Lastly, all secretaries are notified about the confirmation and remove the selected slot from their calendars (see Figure 3.12).

Service Protection

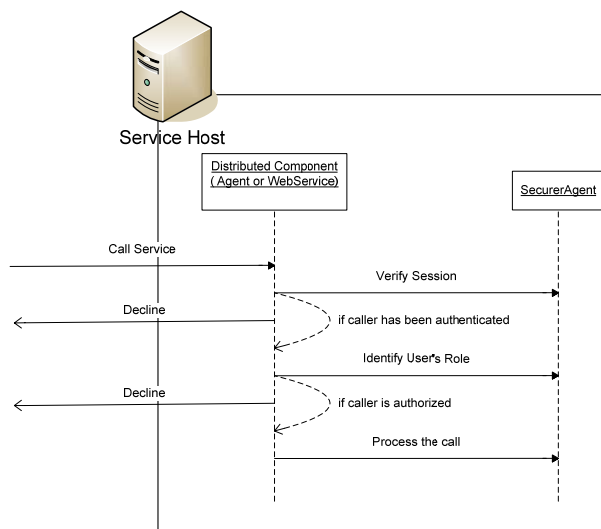


Figure 3.13: A Sequence Diagram for the Security Mechanism of MS

A securer agent is used to handle Meeting Scheduler's security mechanism. We know that the securer agent provides a login service for the system. Other than this function, it also supports authentication and authorization of the system services. When a service from a distributed component is invoked, that component will pass the caller's session to the securer agent before any operation is carried on. If the caller can't be authenticated, the component will decline this request. Afterward, depending on the requirements of this service, the component may require the securer's authorization for this caller. For instance, only roles above group leader are allowed to initialize meetings. To achieve this mission, the component queries the securer agent about the role of the caller. The call can be processed only if the caller's role satisfies the requirements of this service.

3.3.2 Identification of Task Agents

A task agent is delegated to remotely execute part of the computation of a program. Two factors are normally considered when making the decision of using task agents.

- 1) **Transmission cost** is used to measure the amount of data transferred between hosts for certain procedures or user events. Task agents can significantly reduce the transmitting cost if a program needs to process large amount of remote data.
- 2) **Processing power** is referred to as the ability of a computer to process data. Two aspects are considered for processing power. One is the CPU's processing speed of a computer, and the other aspect is the physical and virtual memory for caching data. This factor becomes a major concern on small mobile devices such as PDAs because the processing power of this type of device is usually pretty low. As one solution, a task agent can move the computation of a program to a powerful host, and perform it there.

Negotiator Agent

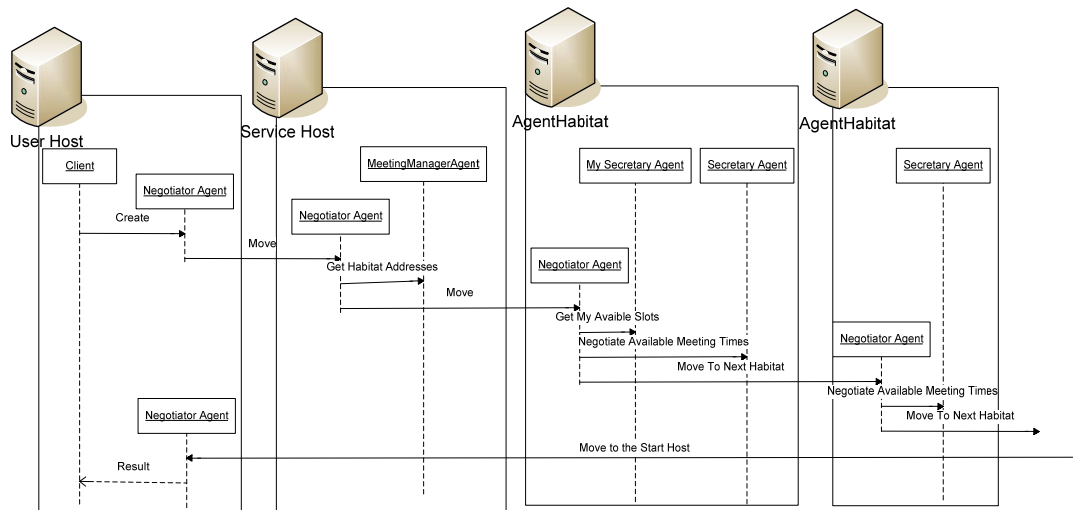


Figure 3.14: The Sequence Diagram of meeting navigation with task agent

Using a negotiator agent can largely cut down the transmitting cost of the procedure for meeting time negotiation. A negotiator agent is created by the client program after receiving the scheduling command. Afterward, it moves to the service host and retrieves the habitat addresses from the meeting manager agent. Based on the habitat addresses, a series of activities is scheduled for the negotiator. The next destination for the negotiator is the habitat host for proposer's secretary agent. On that habitat host, the negotiator agent queries the proposer's secretary about the available time slots that can be used to schedule this meeting. Then, based on possible meeting times of the proposer, it negotiates with each participant's secretary on the same host. As such, it keeps on moving to each habitat on its schedule and negotiates with participants' secretaries. Finally, the negotiator returns to the original host and displays its results to the user (see Figure 3.14).

Organizer Agent

An organizer agent is created to handle a series of tasks after the proposer confirms the meeting. Firstly, it visits the service host to inform the meeting manager about the confirmation, so participants can learn this message when they are online. Then, the organizer moves around habitat hosts and notifies the secretaries of the proposer and participants. Once a secretary gets notified, it removes the corresponding slot from its calendar. Finally, the organizer returns to the original host and commits this procedure to the proposer (see Figure 3.15).

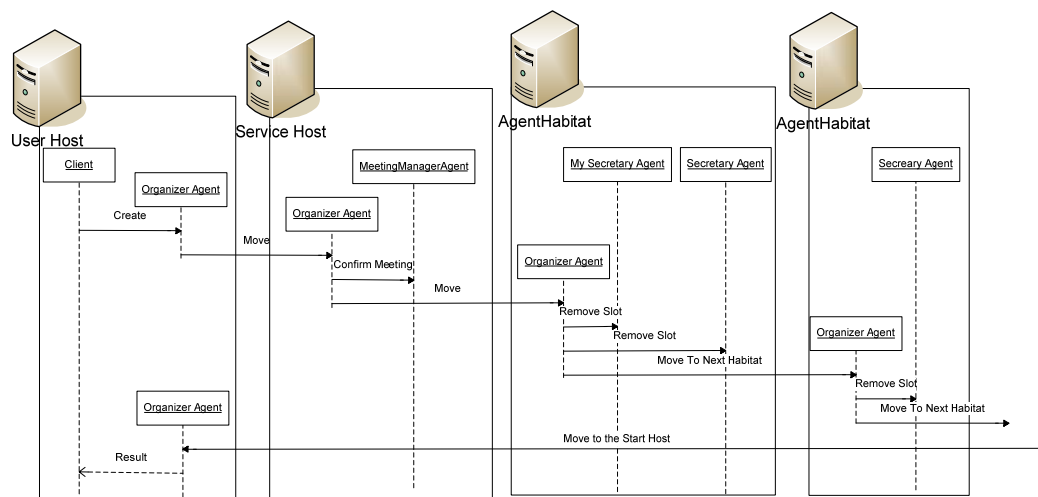


Figure 3.15: The Sequence Diagram of confirming meeting with task agent

3.4 System Design

At this stage, developers can draw UML class diagrams for the static/structural design of the system. In a class diagram, all components are decomposed into parts such as classes, interfaces or structs in order to design the basic structures of those components. The class diagram of a component can be integrated with other components to describe the inner structure of this component. Basically, each component is presented by its constructed classes, connected components and relations between.

Client

The client component (Figure 3.16) provides UIs for users to access the system, communicates with distributed components for various processes and uses a variety of task agents for complex tasks. It can remotely call the contact manager web service, the meeting manager agent and the securer agent. Similar to calling a common web service with the proxy made from WSDL, a pre-made web proxy is included in the client. For communication with the two service agents, the client program obtains their proxies for them from the local host, and then uses agent web method calls to invoke the wanted services. Also, task agent instances are created on the client, and requested to perform predefined tasks. For cooperation with Outlook Calendar, it uses the data accessing services from the Outlook data model component.

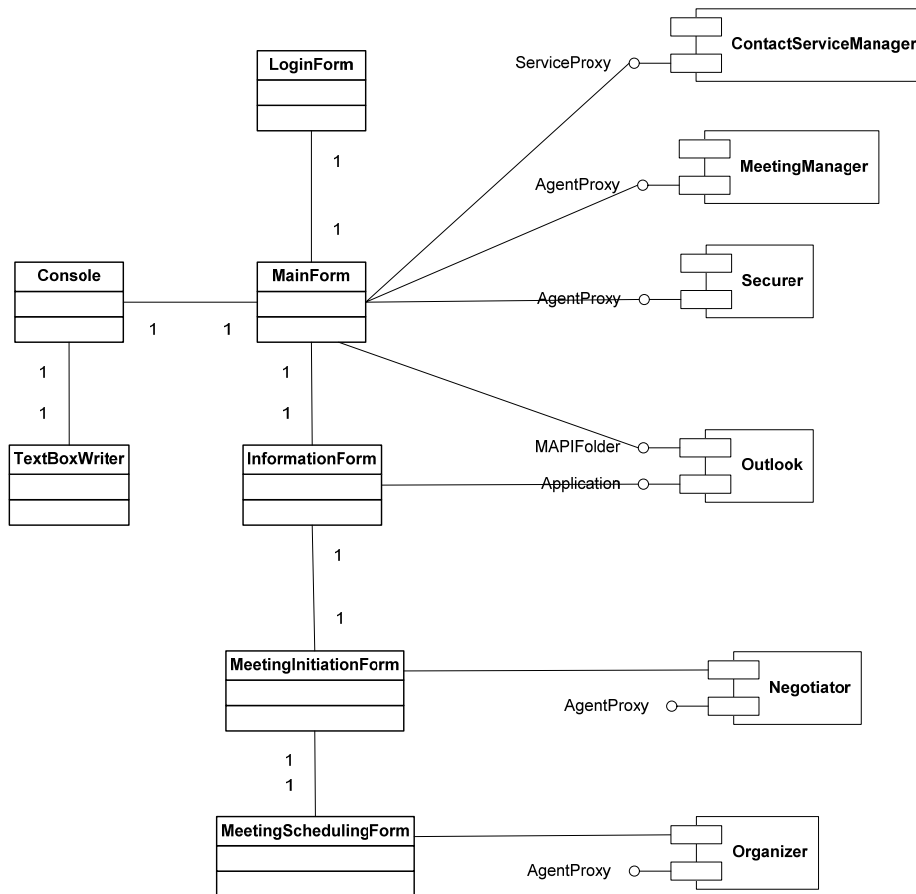


Figure 3.16: A UML class diagram for the client component

Distributed Components

In this system, distributed components consist of static service agents, mobile service agents, task agents and web services. All agents are implemented by EtherYatri, and built as agent components. Furthermore, the service host also contains a web service for accessing contacts information. The following table shows the structural designs of the system components as UML diagrams:

| <i>Web Services</i> | |
|---|--|
| <p><u>ContactServiceManager</u></p> <p>This web service supports the management of contact information with the database proxy component. Just like calling common web services, clients can call this service with its proxy. Moreover, all remote calls to this service have to be verified by the securer agent.</p> | <p>The diagram shows the ContactServiceManager class with methods: <code>+GetMemberList()</code>, <code>+CreateContact()</code>, <code>+RemoteContact()</code>, and <code>+UpdateContactDetail()</code>. It has two dependencies: AgentProxy (to SecurerAgent) and IProxy (to DatabaseProxy).</p> |

| Static Service Agents | |
|---|--|
| <p><u>MeetingManager</u></p> <p>This agent is in charge of all services involved in meeting management such as meeting creation or removal, meeting update, secretary registration and inquiry. As for the contact manager web service, this agent is also protected by the securer agent.</p> | |
| <p><u>Securer</u></p> <p>The securer agent handles a variety of security tasks. One aspect highlighted here is the accessibility of the services. Some algorithms are implemented here to block service calls to VerifySession and IdentifyRole because they are only available to local components for authentication and authorization.</p> | |
| Mobile Service Agents | |
| <p><u>SecretaryAgent</u></p> <p>Secretary agent is the only type of mobile service agent in this system. It is created on a client and transferred to an agent habitat. Then, it manages owner's schedule and assists with meeting scheduling there.</p> | |
| Task Agents | |
| <p><u>NegotiatorAgent</u></p> <p>Negotiators are created on clients, and sent to agent habitats for negotiating available meeting times with participants' secretaries. Then, it delivers the results back to the client. On the way, it also visits the service host to query it about useful information.</p> | |
| <p><u>OrganizerAgent</u></p> <p>Organizer agents are used to notify confirmations of meetings. Their first destination is the services host. There, they interact with the meeting manager and ask for update to the database. Then, it moves to agent habitats and inform all secretaries involved.</p> | |

3.5 System Development

The system is implemented in Microsoft .Net Framework 2.0 with the EtherYatri mobile agent toolkit. This section demonstrates the implementation of the meeting scheduler prototype through a scheduling example. In order to explain this example, we have made the following assumptions:

- 1) Johnh, Johnh, Sans and Maxoo are registered members in this system, their account information has been entered in the database. Each of them has a copy of the client host.
- 2) Port 8000 on local host is used as the service host. Port 8001 and 8002 are used as agent habitats. (See Figure 3.16) In the database, we have configured who shall use which agent habitat – agent habitat 8001 is for johnh and maxoo, and agent habitat 8002 is for johnh and sans.

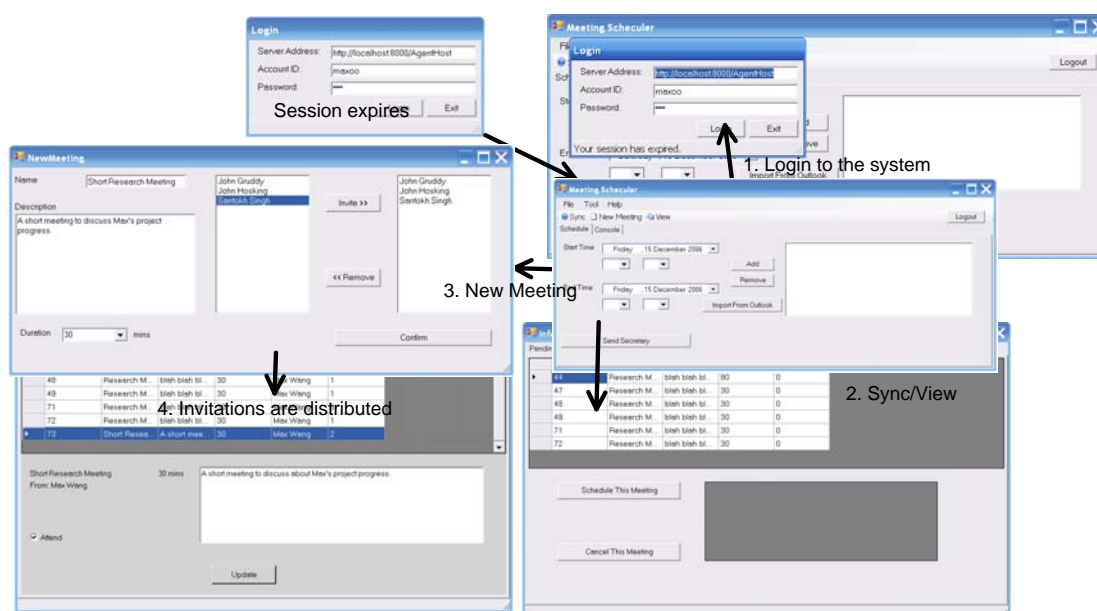


Figure 3.18: Several main panels of Meeting Scheduler

As shown in Figure 3.18, to start the client program, users are required to enter identification information on the login panel and specify the address of the service host. Users can get into the main panel if the log in information provided is correct. Furthermore, if the current session expires during any operation, the login panel will pop up and ask the user to login again. Once users are on the main panel, they can use several of the system’s functions, but before this, they need to click the “Sync” button on the main menu bar to synchronize information from the meeting manager and the contact service manager on the service host. An information form will appear once the synchronization is finished. This form can always be opened by clicking the “View” button on the menu bar. As shown on Figure 3.18, the information panel consists of three tabs which are the pending meeting tab, the invitation tab and the confirmed meeting tab. The pending meeting tab shows all meetings proposed by the user and allows her/him to schedule or cancel meetings. The invitation tab shows all invitations from other users, and the confirmed meeting tab displays all

confirmed meetings. Users can open the meeting initialization form by clicking the “New Meeting” button on the main menu bar. This form collects information of the proposed meeting, and asks the meeting manager on the service host to initialize the meeting. After a meeting is created, all invited participants are able to see the invitation of this meeting when they open the invitation tab.

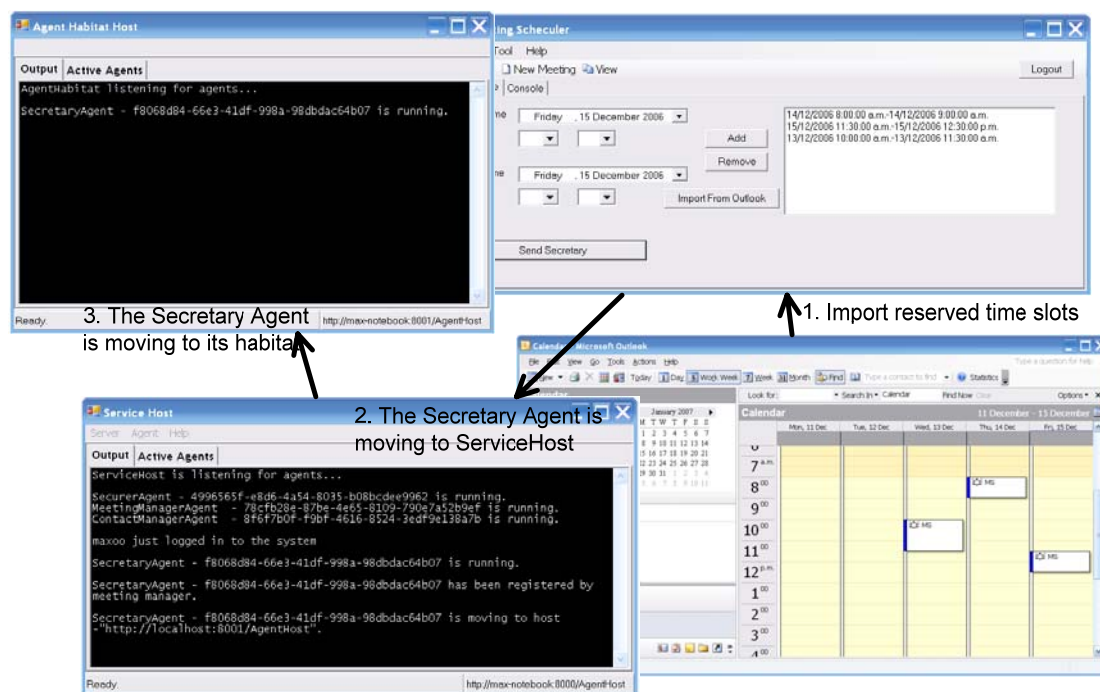


Figure 3.19: The implementation of the secretary section

To actually make the meeting scheduler work, all users have to send out their own secretaries to let others know their current schedules. In our scheduling tool, the way to achieve this mission starts with reserving available meeting slots in the outlook calendar. (See figure 3.19) Afterward, these time slots can be imported into the system using its import function. Alternatively, slots can be entered manually through the main form. Secretary agents are created with these time slots, and sent away to the service host by clicking the “Send Secretary” button. Secretary agents perform two tasks after they get to the service host. Firstly, they request the meeting manager to register them to the system. Then, they query it for the address of its habitat host and move on to that host. Once secretary agents arrive at their habitats, they can start providing scheduling services on behalf of their owners.

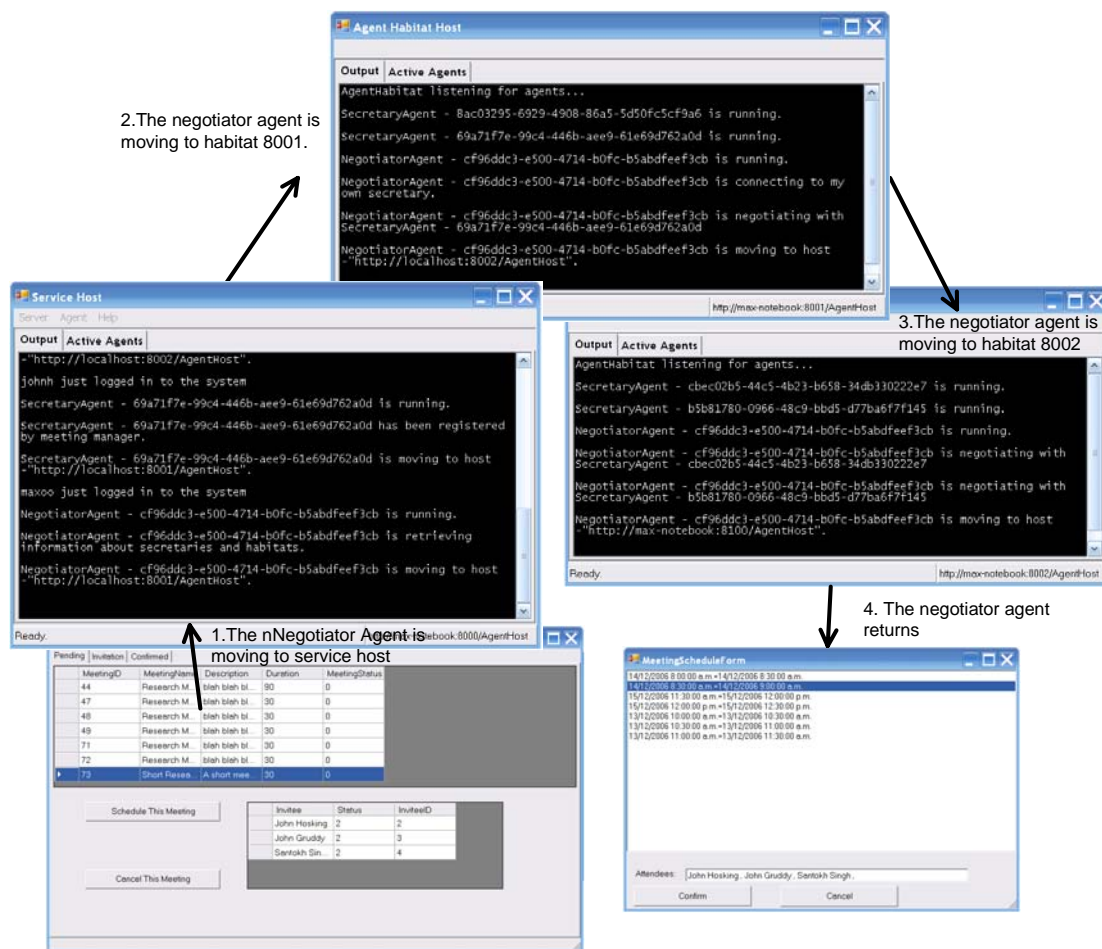


Figure 3.20: The implementation of the negotiation section

Users can check the status of the meetings that they propose on the pending meeting tab. (See Figure 3.20) On this tab, they can see the responses from the invitees. There are three states for an invitation; 0 for no response, 1 for declined and 2 for accepted. Proposers can request to schedule or cancel a meeting at any time. To schedule a meeting, a negotiator agent is created and sent to the service host. Once the negotiator arrives at the service host, it retrieves information about secretaries and habitats. Afterward, it moves to the agent habitat where the proposer's secretary resides and communicates with the proposer's secretary to get the available meeting times for all possible slots of the meeting. Then, if there are any secretary agents of the participants on this habitat, the negotiator agent will negotiate with each of them for their available slots. Through this way, the negotiator goes on to visit all habitats that participants' secretary agents reside and perform negotiation. Finally, it delivers the meeting schedule back to the client. This form shows the negotiation result which presents the available participants in each possible time slot. (See Step 4)

To confirm a meeting, the proposer can select a preferred time slot from the meeting schedule form, and press the "Confirm" button. Then, an organizer agent is created and sent away to the service host. On the service host, the organizer agent confirms the meeting with the meeting manager. Afterward, the organizer agent visits the habitats to notify the secretary agents of the proposer and the participants, and removes the meeting time from their available slots. Once the update is completed, all participants shall see this meeting on the confirmed meeting tab and update their

outlook calendars accordingly.

5. Update Outlook Calendar

1. The organizer agent is moving to service host.

4. The organizer agent returns

3. The organizer agent is moving to habitat 8002.

2. The organizer agent is moving to habitat 8001.

| MeetingID | MeetingName | Description | Duration | StartTime | EndTime |
|-----------|--------------|--------------------------|----------|----------------------|----------------------|
| 43 | Research M. | blah blah blahl | 60 | 9/06/2006 12:00 p.m. | 9/06/2006 1:00 p.m. |
| 50 | Research M. | blah blah blahl | 60 | 9/06/2006 10:00 a.m. | 9/06/2006 11:00 a.m. |
| 73 | Short Reses. | A short meeting to di... | 30 | 14/12/2006 9:30 a.m. | 14/12/2006 9:00 a.m. |

Figure 3.21: The implementation of the organization section

3.6 Summary

In this chapter, we have described the development process of an exemplar mobile agent system as the practical study for this project. In each phase of the development life cycle, common object oriented modelling techniques are used for analysis and design of the system. Meanwhile, we also found some interesting issues and discoveries during the development such as classification of agents, the use of task agents and agent habitats. At the end of this phase, we came out with a prototype of the meeting scheduling system using a conventional object oriented approach. The next chapter will analyse issues arising from this design approach, and how they can be solved through using an aspect oriented approach.

Chapter 4 Analysis of Meeting Scheduler and AOMA

In the previous chapter, we discussed the various processes for developing the meeting scheduler with traditional object oriented methodologies. During the development, we noted that there are a number of design problems and issues related to cross-cuts between components throughout the life cycle. These problems and issues include cross-cutting concerns in requirements analysis, various types of cross-cutting issues appearing in mobile agent systems, inadequate support for architectural analysis and design for mobile agent systems, cross-cutting issues in several domains and the high complexity of mobile agent scheduling. In this chapter, we analyse and describe the issues and problems in each phase. We also introduce the AOMA methodology and discuss details about how the problems can be effectively solved with early aspects as the early solution of AOMA. Some of the AOMA techniques, including delegate aspects, enhanced component view types and the aspect oriented architecture design, are intercepted and developed with our AOURL modeling tool.

4.1 Cross-Cutting Requirement Analysis

Cross-cutting issues are found when candidate components are identified using traditional object-oriented approaches. Similar to traditional component based systems, identification and description of horizontal connections among agents, distributed components and traditional components are important for later implementation and documentation. As part of the AOCE methodology, Aspect-Oriented Component Requirements Engineering (AOCRE) focuses on identifying and specifying the functional and non-functional requirements relating to key early aspects of a system that each component provides or requires services from. Thus, using AOCRE, we can analyze the candidate components, and identify provided and required services.

| Component | Aspect | Aspect Detail |
|-----------------|--------------------|----------------------------------|
| Client | Contact Management | -Contact List |
| | Scheduling | -Create/Remove/Update Meeting |
| | | -Create/Remove/Update Invitation |
| | | -Meeting List |
| | | -Invitation List |
| | -Calendar | |
| | Security | -Logon |
| -Logout | | |
| Secretary | Scheduling | -Register Secretary |
| | | +Update Schedule |
| | | +Schedule Information |
| Organizer | Scheduling | -Secretary Information |
| | | -Schedule Information |
| Negotiator | Scheduling | -Update Schedule |
| | | -Update Meeting |
| Meeting Manager | Scheduling | +Meeting Management |
| | | +Invitation Management |
| | | +Secretary Management |

| | | |
|-----------------|--------------------|-------------------------------|
| | Security | -Authentication |
| | | -Authorization |
| Securer | Persistency | -Transactional Update |
| | Security | +Login/Logout |
| | | +Authentication |
| | | +Authorization |
| | Persistency | -Transactional Update |
| Contact Manager | Contact Management | +Contact List |
| | | +Create/Remove/Update Contact |
| Outlook | Scheduling | +Calendar |
| DatabaseProxy | Persistency | +Transactional Update |

Figure 4.1: Provided and Required aspects of the components of Meeting Scheduler

During the first stage we only focus on the business logic such as meeting scheduling, persistency, and contact management. Figure 4.1 shows the candidate components with their provided and required aspects. In AOCE, aspect details or services provided by components are indicated with the “+” sign, and required services are identified by the “-” sign. For example, as shown in figure 4.1, the security services including login, authentication and authorization are prefixed with a “+” sign as these are provided by the securer component for other component to implement the security mechanism. This component also requires the transactional update service (prefixed with a “-” sign) from the database proxy to query about users’ account information with the database. While we worked on the properties of these aspect details, we also identified several common properties for the aspect details in hybrid web applications. Thus, we consider these properties as common attributes for aspect details of distributed components, and add them to AO-XML, which is an XML schema for exchange of aspectual information [40].

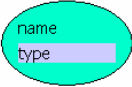
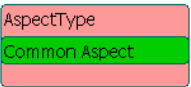
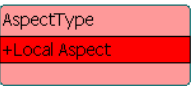
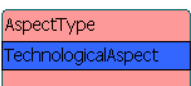

| Visual Symbol | Description |
|---|---|
|  | A software component is a reusable piece of software which has certain functions, and can be integrated with other components. In addition, this icon is extended to describe the component type as well. |
|  | A common provided and required aspect detail is represented by a green cell in an aspect. |
|  | A local aspect detail is represented by a green cell in an aspect. |
|  | A technological detail aspect is represented by a blue cell in an aspect (see Section 4.3) |
|  | A delegate is a relationship in which a mobile agent component is generated by another component to move the computation of its creator to another host. |

Figure 4.2: The Enhanced modeling capabilities of the Component view type

Figure 4.2 describes the new visual elements of the component view type in AOXML’s component view. These include the component, class, aspect and delegate symbols. Each component contains a type attribute now, which is used to specify the component type. Some new features are added with

respect to service invocation. A call type attribute is first used on the aspect tag to specify the types of distributed technologies used on certain services, and component required the services should make calls accordingly. Secondly, three types of aspect details are defined for the sake of distributed components. Other than green cells to represent common provided and required aspect details, red cells and blues cells are used to specify local aspect details and technological aspect details. A local aspect indicates limited accessibility of a provided aspect detail because some services can only be accessed locally by mobile agents instead of being called remotely such as Authentication and Authorization of the securer agent. Furthermore, blue cells represent technological aspect details provided by frameworks or toolkits, and please see section 4.3 for details.

Both the mobile service agent and the task agent are used to act for their creators to provide services or execute computation. Thus, in AORCE, we cannot just explain the relationship between mobile agents and their creators as simple connections because it would cause confusion to the services involved in mobile agents. Instead, we consider a delegate relationship for delegated mobile agents and their creators. This relationship is not only used to indicate the relationship between mobile agents and their creators but also contains other meanings. The aspects provided and required by mobile service agents or task agents are actually delegated from their creators. Thus, their creators also own these aspects while being searched or enquired. A meaningful discovery, we note here, is that ideal candidates of delegate components could be verified according to whether such components purely contain delegate aspects. If the answer is no for a component, it is probably not well designed as a delegate, and requires further elaboration.

```

.....
<xs:attribute name="callType" form="unqualified" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="SOAP"/>
      <xs:enumeration value="AgentWebMethod"/>
      <xs:enumeration value="ACL"/>
      <xs:enumeration value="HTTP"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="remote" form="unqualified" type="xs:boolean" />
.....
<xs:attribute name="componentType" form="unqualified" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Executable"/>
      <xs:enumeration value="Library"/>
      <xs:enumeration value="WebService"/>
      <xs:enumeration value="EJB"/>
      <xs:enumeration value="Agent"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="agentType" form="unqualified" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="StaticServiceAgent"/>
      <xs:enumeration value="MobileServiceAgent"/>
      <xs:enumeration value="TaskAgent"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
.....

```


A Simple Example of the extended AOXML document:

```

<aoxml:application name=" " xmlns:aoxml=" ">
  <aoxml:components>
    <aoxml:component description="" name=" " componentType="">
      <aoxml:aspects>
        <aoxml:aspect name=" ">
          <aoxml:aspect-detail name=" " provided="false"/>
          <aoxml:aspect-detail name=" " provided="true" callType = " " remote=" "/>
          .....
        </aoxml:aspect>
        <aoxml:aspect name=" " >
          .....
        </aoxml:aspects>
      <aoxml:delegates>
        <aoxml:component description="" name=" " componentType=" " agentType=" ">
          <aoxml:aspects>
            .....
          </aoxml:aspects>
        </aoxml:delegates>
      </aoxml:component>
    </aoxml:components>
  </aoxml:application>

```

Figure 4.3: A snippet of the extended AO-XML schema

To be compatible with the enhanced component view, the AO-XML schema is extended with several new attributes as shown in figure 4.3. (Please see Appendix 9.1 for the full schema of the extended AO-XML schema) Each component tag contains a type attribute, which is used to specify the component type. If a component is an agent, then the agentType attribute is required as well. A calling type attribute is used on the aspect tag to specify the types of distributed technologies used on certain services, and component required the services should make calls accordingly. Furthermore, available calling types for various components can vary. For example, among the calling types declared in figure 4.3, SOAP and HTTP are connection types for web services, while AgentWebMethod and ACL are communication techniques for mobile agents. Using this attribute could help developers look up appropriate service providers with proper calling approaches. A remote attribute is then added to declare the accessibility of provided aspects because some services are required to access locally by mobile agents instead of being called remotely such as Authentication and Authorization of the securer agent. This attribute is also very useful for architectural design because developers are aware of the accessibility of developed or existing components before considering what techniques to use.

4.2 Architecture Design Analysis

As a hybrid web system, the meeting scheduler consists of a variety of services with different accessibilities indicated with local aspects. This makes aspect mappings between components extremely complex and error-sensitive because local aspects are not allowed in a mapping related to remote service invocation. Also, a common distributed web system may contain tens or even hundreds of components, so we need a good approach to display all the mapping between them. Thus, we define an aspect-oriented architecture view type for AOXML to support architecture designs of mobile agent systems with aspects.

The basic modeling capabilities are shown in figure 4.4. In an ao-architecture diagram, all aspect mappings between two components are represented by a raw connection, and, each connection on AOXML's architectural view actually contains a sub aspect view to define all aspect mappings involved. In addition, several rules are defined for this view type:

- A connection between two components, neither of which are mobile service agents, from different nodes is considered to be a cross-nodes connection, meaning that mappings with any local aspects are invalid.
- All connections with mobile agents are valid connections.
- Provided aspects can only be mapped to required aspects, and vice versa.

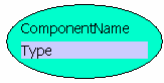




| Visual Symbol | Explanation |
|---|---|
|  | A component in UML is represented by an eclipse with its name and component type. |
|  | An architectural node in UML is represented by a rounded rectangle with its node name on the top |
|  | A raw connection indicates an original relationship between two components without any aspect mappings. |
|  | An invalid connection results from a raw connection that contains some infeasible aspect mappings. |
|  | A valid connection is resulted from a raw connection in which all mappings are verified to be feasible. |

Figure 4.4: Basic modeling capabilities of AO-Architecture View on the architecture view type

We also define the following rules for this diagram type with respect to the locality of distributed components:

- Static components are set in their residing nodes as usual
- Mobile service agents are hosted in their destination node. For instance, the secretary agent is in the agent habitat node where it provides services to other agents.
- Task agents such as negotiator agents and organizer agents are around system nodes. If a node is linked to a task agent, it means this node is on the route of the task agent. .

4.3 Domain Analysis

Up to this stage, all of the aspects analysed and designed belong to the business logic layer, which means they only focus on the problem domains that deal with the performance of business-related tasks. In the next stage of analysis, AOMA needs to take into consideration the technology domains which consist of aspects for mobile agent systems and hybrid web applications. We call this kind of aspects “technological aspect”, and most of them are implementation aspects.

Early aspects are also sometimes made of technological aspects plus conventional code. We call such aspects “composite aspects” which is in contrast to “basic aspects”. Basic aspects are those that do not contain other aspects, or in other words, contain conventional code only. Obviously the term “Basic Aspect” is named for the sake of composite aspects, which are made of basic aspects

and code. According to our study, basic aspects are mainly found in technological aspects from frameworks or toolkits. Composite aspects are referred to aspects which are made of basic aspects and conventional code. Normally, composite aspects are derived from early aspects which are identified and analysed in early phases. During system design, based on patterns or frameworks used, developers are supposed to identify basic aspects and composite aspects. We recommend that developers analyse and develop composite aspects beforehand since doing so can make the actual system development more efficient and systematic. Systems should be developed in the order of basic aspects – composite aspects – components. However, two rules should be noted here:

- If the potential composite aspect only contains basic aspects and nothing else, then it doesn't have to be made as a composite aspect. For example, the login aspect.
- If two or more basic aspects repeatedly appear together and have never been used individually, they might be merged as one basic aspect.

4.3.1 Mobile Agent Systems

In the code of MS, interaction with agent hosts and agents are the most used section of the EtherYatri framework. According to our investigation, although different frameworks may have different system architectures or operational solutions, these frameworks have some common characteristics, which we believe to be unique for mobile agent systems. Therefore, based on these characteristics, we uniform a set of aspects to handle the horizontal concerns for mobile agents as shown in figure 4.5.

| Aspect Detail | Explanation |
|----------------------|--|
| Host Switch(On/Off) | This is an aspect to switch on/off agent hosts; it is normally the starting point or the end point of an application. |
| Create/Dispose Agent | As the name, these aspects are used to create and dispose mobile agent. They could be called by agents or hosts based on frameworks used. |
| Move Agent | An aspect to move an agent to another host. As with the prior aspect, it could be called by mobile agents or hosts based on frameworks used. |
| Set Parameter | A very useful aspect to either set up or change parameters of mobile agents. |

Figure 4.5: Useful Mobile Agent Aspects

Host switch on/off are aspects to control mobile agent hosts in mobile agent system, and they are extremely useful if appropriate pointcuts are defined. Using Aspects to create, dispose and move mobile agents can make the system designs greatly independent from mobile agent frameworks used. The “Set Parameter” aspect is another useful aspect to configure parameters of mobile agents which are used for configuring various information details such as the agents' own information, schedule information and/or auditing information.

4.3.2 Hybrid Web Application

Hybrid web applications consist of a variety of distributed technologies which have their own

formats for distributed components as well as unique ways to communicate with their components. Our scheduling tool is designed as a hybrid web application which consists of web services and agents. During the development of the first prototype, we found some similarities while the distributed components interact with each other or with other types of components. Based on that, we identify a collection of aspect details which can be used to support interaction with distributed components of various technologies as shown in figure 4.6:

| Aspect Detail | Explanation |
|-------------------------------------|--|
| Proxy Lookup (Distribution) | This aspect is for the frameworks which can generate proxies for remote objects to lookup required proxies. |
| Invoke Service (Distribution) | This is a distribution aspect used to invoke services of distributed components directly or through proxies. |
| Handle Response (User Interface) | This aspect is used to handle responses of service calls. Normally, it contains codes about displaying or updating interfaces and processes for various responses. |

Figure 4.6: The Hybrid Web Applications Aspects

Some of the distributed technologies such as agents, remote objects and web services offer solutions for developers to generate proxies remotely and invoke methods through them. Therefore, developers could use an aspect for proxy lookups of these technologies, although the methods for generating proxies vary for different types of remote objects. Having an aspect to manage all proxy lookups allow common aspectual properties sharing as well as easy adaptation of related codes. Similar to the Proxy Lookup aspect, the Invoke Service aspect is also designed for hybrid web applications. Normally, depending on technologies used, services of distributed components can be invoked by proxies or some protocol messages. With this aspect, management of remote communication become much easier since remote calls have to go through this aspect. Also, effort for code adaptation is greatly reduced just as for the Proxy Lookup. Handle Response is extremely useful with the invoke service aspect for completing a remote procedure and informing users through interfaces. Two points are remarked here: firstly, a uniform format is needed for the results of any responses in order to make this aspect work properly; secondly, although this aspect usually executes immediately after the Invoke Service aspect from our experience, this is not the only case; one example is the asynchronous call mechanism of web services. So developers need to make sure this aspect is applied properly. In conclusion, the main purpose of the above aspects is to reduce as much impact from modifications of the distributed technologies as possible. Based on the needs of a system, techniques handled by these aspects can be customised accordingly.

4.3.3 Security

In the meeting scheduler, a Securer agent is used in charge of authentication and authorization of users for the service protection of a particular service. In fact it is better to explain these two security procedures as aspects provided by the Securer agent, and in a hybrid web application, these two aspects could be implemented in any manner rather than using securer agents. (see Figure 4.7) However, no matter what techniques or approaches are used for implementations, we suggest developers to weave them before all services for reliability of mobile agents.

| Aspect Detail | Explanation |
|---------------------------|--|
| Authentication (Security) | A security aspect for authenticating service callers with an id key. |
| Authorization (Security) | A security aspect grabs ids of authenticator and checks their accessibility to certain services. |

Figure 4.7: Security Aspects for Hybrid Web Applications

As known, mobile agents can move around and access services locally. That means if a mobile agent is accepted by a host, it can do whatever it wants inside that host in its life time, which can be very risky to the system. In addition, not all mobile agent frameworks provide security mechanisms for developers by default. Another issue is session management. For instance, the session of a mobile agent expires immediately after it arrives at a host. If the system doesn't check its session before it accesses any service, then it can continue on processing its schedule, which is considered to be illegal. Basically, when a service of any distributed component is being invoked, that component will pass the caller's session to the securer agent before any operation is carried on. If the caller can't be authenticated, the component will decline this request. Afterward, depending on the requirements of this service, the component may also require authorization for this caller. For instance, only roles above group leader are allowed to initialize meetings. To achieve this mission in the meeting scheduler, the component queries the securer agent about the role of the caller. The call can be processed only if the caller's role satisfies the requirements of this service.

4.4 AOMA Scheduling Analysis

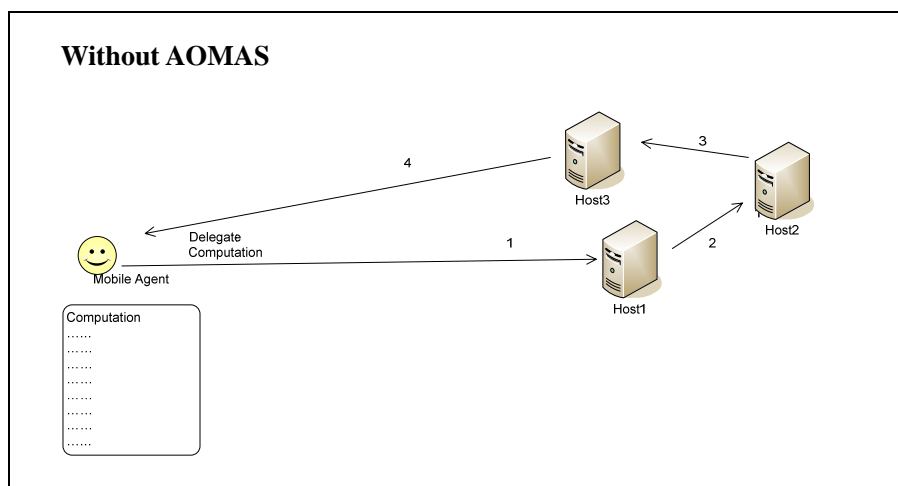


Figure 4.8a: A Program accesses remote hosts without mobile agent.

Agent activities scheduling is one of the most challenging tasks for the programming of mobile agent application, because normally as shown in figure 4.8a, computation of a mobile agent invokes services on remote hosts in a pre-defined order. Therefore, adaptation of the schedule such as changing the execution order or adding new missions becomes quite complex. Hence, we developed a novel approach to schedule and execute mobile agent events with aspects. The procedure would appear as shown in figure 4.8b.

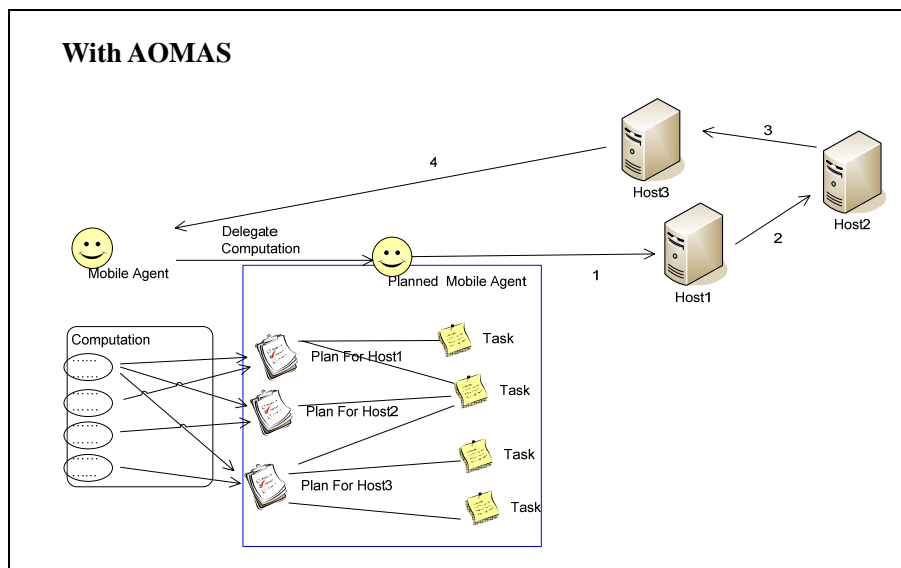


Figure 4.8b: A Program accesses remote hosts with mobile agent.

Actually, these codes can be divided into several tasks based on what executions are required on each host. Some tasks might be reusable for several plans, so developers should use certain methodologies to identify and design these tasks properly. This approach, called “Aspect Oriented Mobile Agent Scheduling” (AOMAS), contains three major facets. The first part of AOMAS is a design pattern solution which can be applied into any mobile agent frameworks. Second, developers need to implement this pattern with frameworks being used. Finally, AOMAS provides a set of aspects for developers to conduct this mechanism.

4.4.1 Design Pattern

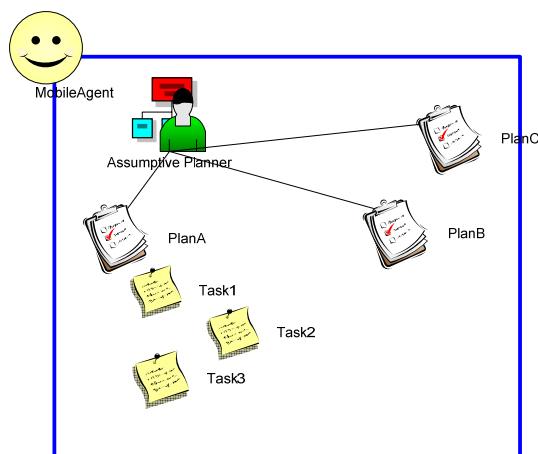


Figure 4.9: Intrinsic Architecture of AOMAS’s Design Pattern

In AOMAS, each mobile agent is associated with a planner, which is in charge of all scheduling activities with a set of plans. The Mobile Agent Scheduling Description Language (MASDL) is introduced as the communication protocol for agent scheduling aspects. (Please see the Appendix 9.2 for the full schema of the MASDL document).

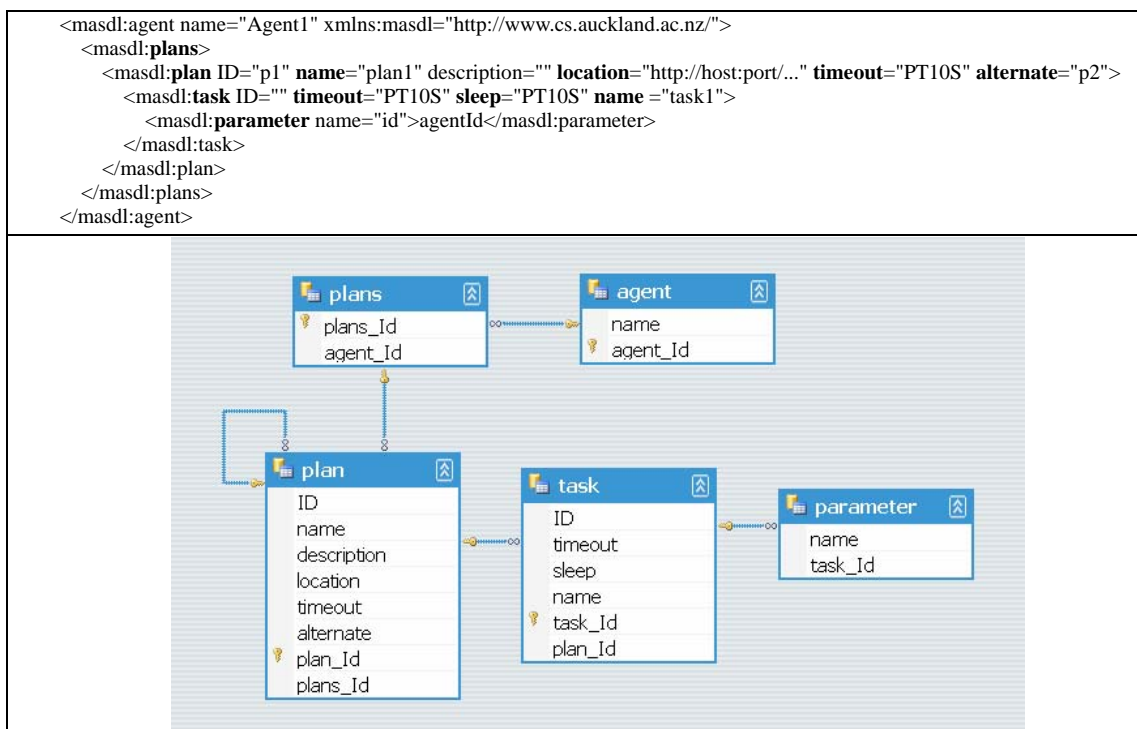


Figure 4.10: An Example and A Relation Diagram of the MASDL document

Figure 4.10 shows an example of a snippet of the MASDL document. As shown here, the MASDL document contains a set of plans, and each plan is expressed with its name, id, location to execute at, timeout for this plan and alternate plan if it fails. Moreover, use of these attributes for those tags depends on the design needs of the application. Two things need to be noted and taken into consideration here. Firstly, a planner is able to export schedule information as a MASDL document. Secondly, the parameters embedded in task tags can be set up or changed through the SetParameter aspect.

4.4.2 Implementation

Although the design pattern of AOMAS is uniform, actual implementations for specific frameworks are still required. We believe that implementations of this pattern for various mobile agent frameworks are feasible since we have successfully tested this pattern with EtherYatri and ADK.

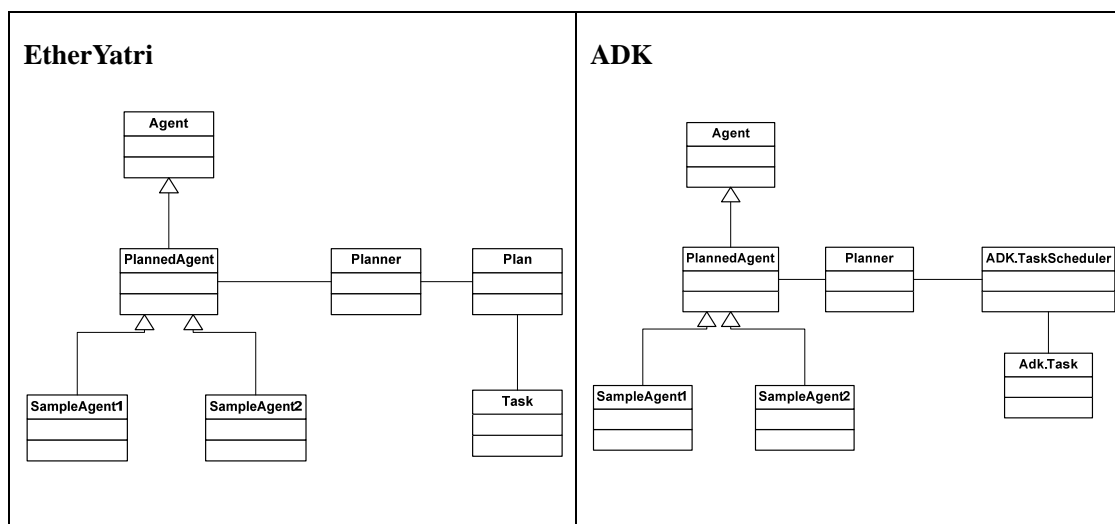


Figure 4.11: Implementations of AOMAS for EtherYatri and ADK

Two examples of the implementation of the AOMAS, EtherYatri and ADK, are shown in figure 4.11. For EtherYatri's implementation, a `PlannedAgent` class is defined to extend the `Agent` base class with a `Planner` class. Data models for plans and tasks are used to carry schedule information for planners. All operations of agent scheduling are implemented in the `PlannedAgent` class. Finally, all task agents and mobile service agents are required to extend `PlannedAgent` to use AOMAS. For ADK, some features of AOMAS are already included in `TaskScheduler` and `Task`, so they are modified to represent the plan and the task models. As with EtherYatri, a planned agent is used to implement the features of AOMAS, and task agents and mobile service agents are required to extend this class.

4.4.3 AOMAS Aspect

Most importantly, AOMAS offers a set of aspect details for developers to manage scheduling activities as well as controlling the planner. Since the design pattern of AOMAS has been implemented with agent frameworks, these aspects are provided by any agents extending the planned agent class. Moreover, use of these aspects would depend on the individual needs of an application.

| Aspect Details | Explanation |
|------------------------------|--|
| Start/Stop Planner | Starting or Stopping the planner of an agent. When a planner is started, it executes the schedule from the beginning. While it is stopped, it halts its schedule that is running no matter what step it is on, and the executing point returns back to the starting point. |
| Suspend/Resume Planner | Suspending a planner means the executing pointer is stopped and just stays on the current task till the planner is resumed. |
| Add/Push/Equeue/Remove Plans | By using these aspects, schedule can be updated at anytime with several aspectual properties to specify the position to inject or remove plans. |
| Plan Information | Return schedule information of a planned agent as an MASDL document. |

Figure 4.12: A List of details for the AOMAS Aspect

The common AOMAS aspect details of AOMAS that we identified from the practical study are

shown in figure 4.12. These aspect details are based on the scheduling planner. Their rationales are shown below:

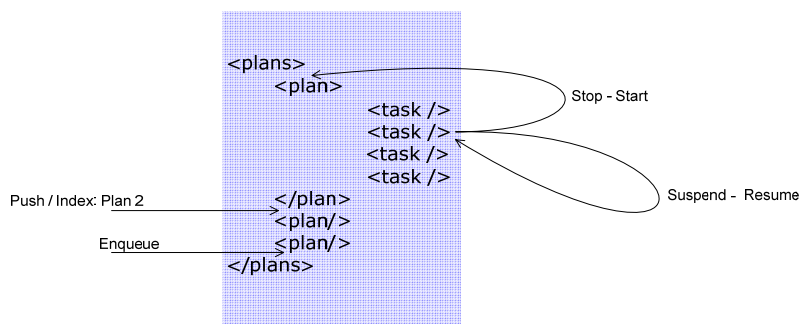


Figure 4.13: A diagram to explain the usage of AMOAS's aspects

We can imagine that a planner is similar to a media player, and the aspects are the player's functions such as Play, Stop, Suspend and Resume. Figure 4.13 shows a planner which is executing the first task on its schedule. We can see if a planner is stopped, the pointer moves back to the starting point just like restarting the media player. As such, like watching a movie on a media player, suspending a planner makes it stop before the next task, and resuming makes it continue on execution of the next task. There are three types of plan injection approaches as shown in figure as Push, Index and Enqueue. Pushing plans means to inject them right after the current plan, while enqueueing puts plans to the end of the schedule. Indexes can be used to add/remove one or more plan/plans at certain positions as shown on the above diagram. Furthermore, all of these aspects require MASDL documents for communication.

4.5 Summary

In this chapter, we analysed and discussed the design problems and crossing-cutting issues appearing in the original prototype of the meeting scheduler. We then explained our ideas and strategies to solve these problems with the AOCE model. For the requirement engineering, based on the characteristics of mobile agent systems, the modeling capacities of the component view of AOXML were enhanced with several new features, together with an extended version of the AOXML schema. Furthermore, a new aspect oriented architectural view type was introduced to support architecture designs of the mobile agent systems with aspects. We also analysed several domains for the technological aspects that could be used for system design and implementation. A novel approach called the "Aspect Oriented Mobile Agent Scheduling" was introduced to systematically schedule agent activities with aspects. This new approach also includes a design pattern and a set of aspects as illustrated and explained in this chapter. All the above techniques compose the early solution of our AOMA model. In the next chapter, with the AOXML modeling tool and the PostSharp toolkit, the meeting scheduler prototype will be reengineered using AOMA. We will also discuss the problems and issues encountered during the reengineering process.

Chapter 5 Re-engineering Meeting Scheduler with Aspects

In the previous chapter, we analysed cross-cutting issues arising during the development of the meeting scheduling tool, as well as several clumsy design and development strategies resulting from traditional OO design. To overcome these issues we introduced a novel aspect oriented mobile agent approach (AOMA). In this chapter, we depict how we implement an aspectised version of Meeting Scheduler by reengineering our PostSharp based prototype on the life cycle. We also discuss several techniques for AOP implementation and some interesting issues that we found during the development of the aspectised version of Meeting Scheduler (MS) and our technical solutions.

5.1 AO-Requirement Engineering

By using the object-oriented approach as shown in section 3.1, the requirements of the stake holders and the proposed system have been analysed and well modelled with UML's use case diagrams. Afterward, candidate components were found from the use cases. However, cross-cutting requirement properties and issues were not able to be analysed and modelled. Therefore, in section 4.1, we introduced the extended component view in AOXML together with the enhanced AOXML document for modeling early aspects of the candidate components. In this section, using the extended AOXML modeling tool, we discuss how to analyse and design various components of the Meeting Scheduler with the AOMA techniques.

5.1.1 Delegate Components

First of all, let us look at how to draw AOXML component diagrams for traditional static components. Figure 5.1 is an exemplar ao-component diagram which shows the aspectual design of the client component and its delegate components. The client component comprises three groups of aspects which are related to scheduling, contact management and security. Scheduling involves aspects that are provided and required for meetings scheduling. For a client, logon and logout are required as part of the security mechanism of this system. Meanwhile, a full list of the contacts is required for a variety of functions.

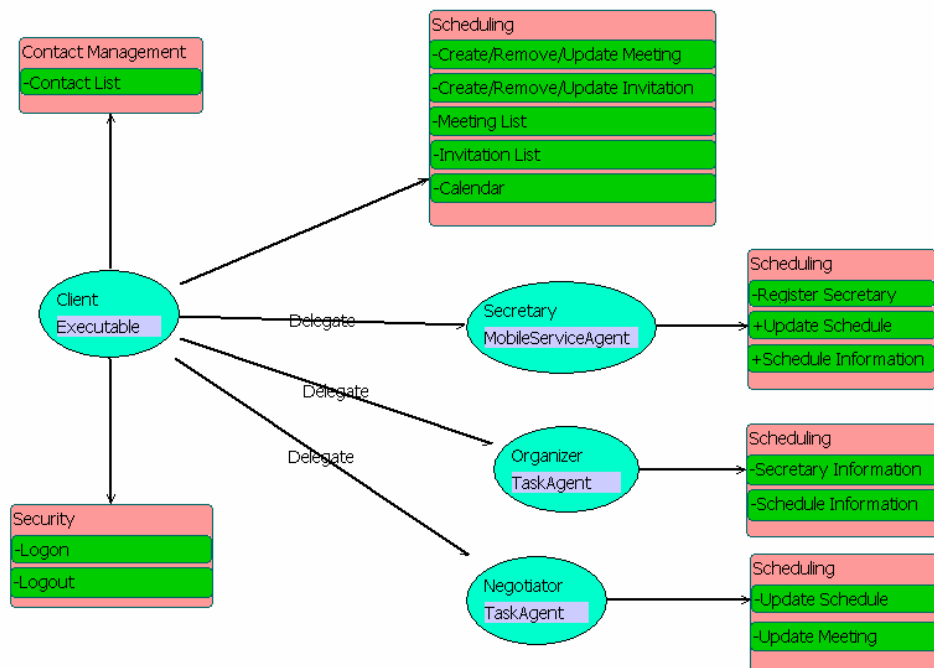


Figure 5.1: An AOURL component diagram for the client component and its delegates

To represent the relation between a delegate agent and its parent, we add a new connection notation called “delegate” to AOURL. (see section 4.1) As shown in figure 5.1, the client component contains three delegates. Among them, the secretary agent is classified as a mobile service agent, and the negotiator agent and organizer agent are task agents. Being a mobile service agent, the secretary agent requires a registration service from the system, and also provides schedule information as well as an update service. These services are actually delegated from the client component, and could also be set up on it. As such, the two task agents, the negotiator agent and the organizer agent, also have the same duty to execute client’s aspects on remote hosts.

Furthermore, AOURL can generate an AO-XML document based on the requirement level system design. A snippet of the AOXML document for the client component is shown in figure 5.2. As shown, the portion of the AOXML document tells the aspects and their details of the client component and its delegates, and it also contains other useful information such as component types, agent types and calling approaches of aspects.

```

<aoxml:application name="MeetingScheduler" xmlns:aoxml="http://www.cs.auckland.ac.nz/">
  <aoxml:components>
    <aoxml:component description="" name="Client" componentType="Executable">
      <aoxml:aspects>
        <aoxml:aspect name="Scheduling">
          <aoxml:aspect-detail name="Create/Remove/Update Meeting" provided="false"/>
          <aoxml:aspect-detail name="Create/Remove/Update Meeting" provided="false"/>
          <aoxml:aspect-detail name="Meetings List" provided="false"/>
          <aoxml:aspect-detail name="Invitations List" provided="false"/>
          <aoxml:aspect-detail name="Calendar" provided="false"/>
        </aoxml:aspect>
        <aoxml:aspect name="ContactManagement" >
          .....
        </aoxml:aspects>
      <aoxml:delegates>
        <aoxml:component description="" name="SecretaryAgent" componentType="MobileAgent"
agentType="MobileServiceAgent">

```

```

<aoxml:aspects>
  <aoxml:aspect name="Scheduling">
    <aoxml:aspect-detail name="Register Secretary" provided="false"/>
    <aoxml:aspect-detail name="Update Schedule" provided="true"
      callingType="AgentWebMethod" remote="true"/>
    <aoxml:aspect-detail name="Schedule Information" provided="true"
      callType="AgentWebMethod" remote="true"/>
  </aoxml:aspect>
</aoxml:aspects>
</aoxml:component>
<aoxml:component description="" name="NegotiatorAgent" componentType="MobileAgent"
agentType="TaskAgent">
  .....
  </aoxml:delegates>
</aoxml:component>
.....
</aoxml:components>
</aoxml:application>

```

Figure 5.2: Portion of an AOXML document for the client component and its delegates

5.1.2 Service Components

Service components can be common components or distributed components such as Meeting Manager, Contact Manager and so on. Using AOMA, service components can be designed and analysed with AOXML as shown below in figure 5.3.

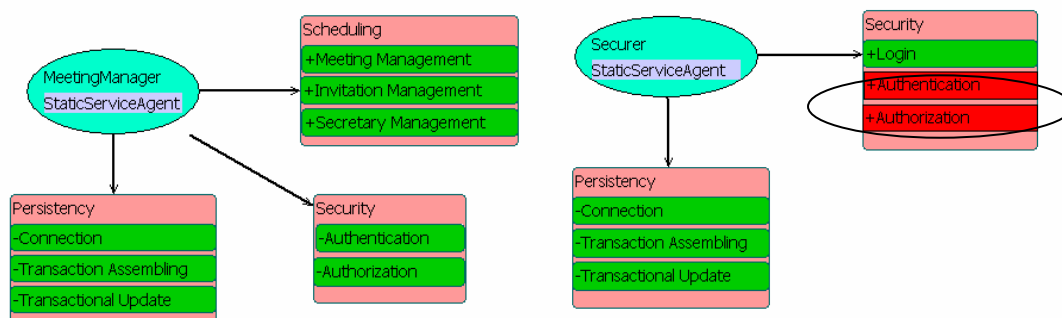


Figure 5.3: An AOXML component diagram for Meeting Manager and Securer

Meeting Manager, shown in Figure 5.3, consists of aspects for scheduling, security and persistency. Another component is the Securer agent, which includes two security aspect details, Authentication and Authorization. Authentication and Authorization are defined as local aspects represented by red cells because they are only for protection of system services and not supposed to be accessed by external components

```

<aoxml:component description="" name="MeetingManager" componentType="Agent"
agentType="StaticServiceAgent">
  <aoxml:aspects>
    <aoxml:aspect name="Scheduling">
      <aoxml:aspect-detail name="Meeting Management" provided="true"
callingType
="AgentWebMethod" remote="true"/>
      <aoxml:aspect-detail name="Invitation Management" provided="true"
callingType
="AgentWebMethod" remote="true"/>
      <aoxml:aspect-detail name="Secretary Management" provided="true"
callingType
="AgentWebMethod" remote="true"/>
    </aoxml:aspect>
    <aoxml:aspect name="Security">
      <aoxml:aspect-detail name="Authentication" provided="false"/>

```

```

.....
    </aioxml:aspect>
  </aioxml:aspects>
</aioxml:component>
<aioxml:component description="" name="Securer" componentType="Agent" agentType="StaticServiceAgent">
  <aioxml:aspects>
    <aioxml:aspect name="Security">
      <aioxml:aspect-detail name="Login" provided="true" callingType = "AgentWebMethod"
remote="true"/>
      <aioxml:aspect-detail name="Authentication" provided="true" remote="false"/>
    </aioxml:aspect>
  </aioxml:aspects>
.....
  <aioxml:component description="" name="ContactManager" componentType="WebService">
    <aioxml:aspects>
      <aioxml:aspect name="ContactManagement">
        <aioxml:aspect-detail name="Contact List" provided="true" callingType = "SOAP" remote="true"/>
        <aioxml:aspect-detail name="Add/Remove/Update Contact" provided="true" callingType = "HTTP"
remote="true"/>
      </aioxml:aspect>
      <aioxml:aspect name="Persistency">
    </aioxml:aspects>
  </aioxml:component>
.....

```

Figure 5.4: Portion of the AOXML document for the meeting manager and the securer

As mentioned previously, we have extended AOXML to support aspectual information export for hybrid web applications. Some new features are added with respect to service invocation. Figure 5.4 shows portion of the AOXML document for the meeting manager and the securer, generated by AOXML. As seen, all aspect details are described with their call types and accessibilities, which provides adequate information for hybrid web application design. A typical example here is the Securer agent, which requires accessibility restrictions for its authentication and authorization services. This attribute is very useful for architectural design because developers are aware of the accessibilities of developed or existing components before considering what techniques to use. Notably, developers should only concentrate on the aspects of business logics at this stage regardless of the actual implementation.

5.2 AO Architecture Design

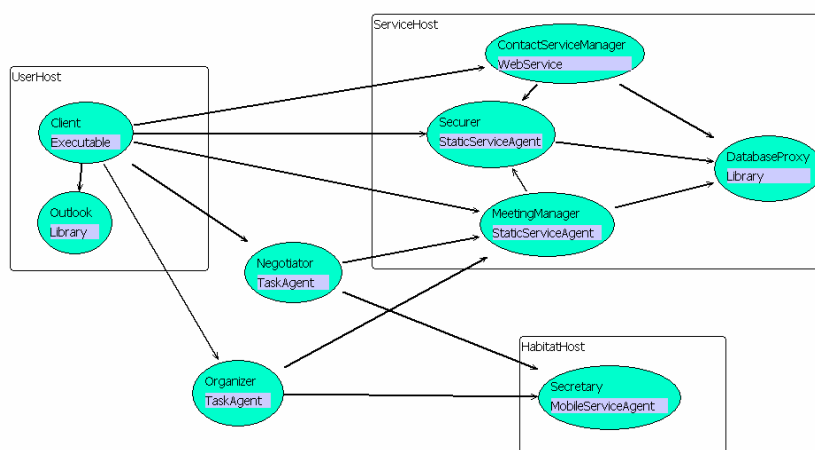


Figure 5.5: A raw AO-ADL diagram for MS

In section 4.2, we have defined an aspect-oriented ADL (AO-ADL) view type for AOXML to support architecture designs of mobile agent systems with aspects. In an AO-ADL diagram, all

aspect mappings between two components are represented by a raw connection, and, each connection on AOURL's architectural view actually contains a sub aspect view to define all aspect mappings involved. Basically, an AO-ADL diagram starts with the architect drawing basic architectural nodes, components and connections of a system. At this stage all the connections are raw connections. Figure 5.5 shows a raw AO-ADL diagram for Meeting Scheduler.

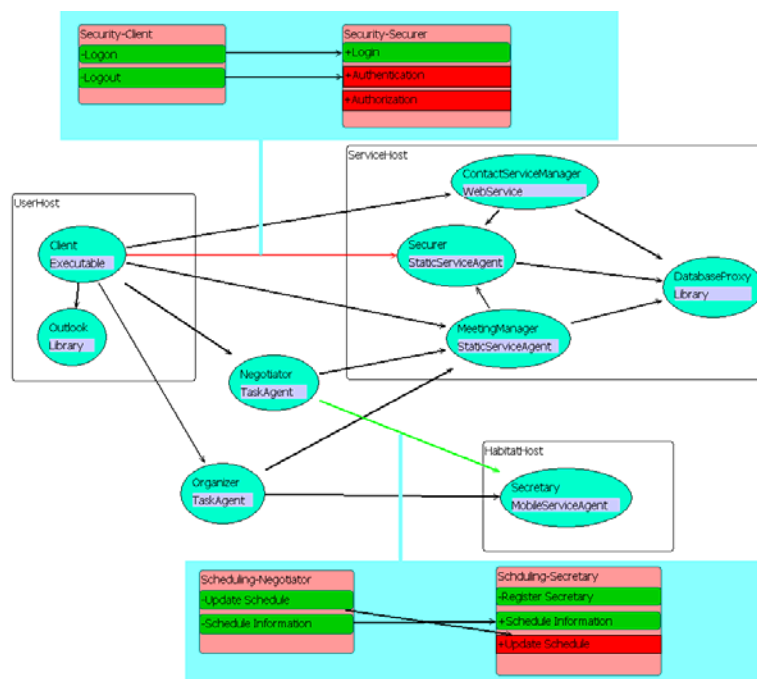


Figure 5.6: The AO-Architectural Design for MS with several verified connections

Next, each raw connection on an AO-ADL diagram must be completed with its aspect mappings on a sub-view in order to verify validity. In AOURL, each raw connection can be attached to a sub aspect view, which is created by importing the related aspects from the two connected components. For example, the aspect view for the connection between Client and Securer is show on top of Figure 5.6, while the aspect view for the connection between Negotiator and Secretary is shown at the bottom.

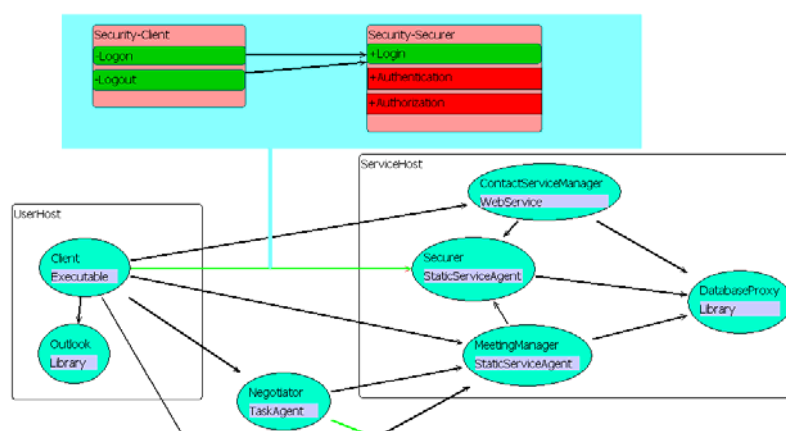


Figure 5.7: The correct mapping between Client and Securer

Once aspect mapping is finished for a connection such as in Figure 5.6, AO-UML is able to verify the validity of this connection using the rules that we defined for AO-ADL. (see section 4.2) As a result, in figure 5.6, the top connection between Client and Securer is identified as an invalid connection because `-Logout` is mapped to `+Authorization` which is a local aspect detail, while the connection for Negotiator and Secretary is consider valid since Negotiator is a mobile agent which can access local aspect details by moving to their hosts. The correct mapping for Client and Securer is shown on figure 5.7, and the connection is verified as valid this time. The ultimate goal for AO-architectural design is to complete all aspect mappings between components as well as making all connections valid.

5.3 AO System Design

AO system design covers all aspects of the structural design for the system including the technology domains such as mobile agents and hybrid web applications, converting early aspects into implementation aspects, identification and design of useful implementation aspects and so on. In this chapter, up to now, all of the aspects analysed and designed belong to business logic layer, which means they only focus on problem domains that deal with the performance of business-related tasks, such as scheduling activities and transactions in the Meeting Scheduler. From now on, the design also takes care of the technology domains which consist of aspects for mobile agent systems and hybrid web applications. Most of these aspects are implementation aspects. The structural design of a system is one of the most challenging tasks in AOCE. Thus, AO-UML's UML view provides icons and connectors for developers to draw Aspect-Oriented UML class diagrams with pre-populated aspects from the component view.

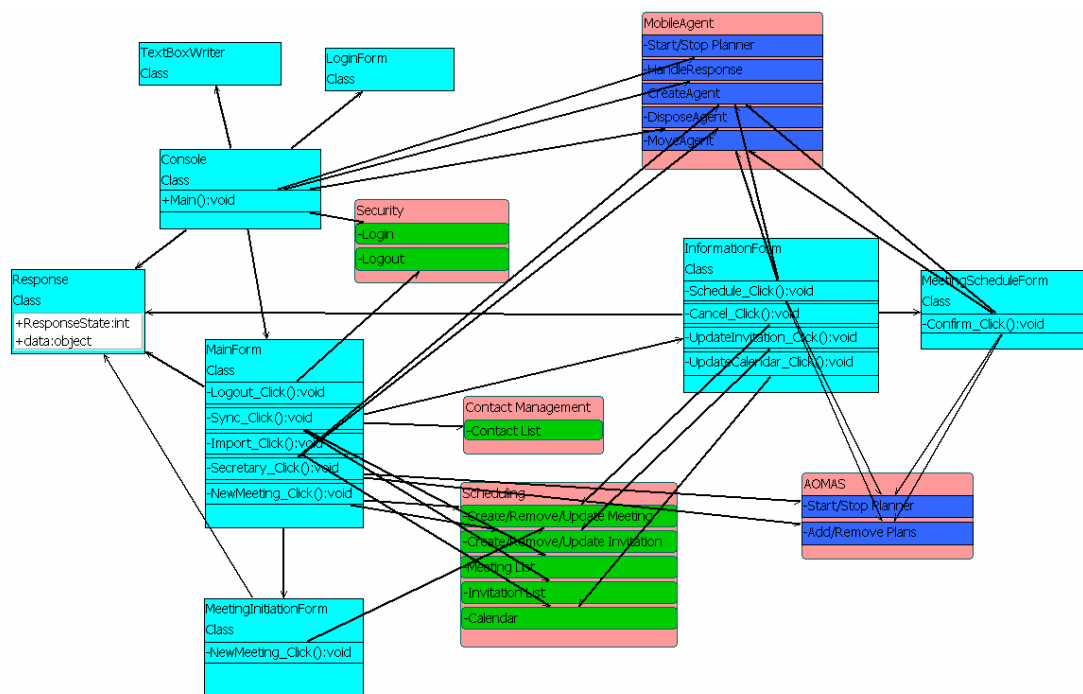


Figure 5.8: An Aspect-Oriented UML class diagram for the Client component

In figure 5.8, the client component is designed using an AO-UML class view in AO-UML with

pre-populated aspects as described in section 5.1. All elements including classes, interfaces, structs, aspects and so on are designed on this view. Aspects and methods are linked to indicate their relationships, and get ready to be implemented. In order to handle mobile agent system design, AOXML has been enhanced with several additional features as described 5.1. First, instead of green cells, blue cells are used to represent the details of what are called the “technological aspects” (see section 4.3) such as the AOMAS aspect and the Mobile Agent aspect in figure 5.8. Normally, these aspects are technological aspects that are identified and provided by frameworks or system environments.

```

<?xml version="1.0" encoding="utf-8" ?>
<aoxml:application name="MeetingScheduler" xmlns:aoxml="http://www.cs.auckland.ac.nz/">
  <aoxml:components>
    <aoxml:component description="" name="Client" componentType="Executable">
      <aoxml:aspects>
        <aoxml:aspect ID="g1" name="Scheduling">
          <aoxml:aspect-detail ID="a1" name="Create/Remove/Update Meetig" provided="false"/>
          <aoxml:aspect-detail ID="a2" name="Create/Remove/Update Meetig" provided="false"/>
          <aoxml:aspect-detail ID="a3" name="Meetings List" provided="false"/>
          <aoxml:aspect-detail ID="a4" name="Invitations List" provided="false"/>
          <aoxml:aspect-detail ID="a5" name="Calendar" provided="false"/>
        </aoxml:aspect>
        <aoxml:aspect ID="g2" name="ContactManagement" >
          <aoxml:aspect-detail ID="a6" name="Contacts List" provided="false"/>
        </aoxml:aspect>
        <aoxml:aspect ID="g3" name="Security">
          .....
          <aoxml:object ID="o1" description="" name="Console" type="Class">
            <aoxml:method ID='m1' access="public" name="Main">
              <aoxml:return type="void"/>
              <aoxml:details>
                <aoxml:aspect-detail IDREF="a7"/>
                <aoxml:aspect-detail IDREF="a9"/>
                <aoxml:aspect-detail IDREF="a10"/>
                <aoxml:aspect-detail IDREF="a12"/>
              </aoxml:details>
            </aoxml:method>
          </aoxml:object>.....
        </aoxml:component>
      </aoxml:components>
    </aoxml:application>

```

Figure 5.9: Portion of the implementation AOM-XML document for MS

Furthermore, an implementation AOXML document can be generated by AOXML to export the aspectual information of components with the structural design. As we can see in figure 5.9, in addition to the aspect information which has been generated in early phases, the document also contains information about the object elements with aspect mappings. For example, the contents in bold describes specifics of the Main method of the Console class. Through the IDREF links, we are aware of the aspect details included in this method such as “Login” and “Host Switch”.

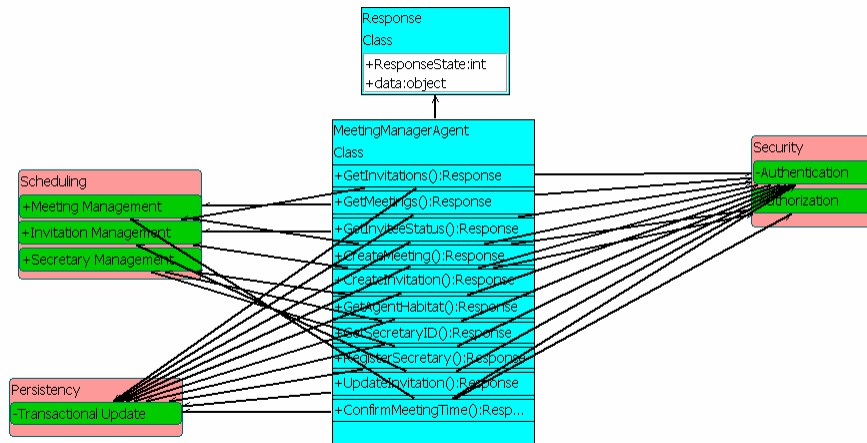


Figure 5.10: An AO-UML class diagram for the Meeting Manager Component

Another example of an AO-UML class diagram is shown in figure 5.10. As we can see, the meeting manager component comprises a planned agent class and a response class. Most methods in the meeting manager agent are actually agent web methods, which are accessible services to other components. All these services require authentication for the sake of security. Some of the services such as CreateMeeting and CreateInvitation also require authorization. Furthermore, like the Client component, the MeetingManager agent also contains a Response class which is used to assemble responses for the HandleResponse aspect.

5.4 Implementation

We choose PostSharp to implement the aspectised version of MS because it is based on C# and it supports a wide range of aspectual characteristics such as aspectual properties, various weaving approaches, global and local aspects and so on.

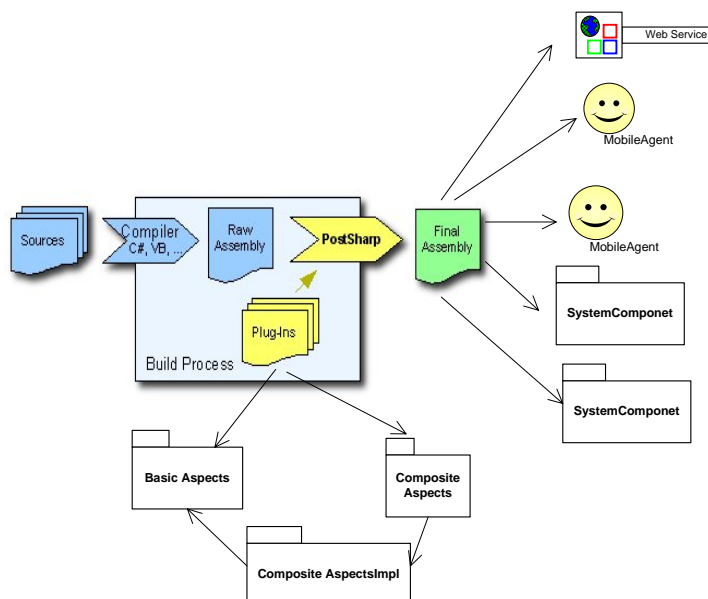


Figure 5.11: Structural Design of Meeting Scheduler with PostSharp

Based on the rationale of PostSharp, we redeveloped an aspectised version of MS. As shown in Figure 5.11, all aspects are implemented as PostSharp plug-in projects, and weaved into the raw assemblies. Through this way, all components, no matter whether they are distributed or static, can be advised before being deployed. In this section, we discuss issues found during development of the scheduling tool as well as techniques to solve them.

5.4.1 Basic Aspects

Basic aspects are those that do not contain other aspects, or in other words, contain conventional code only. Obviously, the term “Basic Aspect” is named for the sake of composite aspects, which are composed of both basic aspects and code. According to our study, basic aspects are mainly found in technological aspects from frameworks or toolkits. Figure 5.12 shows an example of implementation of basic aspects from Meeting Scheduler.

```

MeetingScheduler.ServiceComponents.Securer -- SecurerAgent.VerifyContact()
[AgentWebMethod]
[TransactionalUpdate(Connection = "Data Source=localhost;Initial Catalog=MeetingGroupDB;User ID=sa; Password =
wangywan160", Procedure = "VerifyContact", ExecuteType = ExecuteType.ExecuteScalar , ParametersCaptured =
"UserID;Password", OutputParameter = "result")]
public bool VerifyContact(string UserID, string Password, object result)
{
    if (result != null) return true;
    else return false; }

SQLServer: StoreProcedure -- VerifyContact
@UserID varchar(10), @Password varchar(10)
AS
SELECT UserID
FROM dbo.User
WHERE UserName = @UserID and Password = @Password

```

Figure 5.12: The VerifyContact method and store procedure

This example shows the code implementation for the VerifyContact method, which is the authentication service provided by the securer agent. We already know that, unlike AspectJ, PostSharp uses C# attributes to specify join points, and this is how the transactional update aspect is injected into this method, using the TransactionalUpdate attribute. This attribute also contains a collection of aspectual properties. Basically, this aspect is used to call store procedures in a SQL server with the DatabaseProxy component. In order to make this aspect work, developers need to specify the connection string for a SQL server, the name of the stored procedure, the ExecuteType to define the approach to call the stored procedure, together with ParameterCaptured and OutputParameter defining the communication protocol of this aspect. All this data is specified as aspectual properties of this attribute. In addition to the TransactionalUpdate attribute, the AgentWebMethod attribute is used to declare that this method is an agent web method.

```

MeetingScheduler.Aspects.BasicAspects.Persistency.AspectTransactionalUpdate
[AttributeUsage(AttributeTargets.Assembly | AttributeTargets.Module | AttributeTargets.Class |
AttributeTargets.Struct | AttributeTargets.Method | AttributeTargets.Constructor, AllowMultiple = true, Inherited = false)]
[MulticastAttributeUsage(MulticastTargets.Method | MulticastTargets.Constructor)]
public sealed class TransactionalUpdateAttribute : MulticastAttribute, IRequirePostSharp
{
    string connection = null;
    .....More parameters.....
    public string Connection
    {
        get { return connection; }
        set { connection = value; } }
    .....More Properties.....
    PostSharpRequirements IRequirePostSharp.GetPostSharpRequirements()
    {
        PostSharpRequirements requirements = new PostSharpRequirements();
        requirements.Tasks.Add("TransactionalUpdate");
        return requirements; }
}

```

Figure 5.13: The attribute class of the transactional update aspect

Details of the transactional update attribute are displayed in Figure 5.13. All properties used by this attribute are defined here, along with methods to link this attribute to the actual implementation of the aspect, which is a task class call “TaskTransactionalUpdate” shown in figure 5.14.

```

public class TaskTransactionalUpdate : Task, IAdviceProvider
{
    .....other parameters and methods
    .....
    public IEnumerable<KeyValuePair<MethodDeclaration, IAdvice>> GetLocalAdvices()
    {
        .....
        TransactionalUpdateAttribute attribute = (TransactionalUpdateAttribute)
            CustomAttributeHelper.ConstructRuntimeObject(customAttributeEnumerator.Current.Value,
            this.Project.Module);
        if (!attribute.AttributeExclude)
        {
            TransactionAssemblingAdvice taAdvice = new TransactinAssemblingAdvice (this,
methodDef, attribute);
            yield return new KeyValuePair<MethodDeclaration, IAdvice>(methodDef, taAdvice);
            ProcedureExecutionAdvice peAdvice = new ProcedureExecutionAdvice (this, methodDef,
attribute);
            yield return new KeyValuePair<MethodDeclaration, IAdvice>(methodDef, peAdvice);
        }
    }
    .....
}

```

Figure 5.14: The task class of the transactional update aspect

A task class is in charge of collecting all join points, analysing a few attribute properties and invoking proper advices based on these attributes. In figure 5.14, the transactional update task gathers the join points. Then, based on the exclude attribute, it decides whether to invoke the advices for transactional update. Next, we look at how this aspect is advised back to join points.

```

class ProcedureExecutionAdvice : IAdvice
{
    .....parameters.....
    public static object ExecuteScalar(string connection, string procedure, ArrayList parameters)
    {
        DatabaseProxy db = DatabaseProy.GetAProxy();
        db.OpenConnection(connection, true);
        .....Codes to require transactional update services from
        DatabaseProxy .....
    }
    public static object ExecuteCommand (string connection, string procedure, ArrayList parameters)
    { ..... }
    .....More Execute Approaches, implementation of this aspect and other configuration
    .....methods.....
    public int Priority
}

```

```

    { get { return 1; } }

    public TransactionalUpdateAdvice(AspectStoreData parent, MethodDeclaration targetMethod,
        StoreDataAttribute attribute)
    { ..... }

    public JoinPointKinds JoinPointKinds
    { get { return JoinPointKinds.BeforeMethodBody; } }
    .....

    private void WeaveEntry(WeavingContext context, InstructionBlock block)
    {
        .....here is the place to write the weaved MSIL codes for the BeforeMethodBody join point .....
        .....
        if (attribute.ExecuteType == ExecuteType.ExecuteScalar)
        {
            writer.EmitInstructionString(OpCodes.Ldstr, (LiteralString)(attribute.Connection));
            writer.EmitInstructionString(OpCodes.Ldstr, (LiteralString)(attribute.Procedure));
            writer.EmitInstructionLocalVariable(OpCodes.Ldloc, arraylistVariable);
            writer.EmitInstructionMethod(OpCodes.Call, this.parent.executeScalarMethod);
            writer.EmitInstructionParameter(OpCodes.Starg_S, result);
        }
        else if (attribute.ExecuteType == ExecuteType.ExecuteCommand
        .....
        context.InstructionWriter.DetachInstructionSequence();
    }
    .....

```

Figure 5.15: Portion of the transactional update advice class

An advice class contains the actual implementation for an aspect. The transactional update advice, shown in figure 5.15, implements several approaches to call store procedures with DatabaseProxy. The priority property is used to specify the execution order for a set of advices. The JoinPointKinds indicates where the advice is to be weaved, and it can represent one place or more from various kinds such as BeforeMethodBody, AferMethodBody, AfterMethodBodyAways, AfterMethodBodyFault and so on. For this advice, contents of the WeaveEntry method are injected before the main body of a joint method. Based on the ExecuteType property, different MSIL codes for database communication are weaved in.

```

.method public hidebysig instance bool VerifyContact(string UserID, string Password, object result) cil managed
{
    .custom instance void [EtherYatri.Core]EtherYatri.AgentWebMethodAttribute::ctor()
    = {}
    .....
    .....
    bool V_3,
    class [mscorlib]System.Delegate V_4)

IL_0000:
    newobj instance void [mscorlib]System.Collections.ArrayList::ctor()
    stloc V_4
    ldstr "UserID"
    ldarg UserID
    nop
    .....
    .....
class [mscorlib]System.Collections.ArrayList)
    stloc V_4
    ldstr "Data Source=Max-NoteBook;Initial Catalog=MeetingGroupDB;User ID=sa; Password =
wang19820711"
    ldstr "VerifyContact"
    ldloc V_4
    call                                     object
[MeetingScheduler.Aspects]MeetingScheduler.Aspects.Persistency.AspectTransactionalUpdate::ExecuteScalar(string,
string,
                                     class
[mscorlib]System.Collections.ArrayList)
    starg.s result
    nop

```

```

.....Main body of the original method.....
.....

```

Figure 5.15: The MSIL codes for the VerifyContact method after compilation

Finally, the whole component is compiled to MSIL codes, and all advices are weaved into the raw assembly. Figure 5.15 shows portion of the MSIL codes for the VerifyContact method, and the codes in bold interprets the advices weaved from the update transitional aspect. The advices are executed before the main method body based on the order of their priority properties, and then a value from the database is returned to the “result” parameter for later processing. This is roughly the whole implementation of weaving a basic aspect.

Notably, among various join point kinds, none of them can be used to support injection within the main method body, and this is actually the drawback of PostSharp as well as other aspect frameworks such as AspectJ. Therefore, we use a trick here in order to achieve this goal.

```

MeetingScheduler.WinClient – Console.Main()

.....
    if (If.DialogResult == DialogResult.OK)
    {
Result result = (Result)( JoinPoinRequiredLogin(If.ServerAddress, If.AccountID, If.Password,
null, null, null));
        if (result.State == ResponseState.Succeed)
        {
.....

        [ProxyLookup(ServiceType = MobileAgent, LookupType = LookupType.Remote, OutputParameter =
"agentFound", Type = "MeetingScheduler.ServiceAgents.SecurerAgent")]
        [InvokeService(CallType = CallType.RemoteAgentCall, AgentParameter = "agentFound", Method = "Login",
            ParametersCaptured = "userID;password;null1", OutputParameter = "result")]
        [HandleResponse(OutputParameter = "result", ParametersCaptured = "result")]
        public static object JoinPoinRequiredLogin(string URL, string userID, string password, object agentFound, object
result, object null1)
        {
            ResXResourceWriter rsxw = new ResXResourceWriter("info.resx");
            rsxw.AddResource("SID", result.ToString());
.....

```

Figure 5.17: The Weaving process of the -login aspect

Figure 5.17 shows how the login aspect within the main method is implemented. Instead of weaving codes into the method body which is infeasible in our case, an external method is created to define the join point, and the login aspect, made of three basic aspects, is called before this join point method. In reality, this login aspect is a composite aspect, which is discussed in the next section.

5.4.2 Composite Aspects

As already mentioned, composite aspects are referred to as aspects made of basic aspects and conventional code. Normally, composite aspects are derived from early aspects identified and analysed in early phases. During system design, based on patterns or frameworks used, developers are supposed to identify basic aspects and composite aspects.

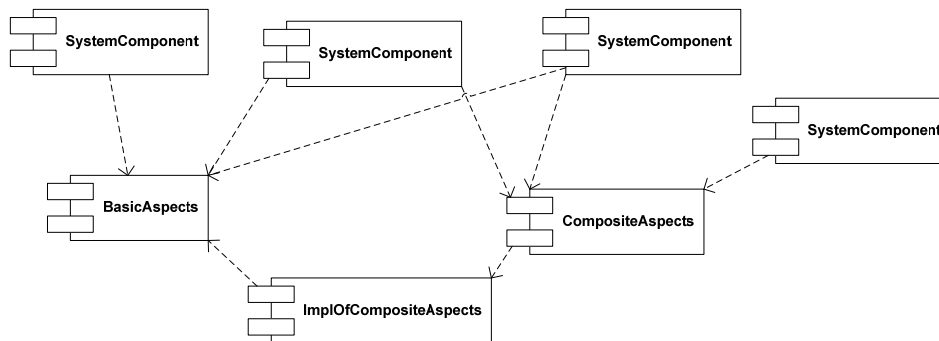


Figure 5.18: The structural design of composite aspects in PostSharp

In PostSharp, advised classes are not allowed to be stored in the same project as aspects. Thus, basic aspects are not able to be directly invoked from projects for composite aspects because actual implementation of composite aspects now become advised classes with basic aspects. As a solution for PostSharp and other analogical frameworks, the implementation of composite aspects is coded into another project which is advised by basic aspects, and then, the project for composite aspects is able to use the implementation project in their advices. An example for the composite aspect Authentication is given next.

```

MeetingScheduler.CompositeAspects.WeavedImpl.Security – AuthenticationImpl
.....
    public static object Authenticate(string sessionID)
    {
        object result = CallVerifySession(sessionID, null, null);
        if (bool.Parse(result.ToString()))
        {
            int userID = (int)(CallGetUserIDBySession(sessionID, null, null));
            return userID;
        }
        else return -1;
    }

    [ProxyLookup(ServiceType = MobileAgent, LookupType = LookupType.Local, OutputParameter =
    "agentFound", Type = "MeetingScheduler.ServiceAgents.SecurerAgent")]
    [InvokeService(CallType = CallType.LocalAgentCall, AgentParameter = "agentFound", Method =
    "VerifySession", ParametersCaptured = "sessionID", OutputParameter = "result")]
    public static object CallVerifySession(object sessionID, object agentFound, object result)
    {
        return result;
    }

    [ProxyLookup(ServiceType = MobileAgent, LookupType = LookupType.Local, OutputParameter =
    "agentFound", Type = "MeetingScheduler.ServiceAgents.SecurerAgent")]
    [InvokeService(CallType = CallType.LocalAgentCall, AgentParameter = "agentFound", Method =
    "GetUserIDBySession", ParametersCaptured = "sessionID", OutputParameter = "result")]
    public static object CallGetUserIDBySession(object sessionID, object agentFound, object result)
    {
        return result;
    }
.....

```

Figure 5.19: The actual implementation of -Authentication in the WeavedImpl project

To authenticate a caller, a service first calls the Securer agent to check the caller's session to see whether this caller already login to the system and his/her session is still valid or unexpired. If the session is invalid or expired, the request gets rejected. Otherwise, the service continues the process to ask the Securer agent for the identity of this caller. As shown in figure 5.19, the whole process involves four basic aspects for the communication with the securer agent.

```

MeetingScheduler.Aspects.BasicAspects.Persistency.AspectTransactionalUpdate
class AuthenticationAdvice : IAdvice
{

```

```

.....Pramaters.....
public IMethod authenticateMethod;
.....
public AuthenticationAdvice(AspectAuthentication parent, MethodDeclaration targetMethod,
AuthenticationAttribute attribute)
{
.....Codes for initiation.....
this.authenticateMethod = module.FindMethod(typeof(AuthenticateImpl).GetMethod("Authenticate",
new Type[] { typeof(string) }), BindingOptions.RequireLinked);
}
.....Other methods.....
private void WeaveEntry(WeavingContext context, InstructionBlock block)
{
.....here is the place to write the weaved MSIL codes for the BeforeMethodBody join point .....
.....
writer.EmitInstructionParameter(OpCodes.Ldarg, pcPD);
writer.EmitInstructionMethod(OpCodes.Call, this.parent.authenticateMethod);
writer.EmitInstructionParameter(OpCodes.Starg, opPD);
writer.DetachInstructionSequence();
}
.....

```

Figure 5.20: Portion of the -authentication advice class

Next, the AuthenticationImpl class is imported to the authentication advice, where is the workplace in another project to construct the composite aspect - Authentication. As we can see in figure 5.20, the method Authenticate is invoked by the advice as part of its weaving construction, and it is finally compiled into MISL codes to system components.

5.4.3 Security

As depicted in section 4.2.1, in a mobile agent system or a hybrid web application system, all services are supposed to be protected with authentication and authorization mechanisms. These two mechanisms can be presented as an Authentication aspect and an Authorization aspect, and could be implemented with PostSharp in this project. The implementation of the authentication aspect has been shown in the previous section, and it authenticates users based on their sessions. Normally, the Authorization aspect executes immediately after the Authentication aspect, but Authorization is not required by the service available to all users such as UpdateInvitation. Next, let us look at how the Security aspects are implemented with PostSharp.

```

MeetingScheduler.ServiceAgents -- MeetingManager.CreateMeeting()
[AgentWebMethod]
[Authentication(ParametersCaptured = "sessionID", OutputParameter = "MemberID")]
//Reject if the caller isn't authenticated
[Authorization(Key = "MemberID", Allowed = "GroupLeader,Manager")]
//Reject if the caller isn't authorizaed
[TransactionalUpdate(Connection = "Data Source=localhost;Initial Catalog=MeetingGroupDB;User ID=sa;
Password = wang19820711", Procedure = "CreateNewMeetingInstance", ExecuteType =
ExecuteType.ExecuteScalar
, ParametersCaptured = "MemberID;MeetingName;Description;Duration", OutputParameter = "result")]
public object CreateMeeting(string sessionID, object MemberID, object MeetingName, object Description,
object Duration, object result, string role)
{
return new Response(ResponseState.Succeed, result);
}

```

Figure 5.21: The CreateMeeting method of The MeetingManager class

Figure 5.21 gives an example of how Authentication and Authorization are implemented with PostSharp for this project. Firstly, callers are required to be authenticated with given session ids. If callers cannot be authenticated, their calls are rejected and responses with failed states or session

expired states are returned. Then, the system checks caller's roles with keys defined by developers. In this project, the member id from authentication is used as the key for authorization. Also, the "Allowed" property needs to be defined to specify the authorized roles. In this example, only users in the GroupLeader group and the Manager group are able to access this service, and others are rejected as unauthorized users. Finally, the actual process is executed after successful authentication and authorization.

5.4.4 Planning

The AOMAS pattern has been implemented with EtherYatri as shown in figure 4.11 of section 4.3.2. Details of the implementation are shown below in figure 5.22.

```
[Serializable]
public class Planner
{ .....Parameters.....
    public Planner(){.....}
    public int CurrentPlanIndex{.....}
    public int CurrentTaskIndex{.....}
    .....Properties.....
    public void InjectPlans(int position, string xmlDescription) {.....}
    public void EnqueuePlans(string xmlData) {.....}
    public void UpdatePlan(int position, Plan plan) {.....}
    .....Methods..... }
[Serializable]
public class Plan{.....}
[Serializable]
public class Task{.....}
```

Figure 5.22: Element classes for the AOMAS solution

A planner class is used to store plans and tasks, and it is like a tool for the planned agent to execute the schedule. As shown in figure 5.22, planner provides various functions and properties for the planned agent to implement AOMAS. You can imagine that a planner is a media player, and planned agent classes are users of this media player.

```
[Serializable]
public abstract class PlannedAgent:Agent
{
    protected Planner planner;
    protected bool running;
    .....Parameters.....
    public override void OnCreation(object init)
    {    planner = new Planner();
      running = false; .....}
    public override void Run()
    {.....}
    if (running && planner.HasNextPlan())
    {    planner.CurrentTaskIndex = 0;
      RunCurrentPlan();
      if (planner.HasNextPlan())
      {    this.MoveTo(planner.GetLocationForNextPlan());
        Console.WriteLine(agentType+" - " + this.Id.ToString() + " is moving to
        host-"+planner.GetLocationForNextPlan()+"\"); } }
    .....}
    private void RunCurrentPlan(){.....}
    protected abstract void ExecuteTask(Task task);
    [AgentWebMethod]public Result GetResult(){.....}
    [AgentWebMethod]public void StartPlanner(){.....}
    [AgentWebMethod]public void StopPlanner(){.....}
    [AgentWebMethod]public void SuspendPlanner(){.....}
    [AgentWebMethod]public void ResumePlanner(){.....}
    [AgentWebMethod]public void InjectPlans(){.....}
    [AgentWebMethod]public object EnqueuePlans(object xml) {.....}
```

```

[AgentWebMethod]public void RemovePlan(){.....}
[AgentWebMethod]public string GetPlans(){.....}
.....Methods.....
.....

```

Figure 5.23: Implementation of the PlannedAgent class

The planned agent is designed as a basic class in C# for other agent class to extend. It provides a set of services related to the AOMAS aspects mentioned previously. When the planner is started, it moves to the first destination. Then, its run() method is activated when it arrives, and then, it runs the first task of the first plan with its planner. The run method is called whenever the planned agent arrives at a new host, so the schedule can continue to run unless the planner is stopped.

```

[Serializable]
public class NegotiatorAgent : TaskAgent
{
    .....
    protected override void ExecuteTask(Task task)
    {
        switch (task.VALUE)
        {
            case "TaskPlanSchedulingEvent":
                TaskPlanSchedulingEvent();
                break;
            case "TaskGetFreeSlots":
                TaskGetFreeSlots(task.Parameters["AgentID"].ToString(), null, null, sessionID);
                break;
            case "TaskNegotiateAvailableTimes":
                TaskNegotiateAvailableTimes(task);
                break;
            case "TaskShowResult":
                TaskShowResult();
                break; } }
    .....
}

```

Figure 5.24: Implementation of the ExecuteTask method for NegotiatorAgent

All agents extending PlannedAgent are required to override the abstract method ExecuteTask. An example for the Negotiator agent is shown in figure 5.24. In this example, the computation for scheduling is divided into a set of tasks. While the procedure is running, these tasks are called with parameters from the schedule.

```

.....
private void Schedule_Click(object sender, EventArgs e)
{
    .....
    AgentProxy proxy = (AgentProxy)JoinPointCreateAgent("D:\\MeetingScheduler\\.....",
        "MeetingScheduler.TaskAgents.NegotiatorAgent", config["SID"] + "," + slots, null);
    object addResult = JoinPointAddPlans(proxy.Id.Id.ToString(), plans, null);
    object startResult = JoinPointStartPlanner(proxy.Id.Id.ToString(), null);
    .....
}

[CreateAgent(OutputParameter = result)]
private object JoinPointCreateAgent(string path, string agentType, string init, object result){return result;}

[StartPlanner(OutputParameter = "result")]
private object JoinPointStartPlanner(string AgentID, object result){return result;}

[AddPlans( InjectApproach = InjectionApproach.Enqueue, OutputParameter = "result")]
private object JoinPointAddPlans(string AgentID, string XmlData, object result){ return result;}
.....

```

Figure 5.25: An example of using the negotiator agent with AOMAS aspects in Meeting Scheduler

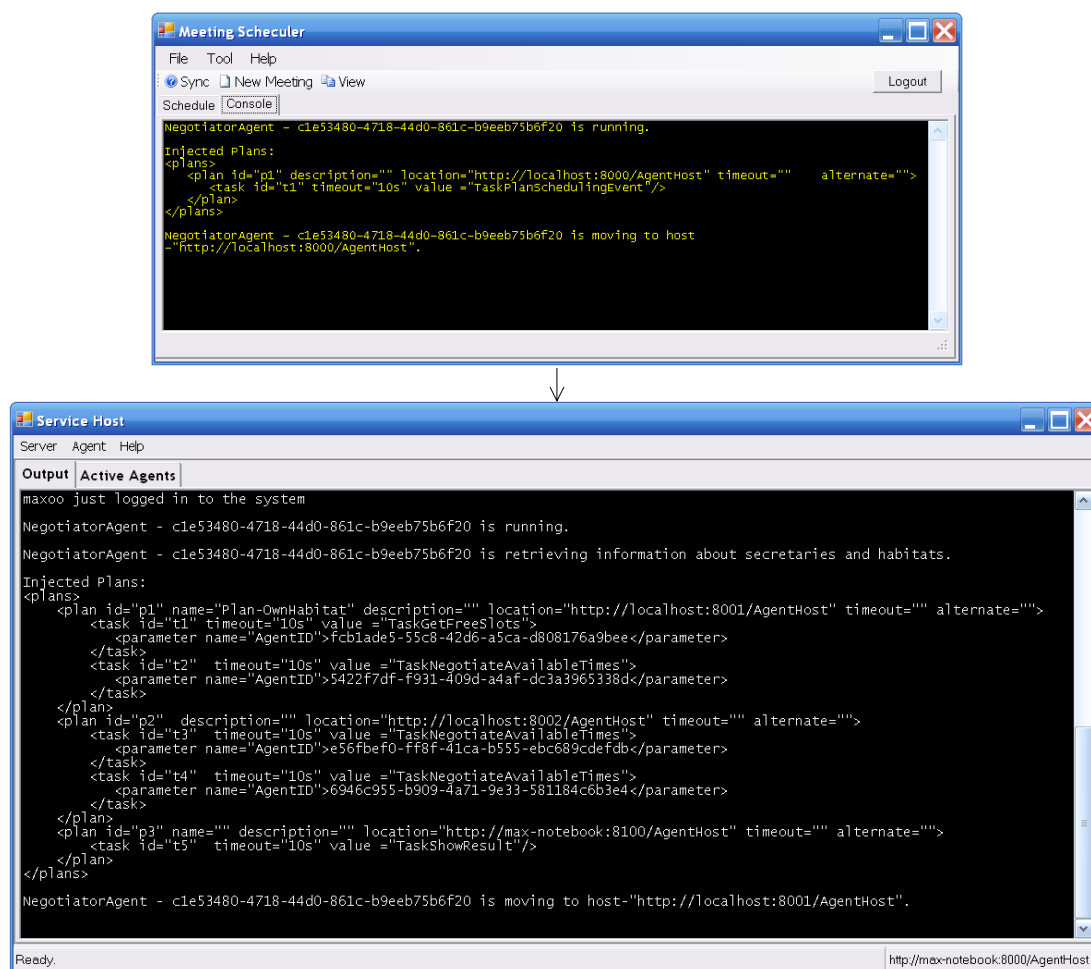


Figure 5.26: The actual implementation of meeting schedule negotiation with AOMAS

Now, let's look at how to use the Negotiator agent with the AOMAS approach. Figure 5.25 shows the `Schedule_Click` method in the meeting schedule form. Three aspects, including the two AOMAS aspects `StartPlanner` and `AddPlans`, are weaved into this method. `AddPlans` is made of a couple of basic aspects including proxy lookup and invoke service, and implemented with a set of aspectual properties such as `InjectionApproach` and some properties for its communication protocol. The communication protocol includes an `AgentID` parameter to specify the target planned agent and an `XMLData` parameter which stores the added plans in MASDL (Mobile Agent Schedule Description Language).

The actual execution of schedule negotiation is demonstrated in figure 5.26. When a Negotiator agent is first created on the user client, the system needs to initiate the schedule with the `AddPlans` aspect. A plan for moving to the service host and scheduling further plans is enqueued on the schedule. Then, as the planer is started, the negotiator agent moves to the destination of the first plan. On the service host, the Negotiator agent executes the plan to retrieve information about Secretaries and Habitats. Then, it generates a collection of new plans based on the information retrieved, and injects these plans into the schedule. Afterward, based on the next plan which is just added, it moves to the next destination which is its own Secretary habitat.

5.5 Summary

In this chapter, we have described the process of reengineering the meeting scheduler prototype with the AOMA methodology. We analysed requirements of the Meeting Scheduler with AOCRE, and model these requirements with the extended AOUML and AOXML. Then, the architectural design of the system is analysed and designed using AOUML's AO-Architecture view. Then, we used the AO-UML class view to refactor the structural design of the system with aspects. Finally, we described the specifics concerning the actual implementation of the aspectised system with PostSharp, and discussed interesting issues that we found during the process together with techniques used to handle them. In the next chapter, by experimenting and comparing the two versions of Meeting Scheduler as well as modeling another mobile agent system, we will thoroughly evaluate the AOMA methodology in several facets.

Chapter 6 Evaluation of the AOMA System

In this chapter, the AOMA methodology is evaluated using a set of criteria measures which cover a variety of aspects including system design, development and deployment. The evaluation is run with a series of scenarios that compare these measures between the two versions of the meeting scheduler. We also evaluate the capability of AOMA by developing another mobile agent system, an Online Bookshop from the Jade tutorials. At the end, the evaluation results are discussed based on the criteria measures applied.

6.1 Evaluation Criteria

In software engineering, many properties have been identified and developed to contribute the criteria of measuring software system quality. In this section, several important measures are listed and explained in details.

Scalability – In the field of software engineering, scalability is a desirable property to measure the ability of a system, an application or a process to either handle growing amounts of work in a graceful manner, or to be readily enlarged [54]. In terms of distributed systems, this property evaluates the difficulty to upgrade existing system nodes or expand with new nodes in order to improve performance.

Complexity – The complexity of a system normally represents the difficulties to design and develop the system. This property can also be explained as the understandability of a system, meaning that how easy your design and implementation can be read and understood by other developers [55]. This attribute could be measure at several levels including requirement engineering, architectural design, system design and development.

Reusability – In computer science and software engineering, reusability is the possibility of a segment of source code (implementation aspect) or a module (class or components) can be used again to add new functionalities to the existing system or new systems with slight or no modification [56]. Reusable modules reduce implementation time; increase the reliability of the system because potential bugs may have been eliminated from prior testing and use.

Traceability – This property refers to the ability to link user requirements back to stake holder's rationales and corresponding design artefacts, code and test cases [57]. It is often used to measure how good a methodology can help you to track change from user requirements to implementation, and vice-versa.

Adaptability – The adaptability of a system or an application represents its quality of being adaptable. This property is always used to measure how robust system is. Usually, software with high adaptability is robust. Adaptation of a system normally happens while its requirements or environment have to change. Adaptability can be verified at the design or the implementation

phases and run-time [58].

Usability – In computer science, usability usually refers to elegance and clarity with which the interaction with a computer program or a web site is designed. Usability is often associated with the functionalities of the software. When evaluating user interfaces for usability, the definition can be as simple as “the perception of a target user of the effectiveness and efficiency of the interface” [59].

Maintainability – In software engineering, software maintenance is the process of enhancing and optimizing deployed software (software release), as well as remedying defects. In CBSD, the maintenance activities include: gluing and wrapping components, tailoring the generic functionality of the components, fault identification and isolation, updating component configuration, monitoring and auditing system behaviour and component testing. The maintainability of a component based system can be measured by the performance of these maintenance activities [60].

6.2 Evaluation Scenarios

After discussing the criteria used for software system quality, a couple of scenarios are designed to evaluate the merits and drawbacks of AOMA. There are five scenarios discussed in this section including: 1. Requirement Change, 2. System Update, 3. Architecture Adaptation, 4. Auditing and Monitoring and 5. Development of A New System.

6.2.1 Scenario One

In the first scenario, we change a requirement to the meeting scheduler and model the refactoring for both versions to measure the performance of AOMA. The modified requirement is that “group members or common users can now initialize meetings.”

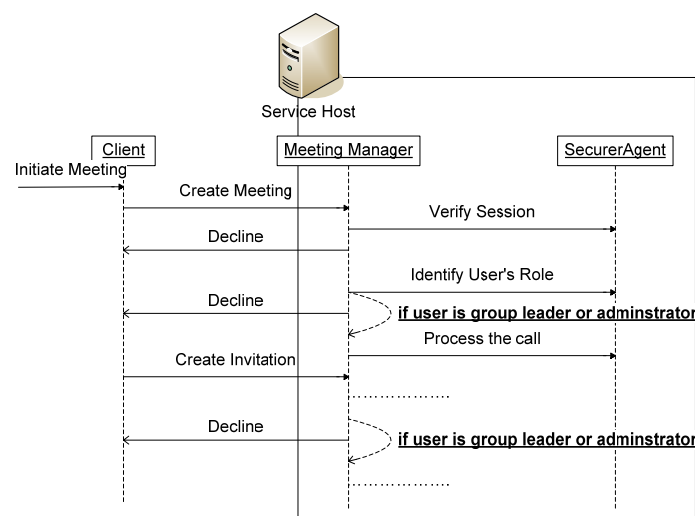


Figure 6.1: A sequence diagram showing the change in the original version

Figure 6.1 shows a brief overview of the things to change in the original version. As shown, the member group should be authorized to create meetings and invitations now. As the starting point, the involved components corresponding to this requirement need to be addressed. For the non-AOMA version, we have to figure out these components from the design and analyse source code which is sometimes unavailable for third-party users. If you are not the original developer, this task would become more challenging because components, classes or methods may not be named properly.

```

[Serializable]
class MeetingManagerAgent : EtherYatri.Agent
{
.....
[AgentWebMethod]
public object CreateMeeting(string sessionID, object MemberID, object MeetingName, object Description, object
Duration)
{.....
Code.....
else if ( role == "Manager" || role == "GroupLeader" || role == "Member" )
return new Result(ResponseState.Succeed, result);
else
.....Code
.....}

[AgentWebMethod]
public Result CreateInvitation(string sessionID, object MeetingID, object InviteeID)
{.....
Code.....
else if ( role == "Manager" || role == "GroupLeader" || role == "Member" )
return new Result(ResponseState.Succeed, result);
else
.....Code
.....}

```

Figure 6.2: The implementation change for the non-AOMA version

Figure 6.2 shows the change made for the actual implementation in the non-AOMA version. As shown, developers have to locate the exact place for authorization in each of the methods to modify. Then, member role conditions are added to both methods to suit this requirement. This step may look very simple in the meeting scheduler because it is only an exemplar application for our practical study, but such processes may become extremely complicated in large scale enterprise systems where hundreds of components are used.

The rationale of this process in the aspectised version is shown in figure 6.3. Similarly, we also start from requirement analysis. The task becomes much easier for the AOMA version as we can query about the aspect details to search for the components involved. For example, for this scenario, if we use the key word “meeting” to search for relevant aspects, a set of aspect details such as “create meeting”, “update meeting” or “remove meeting” will be found from the client and the meeting manager components. Alternatively, we can also obtain these components by searching the AOXML.

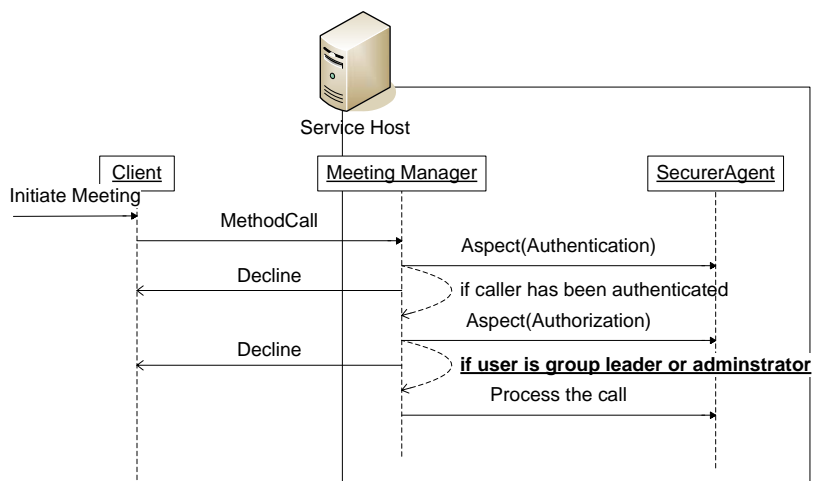


Figure 6.3: A sequence diagram showing the change in the AOMA version

As such, we also have to locate the corresponding methods and code after proper components are found. AOMA saves us much time in handling this problem as we can target the methods easily from the AO-UML diagrams where aspect details and methods have been mapped during development. The next step is to update the meeting manager component to satisfy this requirement as shown in figure 6.4. The only thing to change is the “Allowed” property of the Authorization aspect for the two methods. If we use AspectJ or other AOP frameworks that can capture pointcuts, this step can be even simpler by changing the pointcut of authorization for Create* methods.

```

[Serializable]
class MeetingManagerAgent : EtherYatri.Agent
{
.....
[Authentication(ParametersCaptured = "sessionID", OutputParameter = "MemberID")]
[Authorization(Key = "MemberID", Allowed = "GroupLeader,Manager,Member")]
[StoreData(Connection = "Data Source.....")]
[AgentWebMethod]
public object CreateMeeting(string sessionID, object MemberID, object MeetingName, object Description, object
Duration, object result, string role)
{.....}

[Authentication(ParametersCaptured = "sessionID", OutputParameter = "MemberID")]
[Authorization(Key = "MemberID", Allowed = "GroupLeader,Manager,Member")]
[StoreData(Connection = "Data Source=.....")]
[AgentWebMethod]
public Result CreateInvitation(string sessionID, object MeetingID, object InviteeID, object result, object
MemberID, string role)
{.....}
.....
}
  
```

Figure 6.4: The implementation change for the meeting scheduler

It seems there is not much difference to update the two versions as only small pieces of code are added to grant group members permissions for meeting initialization. However, if we think about the difference between addressing a line of code among hundred lines of codes and modifying an AOP aspect just right above the method constructor, it is very easy to tell their totally distinct difficulties. Furthermore, the likelihood to cause errors for the second case is much lower than the

first one as the Authorization aspect has already separated this horizontal concern from the module.

6.2.2 Scenario Two

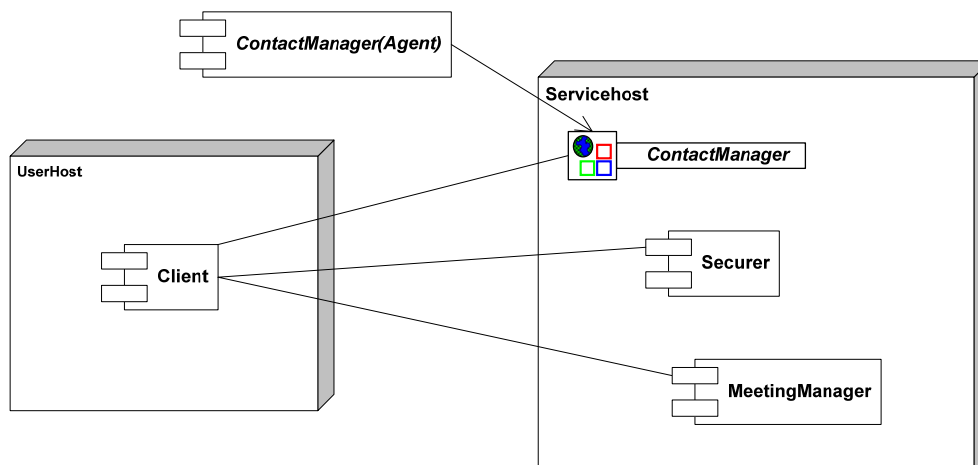


Figure 6.5: The contact manager service is replaced by an agent

The second scenario is about replacing the “Contact Manager” web service with a “Contact Manager” agent as shown in 6.5. This time, we make the “Contact Manager” as a third-party component whose source code has been sealed. As known, the original version doesn’t have aspect support because AOMA isn’t used. How could this be a huge difference from the AOMA version? Let us assume that the staff members from the maintenance department are not the original developers of this system, and don’t know too much about the fundamental design. The materials received for the two versions are shown in figure 6.6.

| The Original Version: | The Aspectised Version |
|--|---|
| The architectural design. | The ao-architectural design |
| An document describing the interface the component | An AO-XML document describing aspects, interface of the component |
| Source code of the system | Source code of the system |
| The packed “Contact Manager” agent component | The packed “Contact Manager” agent component |

Figure 6.6: A comparison of the useful information of the two versions

To start the replacement, developers need to figure out the places of the system to update. For the original version, from the architecture diagram and the description document of the component. developers can tell which components are using the component and need to be updated. In this scenario, it is the client component. Then, they have to read the whole interface description as well as source code of the client component in order to figure out the rationale and where to update. This task is difficult if you are not the original developer, and you may become lost while reading the source code. However, this task can be handled in a very simple way with AOMA for the

aspectised version. According to the ao-architectural design, we can find out what components are involved, but more importantly, this design tells us the exact aspects and methods to look at from the sub aspect view of the connection between them. Firstly, we can figure out the client component requires the contact list service from the contact manager. (see section 5.2) Then, according to the AOXML document, the Sync_Click method in the MainForm class of the client component can be located without too much effort. (Refer to section 5.1 and figure 5.8) The actual implementations of this update for the two versions are demonstrated in figure 6.7.

The Oringal Version:

```
EtherYatri.IAgentHost host = (EtherYatri.IAgentHost)Activator.GetObject(typeof(EtherYatri.IAgentHost),
    config["ServerAddress"].ToString());
AgentProxy proxy = (AgentProxy)(host.FindAgentType("MeetingScheduler.ServiceAgent.ContactManager"))[0];
info.Tables.Add((DataTable)(proxy.InvokeMethod("GetMemberList", config["SID"].ToString())));
```

The Aspectised Version:

```
[ProxyLookup(ServiceType = Agent, LookupType = LookupType.Remote, OutputParameter = "agentFound", Type =
    "MeetingScheduler.ServiceAgents.ContactManagerAgent")
[InvokeService(CallType = CallType.RemoteAgentCall, AgentParameter = "agentFound", Method =
    "GetMemberList", ParametersCaptured = "sessionId;null1;null2", OutputParameter = "result")]
[HandleResponse (ParametersCaptured = "result", OutputParameter="result")]
```

Figure 6.7: The actual implementations in this scenario

The change made for the non-AOMA version of meeting scheduler is replacing the whole block for calling the old web service with the one shown on the top of figure 6.4. For the AOMA version, only a few aspectual properties are changed as the code shown in bold at the bottom of figure 6.7. Compared to the original version, the aspectised version takes much less effort to update. With the aspect information, developers can easily configure the ProxyLookup and the InvokeService aspects to call the new mobile agent component. For the original version, we have to first address the code involved in this remote call process, and then completely replace it with a piece of new code to retrieve the agent proxy and call the target service in that proxy.

Besides the solution discussed above, there is another solution for this scenario which is an ideal case to replace the component on runtime maintenance. Although this part hasn't been implemented in our exemplar system, it provides great potential benefits and very worth to discuss here. This approach totally relies on the AOMA approach, so it is infeasible to implement in the original version.

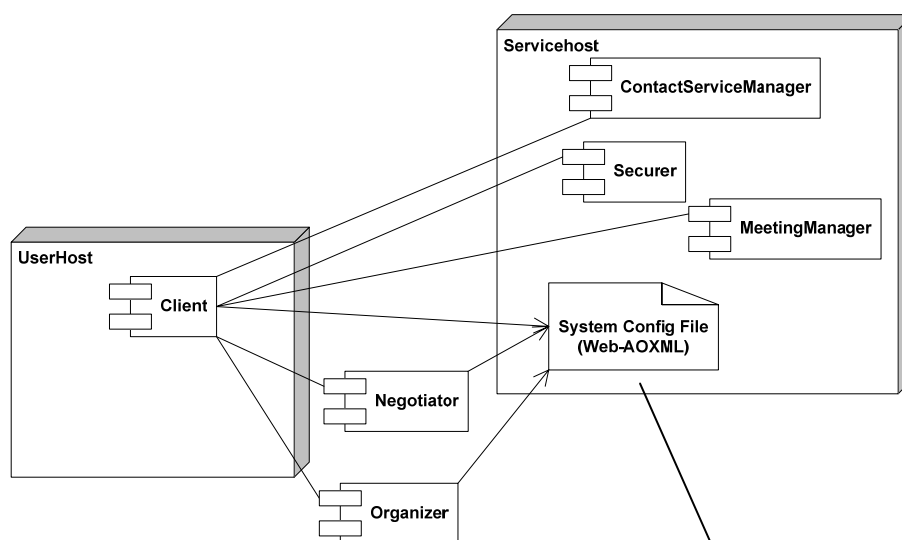


Figure 6.8: Run-Time Aspect Configuration with Web-AOXML

```

<web-aoxml:application name="MeetingScheduler" xmlns:aoxml="http://www.cs.auckland.ac.nz/">
  <web-aoxml:components>
    .....
    <aoxml:component description="" name="ContactManager" componentType="WebService" URL="http://localhost:.....">
      <aoxml:aspects>
        <aoxml:aspect name="ContactManagement">
          <aoxml:aspect-detail name="Contact List" provided="true" callingType="SOAP" remote="true">
            <aoxml:properties>
              <aoxml:property name="ServiceType">WebService</aoxml:property>
              <aoxml:property name="CallType">RemoteServiceCall</aoxml:property>
              <aoxml:property name="Type">MeetingScheduler.WebService.ContactManager</aoxml:property>
            </aoxml:properties>
          </aoxml:aspect-detail>
        </aoxml:aspect>
      </aoxml:aspects>
    </aoxml:component>
    .....
  </web-aoxml:components>
</web-aoxml:application>

<web-aoxml:application name="MeetingScheduler" xmlns:aoxml="http://www.cs.auckland.ac.nz/">
  <web-aoxml:components>
    .....
    <aoxml:component description="" name="ContactManager" componentType="MobileAgent" URL="http://localhost:.....">
      <aoxml:aspects>
        <aoxml:aspect name="ContactManagement">
          <aoxml:aspect-detail name="Contact List" provided="true" callingType="AgentWebMethod" remote="true">
            <aoxml:properties>
              <aoxml:property name="ServiceType">Agent</aoxml:property>
              <aoxml:property name="CallType">RemoteAgentCall</aoxml:property>
              <aoxml:property name="Type">MeetingScheduler.ServiceAgents.ContactManager</aoxml:property>
            </aoxml:properties>
          </aoxml:aspect-detail>
        </aoxml:aspect>
      </aoxml:aspects>
    </aoxml:component>
    .....
  </web-aoxml:components>
</web-aoxml:application>

```

Figure 6.9: A snippet of the Web-AOXML config file of Meeting Scheduler

The architectural design of this run-time aspect configuration approach is shown in figure 6.8. As seen, besides the services components of the Meeting Scheduler, there is a system config file on the service host. This file can be accessed by all clients in order to obtain the aspectual information of the system services. The format of the config file is elaborated from the AOXML schema as shown in figure 6.9, so we call this format “Web-AOXML”. The AOP aspects at bottom of figure 6.7 are dynamically configurable now, and the aspectual properties such as the component type, url, call type, service type and so on now can be retrieved from the config file. Therefore, what types of services to call and what approaches to use are no longer concerns of the developers since these issues can be figured out in the config document.

6.2.3 Scenario Three

In this scenario, based on characteristics of mobile agent applications, we make a small design change to the meeting scheduler as shown in 6.10.

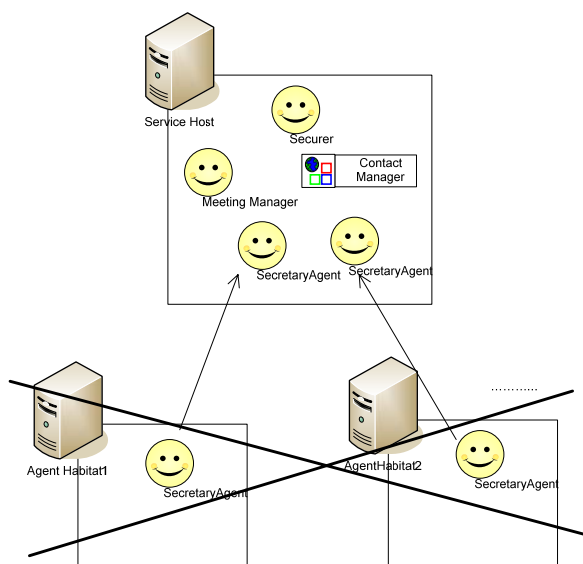


Figure 6.10: The architectural design of MS without Agent Habitats

In the original design, agent habitats are used to host mobile service agents to alleviate memory consumption of the system. Now, agent habitats are removed from the system, and all mobile service agents, only secretary agents in this case, are hosted by the service host now. The rest of the section shows how to modify the two versions of Meeting Scheduler to adapt to this change.

Three components are modified in the original Meeting Scheduler. For the secretary agent, the main change is related its run method. Code and methods for query of habitat information and moving to the corresponding habitat are removed. It is not too difficult to address issues to adapt change for this component since the process to migrate secretary agents to habitats is rather simple according to the sequence diagram. (See figure 3.11 in Chapter 3) Thus, removal of this step means the whole process is taken out, so it is quite safe to do so. Then, codes and methods related to habitat inquiry and agent migration are removed from the negotiator agent. This modification is rather complicated because it is a multi-task process. For the original version, computation of this process is highly coupled, and remaining code needs to be re-assembled. As a result, the likelihood to cause errors is rather high. Similar to the negotiator agent, change to the organizer also involves modification of codes and methods for habitat query and agent migration as well as reassembly.

The aspectised version also involves changes for the same components. By using AOMAS, all agent activities are scheduled with MASDL documents beforehand, and these MASDL documents are stored as XML files. Thus, as shown in figure 6.11, only the MASDL schedule needs to change for the Secretary in order to remove the “MoveToHabitat” task.

```

<plans>
  <plan id="p1" name="Plan-SettleCalendar" description="" location="http://localhost:8001/AgentHost" timeout=""
  alternate="">
    <task id="t1" timeout="10s" value = "TaskRegisterScheduler"/>
    <task id="t2" timeout="10s" value = "TaskMoveToHabitat"/>
  </plan>
</plans>

```

Figure 6.11: The MASDL schedule for Secretary

Modification of the negotiator component includes change of code and MASDL templates. As shown in the comparison of figure 6.12, code to modify in the AOMA version is much less than that modified for the non-AOMA version, since tasks and plans are scheduled and stored in external XML documents, so they can be customised easily. Doing so also greatly reduces the risk to cause extra errors because very little code is recompiled now. The rationale to modify the organizer class is almost the same except fewer tasks are removed or inactivated.

| Lines of Code Impacted | Without AOMA | With AOMA |
|------------------------|--------------|-----------|
| Secretary | 22 | 0 |
| Negotiator | 33 | 22 |
| Organizer | 31 | 19 |
| Overall | 88 | 41 |

Figure 6.12: Measures for the adaptability of the original MS

The most obvious difference between these two test cases is the measure of error proneness. The AOMA version has quite high error proneness since all modifications are addressed to actual code and that makes recompilation very error-susceptive. Conversely, the major change in the AOMA version is in MASDL schedules, so little code is actually modified and re-assembled. Furthermore, using AOMA also reduces the difficulty to address issues related to the change such as affected links, impacted components and rescheduling as they can be easily identified from aspects of the involved components.

6.2.4 Scenario Four

In this scenario, we demonstrate how easy it is to monitor and audit the system by using AOMA. First, we want to record all responses that a client receives on a log file as well as reporting them to the user. To achieve this task in the original version, we have to put extra code in many places following codes for remote service calls, such as most methods in the MainForm, Console and Meeting Initiation Form classes, and it is very likely we would miss some places to update. However, the work for the aspectised version is much less and can be done in minutes.

MeetingScheduler.Aspects.UserInterface.HandleResponseAdvice

```

public class HandleResponseAdvice : IAdvice
{
  public IMethod handleResponseMethod;
  .....
  public static object HandleResponse(object r)
  {

```

```

try{
    Response response = (Response)r;
    .....Code for handling response.....
}
catch(Exception e)
{ log.Record("Error:"+e.ToString());}
finally
{ log.Record("Response:"+response.ToString()); }
}

```

Figure 6.13: Codes for monitoring and auditing responses with AOMA

Figure 6.13 shows how this task is handled in the aspectised version. There is only one place to update in this case which is the `HandleResponseAdvice` class from the `HandleResponse` aspect. As mentioned in section 4.3.2, the `HandleResponse` aspect is used to process responses from services calls and update interfaces accordingly. Thus, its advice class is the exact place to handle this task because it is fired whenever the client component receives a new response. As shown in figure 6.13, by updating the weaved method “`HandleResponse`”, all responses and errors occurring are recorded on a system log.

6.2.5 Scenario Five

Besides comparing the aspectised Meeting Scheduler to the original version, we also apply AOMA for development of another mobile agent application. In this scenario, the online bookshop, a typical mobile agent system from Jade’s Tutorial, uses mobile agents as an effective solution for buyers and sellers management. The following are the requirements quoted from Jade’s document [29]:

“Each buyer agent receives the title of the book to buy (the “target book”) as a command line argument and periodically requests all known seller agents to provide offers. As soon as an offer is received the buyer agent accepts it and issues a purchase order. If more than one seller agent provides an offer the buyer agent accepts the best one (lowest price). Having bought the target book the buyer agent terminates.”

“Each seller agent has a minimal GUI by means of which the user can insert new titles (and the associated price) in the local catalogue of books for sale. Seller agents continuously wait for requests from buyer agents. When asked to provide an offer for a book they check if the requested book is in their catalogue and in this case reply with the price. Otherwise they refuse. When they receive a purchase order they serve it and remove the requested book from their catalogue.”

Other than the above requirements from Jade’s document, we also require the system to record all book sales, and provide login services for both buyers and sellers. Therefore, all users should have their own accounts in the system. Users can always check their purchases and sales through clients.

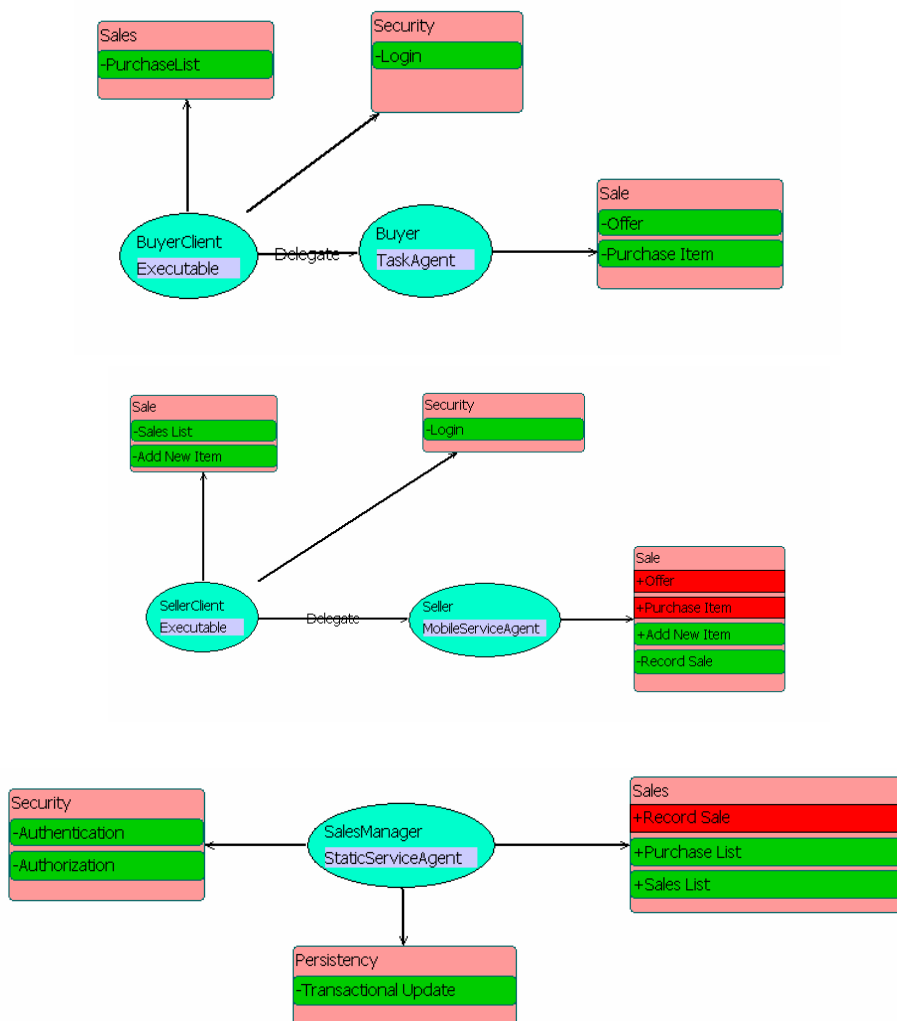


Figure 6.14: Several Candidate Components for the Online Bookshop

By using the aspectual component diagram of AOMA, developers can explore comprehensive facets of candidate components such as cross-cutting concerns, services wanted or provided, types of components, delegate relationships and accessibility of aspects. Selection of usable developed components also becomes much easier. For example, in figure 6.14, sales manager requires Authentication and Authorization for the security reason. Hence, developers could search the component repository for components that may provide these services. They can also search the service host for appropriate running components with query techniques such as AOXML for agents, AO-WSDL for web services and so on. Thus, if a securer agent, such as the one used in Meeting Scheduler, is running on this server, then it could be a perfect match to use.

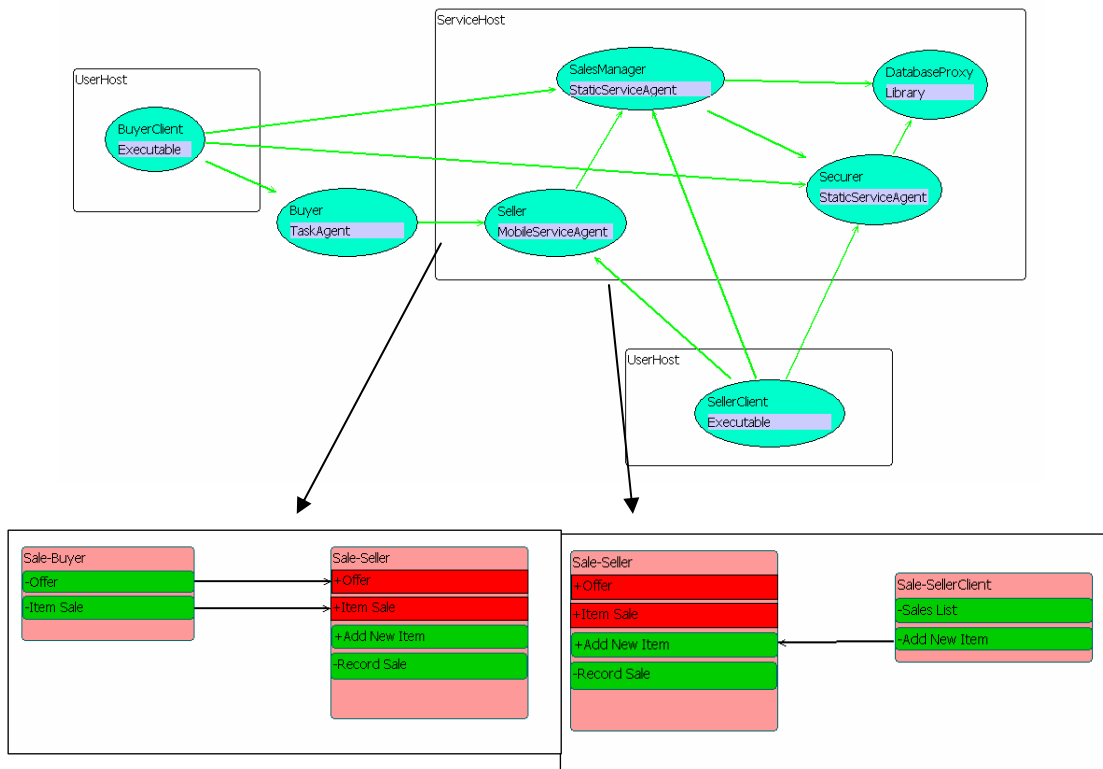


Figure 6.15: An AO-ADL diagram for the online book shop with several sub aspectual views

Afterwards, candidate components are supposed to be drawn onto AO-ADL diagrams such as the one in figure 6.15. All connections in an AO-ADL diagram are aspect mapped and verified as valid connections with their sub aspectual views. AO-ADL diagrams are very easy to use and understand because of Marama and AOUML. By using AO-ADL, locations and communications

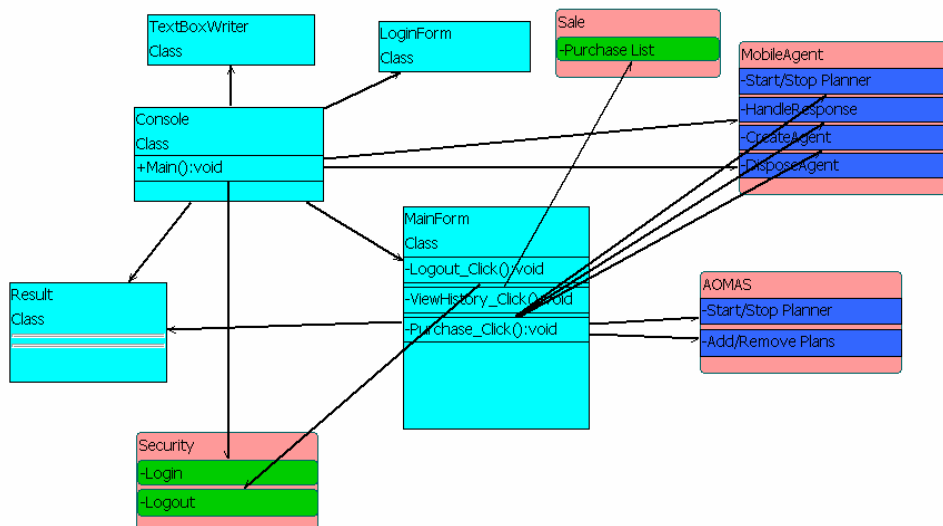


Figure 6.16: Aspectual UML diagrams for the BuyerClient

of all types of components can be clearly depicted. It is extremely useful for mobile agent systems because of their mobility concerns. More importantly, developers can view aspect mappings for each component pair in a straightforward way without getting confused of usages of massive aspect groups.

For system design, most of the technological aspects from AOMA are reusable for this application. Examples include the mobile agent aspects and the AMOAS aspects for the buyer client (see figure 6.16), and the hybrid web application aspects used in many composite early aspects. Being developers, we are very pleased to see it work because all of these technological aspects have actual implementation and can be easily imported from corresponding packages. According to this evaluation study, the domain specific aspects from AOMA are believed to be extremely useful for mobile agent applications, and developers can freely apply them to any designs.

```

OnlineBookshop.ServiceAgents -- SalesManagerAgent.GetSales()
[AgentWebMethod]
[Authentication(ParametersCaptured = "sessionID", OutputParameter = "UserID")]
[Authorization(Key = " UserID ", Allowed = "Seller")]
[TransactionalUpdate(Connection = "Data Source=localhost;Initial Catalog=MeetingGroupDB;User ID=sa; Password =
1234abcd", Procedure = "GetSalesByUserID", ExecuteType = ExecuteType.DataAdapter, ParametersCaptured =
"MemberID;MeetingName;Description;Duration", OutputParameter = "result")]
public object GetSales(string sessionID, object MemberID, object MeetingName, object Description, object Duration,
object result, string role)
{ return new Response(ResponseState.Succeed, result); }

OnlineBookshop.SellerClient -- MainForm.JoinPointGetSalesHistory()
[ProxyLookup(LookupType = LookupType.Remote, OutputParameter = "agentFound", AgentType =
"OnlineBookshop.ServiceAgents.SalesManager")]
[InvokeService(CallType = CallType.RemoteAgentCall, AgentParameter = "agentFound", Method = "GetSales",
ParametersCaptured = "userID;password:null1", OutputParameter = "result")]
[HandleResponse(OutputParameter = "result", ParametersCaptured = "result")]
public static object JoinPoinGetSalesHistory(string URL, string userID, string password, object agentFound, object
result, object null1)
{
.....
}

```

Figure 6.17: Implementation of the GetSales method of the SalesManager agent

Development of the online bookshop is much simpler than we expected. The main reason is that most technological aspects are highly reusable once they are implemented for a specific framework. As shown in figure 6.17, implemental aspects such as Authentication, Authorization and InvokeService can be imported from AOMA without change. Other frequently used aspects include proxy lookup and handle response for both buyer and seller clients. Many composite aspects converted from early aspects also have a very high reuse rate. Typical examples are the two security aspects: Authentication and Authorization. Hence, at most of time, the implementation process mainly involves assembly of technological aspects, customization of aspectual properties and defining joinpoints.

```

<plans>
  <plan id="p1" name="Plan-OnlinePurchase" description="" location="http://localhost:8001/AgentHost" timeout="" alternate="">
    <task id="t1" timeout="10s" value="TaskMakeOrder"/>
    <task id="t2" timeout="10s" value="TaskPurchaseBook">
  </plan>
  <plan id="p1" name="Plan-Return" description="" location="http://localhost:8000/AgentHost" timeout="" alternate="">
    <task id="t2" timeout="10s" value="TaskDisplayResult">
  </plan>
</plans>

```

Figure 6.18: An MASDL document for the Buyer agent

In addition, AOMAS greatly reduces development effort and error proneness for agent activities scheduling. By using MASDL, agent activities as shown in figure 6.18 are well organized and easy to read. New developers don't even need to step into the code to understand the main schedules and activities of certain agents. Furthermore, AOMAS uses tasks to represent sets of activities, so the operation can still go on even if one of its tasks fails. Thus, scheduled activities are more independent to each other now.

6.3 Evaluation Result

In this section, based on the scenarios evaluated, we discuss the impact of AOMA for the criteria measures including scalability, complexity, reusability, traceability, adaptability, usability and maintainability:

Scalability – The impact of scalability by using AOMA is not very significant, because this property mainly depends on the early architectural design and system design. It is more of a vertical modularization issue rather than a horizontal one, and it also affected by the hardware environment. Although AOMA can help identify and separate cross-cutting concerns from modules, it is not able to increase capabilities of system nodes or achieve a better architectural design. In fact, a good design of the architecture is independent from approaches being used for system development, either object-oriented or aspect-oriented. For instance, our meeting scheduler system is still fairly scalable with the design of agent habitat hosts in the first prototype while AOMA is not being used.

Complexity – The evaluation study has demonstrated that AOMA can greatly reduce the complexity of system design and development, and make the system more manageable and understandable. In scenario one, by looking at the aspectual design, developers can very easily address problems and relevant codes of service authorization because aspects provides accurate description about the horizontal interactions between components and manage them in an efficient way. This attribute is also reflected in scenario two. While the contact manger component is replaced, AOMA helps developers effectively understand its working rational and how it is used by other components. It is also the same story for the fourth scenario where the horizontal concerns are easily sliced out from all involved components. At the end, to prove simplicity of developing mobile agent system with AOMA, a mobile agent based online bookshop is developed with AOMA. As demonstrated, the design of this system is dramatically eased in various aspects by reusing domain specific aspects, analysing and identifying early aspects, using ao-architectural design approach, and implementing the system with late aspects

Reusability – High reusability is a major merit of the AMOA approach as revealed at many places of our evaluation study. For instance, the domain specific aspects from AOMA, such as the mobile agent aspects, security aspects, hybrid web application aspects and AOMAS, are highly reusable to develop mobile agent systems when we analysed and developed the online bookshop system in scenario 5 (See Figure 6.10 and 6.11). The late implementation aspects developed from early aspects can be reused in an efficient manner as well, which has also been shown on the system implementation in the same scenario (See Figure 6.13). Besides aspects, using the design pattern of AOMAS improve on mobile agent task scheduling as verified in scenario three and five.

Traceability – AOMA can assist in linking each development phase by using aspects to represent their cross-cutting concerns and interactions. User specifications can be converted to system requirements with early aspects, and packed up with components for later use and reverse tracing (See Figure 6.1 in Scenario One). With the ao-achitecture view and early aspectual requirements, developers can easily locate relevant components and the interactions of their specific requirements, and quickly address the problems or issues wanted. Examples include the scenarios one, two and three. All of them track the implementations down from the requirement level efficiently with AOXML. Therefore, from the evaluation study, it is demonstrated that using AOMA improves the traceability of the system design and development.

Adaptability – The strong flexibility of aspectual properties makes mobile agent applications more robust and adaptable. Most of the time, developers just need to reconfigure the properties of certain aspects to suit the new requirements such as shown in figure 6.2 of scenario one and figure 6.4 of scenario two. This attribute is also reflected in the high reusability of implementation aspects. For example, it is very simple to refactor the system design with the technological aspects as shown in scenario four. However, this property also depends on accurate identification of aspects as well as appropriate use of them. Thus, it requires continuous collection and elaboration of aspect sets for various domains, and formulation of their usage.

Usability – Not much evidence demonstrates that AOMA improves the usability of mobile agent systems; However, this is not unexpected. Usability mainly relies on the quality of interface design as well as work flow controls among interfaces, and these factors are rather hard to captured and modelled with aspects. One point must be cleared here: Although there are also some user interface aspects identified in AOMA, they are only aspects for capturing cross-cutting concerns of the interface classes, which makes the development process more efficient. Therefore, they are not able to either enhance functions of the GUI or make it more usable.

Maintainability – According to our evaluation, systems developed with AOMA are more maintainable because of the excellent configurability of aspects. As shown in scenario two, replacement and upgrade of existing components in the AOMA version is much simpler than this task in the non-AOMA version. Ideally, aspects could be configured at runtime to suit new patches just like the second solution in scenario two. In that case, new components can be replaced on runtime maintenance, and aspects can be reconfigured by modifying the Web-AOXML document in order to fit the components in. Aspects also facilitate monitoring and auditing of mobile agent applications. One classical example is scenario four. Instead of coding relevant tasks all over the system in the original version, responses are efficiently captured and monitored with the

HandleResponse aspect in the AOMA version.

6.4 Summary

In this chapter, we have evaluated the AOMA approach with a set of criteria measures including scalability, complexity, reusability, traceability, adaptability, usability and maintainability. The evaluation study was run with a series of scenarios that compare these measures between the two versions of Meeting Scheduler. We have also evaluated the usability and facility of AOMA by developing another mobile agent system, an Online Bookshop from the Jade tutorials. At the end, the evaluation results were discussed based on the criteria measures applied. The results show that AOMA do help software design and development in various aspects as well as improving its quality. The next chapter will conclude the contributions, future work and major achievement of this thesis project.

Chapter 7 Discussion and Conclusion

In this thesis project, we introduce a novel aspect oriented approach called the Aspect Oriented Mobile Agent system (AOMA) to provide a set of techniques, notations and domain specific aspects for the development of mobile agent systems including their use in both early and late phases of the software development life cycle. Our approach is adapted from our component-based development methodology Aspect-Oriented Component Engineering (AOCE) that uses early aspects for better categorization and reasoning about provided and required services of individual components in software systems. We describe our ideas and techniques by applying aspects to the design, development and verification of an exemplar mobile agent application in the form of a Meeting Scheduler system to show that our ideas are indeed both feasible and applicable. Our approach also uses new extended Aspect-Oriented UML notations, functions and view types to effectively and efficiently describe, capture and manage early aspects with the support of AOMA.

7.1 Contributions

There are four major contributions from this thesis including a meaningful practical study of object-oriented mobile agent system development, introduction and formulation of the Aspect Oriented Mobile Agent system approach (AOMA), a proof of concept reengineering of the meeting scheduler prototype by aspectising its design, and a thorough evaluation of AOMA's merits and drawbacks. The details of these contributions are shown in the following.

- Using traditional object oriented techniques, we developed a typical mobile agent application from scratch in the form of a meeting scheduling system in order to gain adequate practical experiences for the construction of a novel aspect oriented development approach.
- Based on the development of the meeting scheduler prototype, we formulated a novel aspect oriented approach called the Aspect Oriented Mobile Agent System (AOMA) to facilitate the development of mobile agent systems. This approach includes a set of techniques, notations and domain specific aspects for developers to design and implement mobile agent systems with both early and late aspects. We also enhanced our previously developed Aspect Oriented UML tool with new functions, capabilities and view-types to support AOMA.
- Afterwards, we verified the feasibility and applicability of the AOMA approach by reengineering the original prototype using the early aspect solution of AOMA supported by the visual AOUMML modeling tool. While we implemented the aspect-oriented design with the PostSharp toolkit, we also formalized a late aspect solution for AOMA including a set of AOP techniques and development methods.
- Finally, we ran a comprehensive evaluation that demonstrated the utility and effectiveness of AOMA by firstly comparing both versions of the meeting scheduling system applied to several scenarios and secondly applying AOMA to the development of a mobile agent based online bookshop system.

7.2 Future Research

Through the design, development and verification of the exemplar Meeting Scheduler application based on our research, we demonstrated that AOMA is an effective and efficient approach for aspect-oriented mobile agent system development. Here we will discuss some potential issues that need to be addressed to improve our approach. Although we have tested our AOMAS agent scheduling approach with both EtherYatri and ADK, it is difficult to say AOMAS is completely framework independent. Also the technological aspects for hybrid web applications were implemented in the meeting scheduling tool with considerable aspectual properties. These aspects and their properties may need further elaboration and refinement to remove any overlap and cross-cuts within the aspects and their properties.

We considered services protection via the Authentication and the Authorization aspects. Although we successfully implemented these two aspects and injected them to most of the distributed service components of the Meeting Scheduler system, we preferably need to dynamically inject these aspects into mobile service agents. As mobile service agents can migrate to another host and provide services there, injection of security aspects before compilation was found to be not suitable, because we do not want users or third party application to know the mechanisms of the security aspects. Furthermore different hosts may use different mechanisms and levels of security. Our proposed approach to address this security concern is to use runtime injection of the security aspects when the mobile service agents arrive at a certain host, and then to shell these aspects off when the agents leave. The mechanism for achieving this requires future investigation.

Our future work also includes further investigation into the implementation of delegate aspects to gauge their usefulness and performance. We will also examine the issues arising when delegate components are removed and parent components take over their responsibilities. We have tested this idea to a certain extent in the meeting scheduling tool by removing a task agent and a mobile service agent, and then re-developing their parent component with the implemented delegate aspects. However, some properties of the delegate aspects need to be changed to fit the parent component, and we need to come up with extended rules or formulas for this procedure and further enhance the AOURL tool with them for auto-conversion of the delegate aspects.

In its current form, the tasks used for AOMAS need to be pre-defined and coded into the mobile agents. This means that other components can only schedule activities of the mobile agents that are already pre-defined. This is not a very flexible approach because other components may want to arrange some tasks which are hard to predict or are not known beforehand by the creators of the mobile agents. As such, a future challenge is how to dynamically inject computation into the mobile agents together with scheduled tasks with no activities and tasks predefined during their creation.

7.3 Conclusion

In this thesis, we have proposed and successfully developed a novel aspect oriented approach called the Aspect Oriented Mobile Agent system (AOMA) based on the development and reengineering of an exemplar mobile agent system in the form of a meeting scheduling tool. We also extended the Aspect-Oriented UML language to support effective and efficient description, capture and

management of aspects, and used our visual Aspect-Oriented UML tool to carry out the design and development of the Meeting Scheduler.

AOMA provides a set of techniques, notations and domain specific aspects for the development of mobile agent systems in the whole life cycle. The formal evaluation carried out as part of this thesis work demonstrates AOMA affords mobile agent systems better adaptability and understandability because of their advanced usability and facility.

References

- [1] Pour, G. Component-based software development approach: new opportunities and challenges. *Technology of Object-Oriented Languages.*, 3 - 7 Aug. 1998 Page(s): 376 – 383
- [2] Pour, G. Moving toward Component-Based Software Development Approach. *Technology of Object-Oriented Languages. TOOLS 27. Proceedings 22 - 25 Sept. 1998* Page(s): 296 – 300
- [3] Wikipedia.org, “Commercial Off the Shelf”, <http://en.wikipedia.org/wiki/COTS/>
- [4] Yang, W., Singh, S., Hosking, J. and Grundy, J., An aspect-oriented UML tool for software development with early aspects, *Proceedings of the 2006 international workshop on Early aspects at ICSE EA '06*
- [5] Panas, T., Andersson, J. and Assmann, U. The editing aspect of aspects. In I. Hussain, editor, *Software Engineering and Applications (SEA 2002)*, Cambridge, November 2002. ACTA.
- [6] Grundy, J.C., Ding, G., and Hosking, J.G. Deployed Software Component Testing using Dynamic Validation Agents, *Journal of Systems and Software*, vol. 74, no. 1, Jan 2005, Elsevier, pp. 5-14.
- [7] Eclipse.org, Eclipse Modeling Framework (EMF), at URL: <http://www.eclipse.org/emf/>
- [8] Grundy, J. Multi-perspective specification, design and implementation of software components using aspects. *International Journal of Software Engineering and Knowledge Engineering (2000) vol.10, No. 6.*
- [9] Sutherland, J., and Heuvel, W., J., Enterprise Application Integration and Complex Adaptive Systems, *Communication of The ACM October 2002/Vol. 45, No. 10*
- [10] Self, A., L. and DeLoach, S., A., Designing and Specifying Mobility within the Multigent System Engineering Methodology, *ACM SAC 2003, Melbourne, Florida, USA*
- [11] Dianxiang Xu and Yi Deng. Modeling Mobile Agent Systems with High Level Petri Nets. *IEEE Systems, Man, and Cybernetics*, 2000, pp.3177-3182.
- [12] Loke, S., W., and Ling, S., Mobile Agent Itineraries and Workflow Nets for Analysis and Enactment of Distributed Business Processes, *Proceedings of the International Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA'2000) at the International ICSC Congress on Intelligent Systems and Applications (ISA'2000), Wollongong, Australia, (ed.) H. Tianfield, December 2000, pages 459 - 466, ICSC Academic Press.*
- [13] Sutandiyo, W., Chhetri, M., B., Krishnaswamy, S., and Loke, S., W., Experiences with Software Engineering of Mobile Agent Applications. *Proceedings of 2004 Australian Software Engineering Conference (ASWEC'04)*
- [14] Wikipedia.org, Mobile Agent, at http://en.wikipedia.org/wiki/Mobile_agent/
- [15] W3C, Web Services, at <http://www.w3.org/2002/ws/>
- [16] OMG, CORBA, at <http://www.omg.org/>

- [17] Wikipedia.org, CORBA, at <http://en.wikipedia.org/wiki/CORBA//>
- [18] Microsoft, .Net Remoting, at <http://msdn2.microsoft.com/en-us/webservices/aa740645.aspx>
- [19] Sun, Java RMI, at <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>
- [20] Araujo, J., Baniassad, E., Clements, P., Moreira, A. and Tekinerdogan, B. Early Aspects: The Current Landscape, Feb 2005.
- [21] Chitchyan, R. and Rashid, A. Survey of Aspect-Oriented Analysis and Design Approach. AOSD-Europe (May 2005).
- [22] Panas, T., Andersson, J. and Assmann, U. The editing aspect of aspects. In I. Hussain, editor, *Software Engineering and Applications (SEA 2002)*, Cambridge, November 2002. ACTA.
- [23] Singh, S., Chen, H., C., Hunter, O., Grundy, J., C. and Hosking, G., J. Improving Agile Software Development using eXtreme AOCE and Aspect Oriented CVS, APSEC 2005, Taiwan.
- [24] Baniassad, E. and Clarke, S. Theme: An approach for aspect-oriented analysis and design. Proceeding of the 26th International Conference on Software Engineering (ICSE'04) IEEE 2004.
- [25] Sampaio, A., Chitchyan, R., Rashid, A. and Rayson, P. EA-Miner: a Tool for automating aspect-oriented requirements identification. Aspect Software Engineering 2005 (ASE'05) ACM
- [26] Sampaio, A., Rashid, A. and Rayson, P. Early-AIM: An approach for identifying aspects in requirements. *Proc. 12th International Conference on Requirements Engineering (RE'05)* IEEE 2005
- [27] Singh, S., Grudy, J.C. and Hosking, J.G. Developing .NET web service-based applications with aspect-oriented component engineering. Proc. AWSA 2004 Australia.
- [28] SourceForge, Aglet, at <http://aglets.sourceforge.net/>
- [29] Bellifemine, F., Caire, G., Poggi, A. and Rimassa, G., Jade A White Paper, *exp – Volume 3 –n. 3 September 2003*
- [30] Tryllian.org, ADK, at <http://www.tryllian.org/>
- [31] Uppal, S., Agera, S. and Shelke, V., EtherYatri, a mobile agent toolkit for the Microsoft .Net platform, *A Presentation from http://etheryatri.pbwiki.com/*
- [32] Wikipedia.org, Distributed Computing, at http://en.wikipedia.org/wiki/Distributed_computing
- [33] Sun, Java RMI over IIOP, at <http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/>
- [34] Thangarathinam, T., .Net Remoting Versus Web Services, at <http://www.developer.com/net/>
- [35] Sun Developer Network (SDN), jGuru: Remote Method Invokation, at <http://java.sun.com/developer/onlineTraining/rmi/RMI.html>
- [36] Foundation for Intelligent Physical Agents, FIPA Abstract Architecture Specification, at <http://www.fipa.org/>
- [37] Wikipedia.org, JADE, at [http://en.wikipedia.org/wiki/JADE_\(programming_language\)](http://en.wikipedia.org/wiki/JADE_(programming_language))
- [38] Araujo, J., Baniassad, E., Clements, P., Moreira, A. and Tekinerdogan, B. Early Aspects: The Current Landscape, Feb 2005.

- [39] OMG.org. *Unified Modeling Language (UML) Specification*, October, 2004, version 2.0, at <http://www.uml.org/#UML2.0>
- [40] Singh, S., AO-XML Specification Version 1.0, *PhD Thesis, Department of Computer Science, 2006.*
- [41] Sequence Diagram Presentation in Technical Document Bist, B., MacKinnon, N., Murphy, S., *SIGDOC'04*, October 10-13, 2004, Memphis, Tennessee, USA
- [42] Sun Microsystems, Inc., Enterprise JavaBeans Technology, at <http://java.sun.com/products/ejb/>
- [43] PostSharp.org, PostSharp, <http://www.postsharp.org/>
- [44] Eclipse.org, The AspectJ Project, <http://www.eclipse.org/aspectj/docs.php>
- [45] SpringFramework.org, the Spring Framework, <http://www.springframework.org/>
- [46] SpringFramework.net, Spring.Net Application Framework, <http://www.springframework.net/>
- [47] Tigris.org, AspectDNG, <http://aspectdng.tigris.org/>
- [48] CastleProject.org, Aspect#, <http://www.castleproject.org/aspectsharp/index.html>
- [49] Wikipedia.org, Web Service, at http://en.wikipedia.org/wiki/Web_Service
- [50] W3C, WSDL Tutorial, <http://www.w3schools.com/wSDL/default.asp>
- [51] W3C, SOAP Specification, <http://www.w3.org/TR/soap/>
- [52] UDDI.org, UDDI Specification, <http://www.uddi.org/>
- [53] KicZales, G., Aspect-Oriented Programming, *ACM Computing Surveys (CSUR) Volume 28, Issue 4es (December 1996)*
- [54] Mark D. Hill, 'What is scalability?' *ACM SIGARCH Computer Architecture News, December 1990, Volume 18 Issue 4, pages 18-21, (ISSN 0163-5964)*
- [55] Alsharif, M., Bond, W., P. and Otaiby, T, A., Assessing the complexity of software architecture, *ACM Southeast Regional Conference, Proceedings of the 42nd annual Southeast regional conference Huntsville, Alabama*
- [56] Poulin, J., S., "Measuring Software Reusability", *Software Reuse: Advances in Software Reusability, 1994. Proceedings., Third International Conference on*
- [57] Wikipedia.org, "Traceability", <http://en.wikipedia.org/wiki/Tracability>
- [58] Wang, Q. and Liu, Xia., Study on application of a quantitative evaluation approach for software architecture adaptability, *Quality Software, 2005. (QSIC 2005). Fifth International Conference on 19-20 Sept. 2005 Page(s):265 - 272*
- [59] Ivory, M., Y. and Hearst M., A., The state of the art in automating usability evaluation of user interfaces, *ACM Computing Surveys (CSUR) Volume 33, Issue 4 (December 2001)*
- [60] Prof. Stafford, Software Maintenance as part of the software life cycle. *Department of Computer Science, Tufts University December 16, 2003*
- [61] ELCA Information SA, IIP.net, <http://iip-net.sourceforge.net/contact.html>

Appendix

9.1 The Extended AO-XML Schema

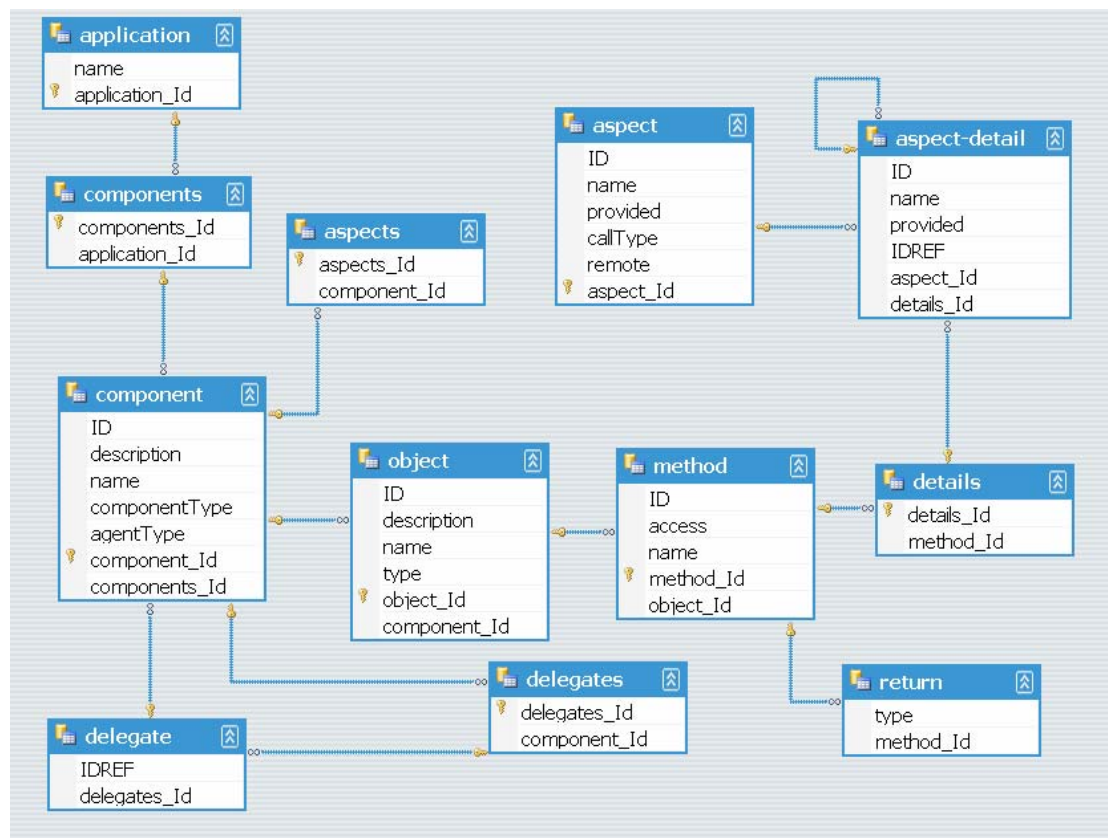


Figure 9.1: A relation diagram of the extended AO-XML schema

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="NewDataSet" targetNamespace="http://www.cs.auckland.ac.nz/" xmlns:mstns="http://www.cs.auckland.ac.nz/"
xmlns="http://www.cs.auckland.ac.nz/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" attributeFormDefault="qualified" elementFormDefault="qualified">
  <xs:element name="aspect" msdata:Prefix="aoxml">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="aspect" minOccurs="0" maxOccurs="unbounded" />
        <xs:element ref="aspect-detail" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="ID" form="unqualified" type="xs:string" />
      <xs:attribute name="name" form="unqualified" type="xs:string" />
      <xs:attribute name="provided" form="unqualified" type="xs:boolean" />
      <xs:attribute name="callType" form="unqualified" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="SOAP"/>
            <xs:enumeration value="AgentWebMethod"/>
            <xs:enumeration value="ACL"/>
            <xs:enumeration value="HTTP"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

```



```

        <xs:simpleType>
          <xs:restriction base="xs:string" >
            <xs:enumeration value="Executable"/>
            <xs:enumeration value="Library"/>
            <xs:enumeration value="WebService"/>
            <xs:enumeration value="EJB"/>
            <xs:enumeration value="Agent"/>
          </xs:restriction >
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="agentType" form="unqualified" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="StaticServiceAgent"/>
            <xs:enumeration value="MobileServiceAgent"/>
            <xs:enumeration value="TaskAgent"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="name" form="unqualified" type="xs:string" />
</xs:complexType>
</xs:element>
<xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true" msdata:Prefix="aoxml">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="aspect" />
      <xs:element ref="aspect-detail" />
      <xs:element ref="application" />
    </xs:choice>
  </xs:complexType>
</xs:element>
<xs:annotation>
  <xs:appinfo>
    <msdata:Relationship name="delegate_component" msdata:parent="delegate" msdata:child="component"
msdata:parentkey="IDREF" msdata:childkey="ID" />
    <msdata:Relationship name="aspect-detail_aspect-detail" msdata:parent="aspect-detail" msdata:child="aspect-detail"
msdata:parentkey="IDREF" msdata:childkey="ID" />
  </xs:appinfo>
</xs:annotation>
</xs:schema>

```

Figure 9.2: The full schema of the extended AO-XML document

9.2 MASDL

```

<?xml version="1.0" encoding="utf-8"?>
  <xs:schema id="NewDataSet" targetNamespace="http://www.cs.auckland.ac.nz/"
    xmlns:mstns="http://www.cs.auckland.ac.nz/" xmlns="http://www.cs.auckland.ac.nz/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified" elementFormDefault="qualified">
    <xs:element name="agent" msdata:Prefix="masdl">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="plans" msdata:Prefix="masdl" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="plan" msdata:Prefix="masdl" minOccurs="0" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="task" msdata:Prefix="masdl" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="parameter" msdata:Prefix="masdl" minOccurs="0"
maxOccurs="unbounded">
                              <xs:complexType>
                                <xs:attribute name="name" form="unqualified" type="xs:string" />
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                        <xs:attribute name="ID" form="unqualified" type="xs:string" />
                        <xs:attribute name="timeout" form="unqualified" type="xs:duration" />
                        <xs:attribute name="sleep" form="unqualified" type="xs:duration" />
                        <xs:attribute name="name" form="unqualified" type="xs:string" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                <xs:attribute name="ID" form="unqualified" type="xs:string" />
                <xs:attribute name="name" form="unqualified" type="xs:string" />
                <xs:attribute name="description" form="unqualified" type="xs:string" />
                <xs:attribute name="location" form="unqualified" type="xs:string" />
                <xs:attribute name="timeout" form="unqualified" type="xs:duration" />
                <xs:attribute name="alternate" form="unqualified" type="xs:string" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="name" form="unqualified" type="xs:string" />
  </xs:complexType>
</xs:element>
<xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true" msdata:Prefix="masdl">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="agent" />
    </xs:choice>
  </xs:complexType>
</xs:element>
<xs:annotation>
  <xs:appinfo>
    <msdata:Relationship name="plan_plan" msdata:parent="plan" msdata:child="plan" msdata:parentkey="alternate"
msdata:childkey="ID" />
  </xs:appinfo>
</xs:annotation>
</xs:schema>

```

Figure 9.3: The full schema of the MASDL document