Department of Computer Science

The University of Auckland
Te Waananga O Waipapa
Auckland, New Zealand

Supporting Web Services Systems Specification using Aspect-Oriented Component Engineering

Santokh Singh

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science at The University of Auckland.

August 2006

© 2006 Santokh Singh

This thesis is dedicated with love to my respected parents and parents-in-law, my beloved wife Nimi and our wonderful children Harveen and Ruveena, and my caring brothers, sisters, other relatives and friends.

Abstract

Web services are an emerging technology driven by the will to securely expose business logic even beyond firewalls and to achieve application-to-application integration seamlessly and dynamically irrespective of platform, language or culture. Web services have become a popular new technology for describing, locating and using distributed system functionality. The currently available web services development methodologies that use component-based systems engineering tend to expend considerable time and resources because of the effort involved in trying to understand and develop complex software based on low-level designs and implementations. This leads to wastage in terms of manpower, material and money. The bigger and more sophisticated the software system that is being considered, the greater are these overheads. These losses should be minimised or even prevented. We propose an approach that allows reasoning at higher levels through the use of a development methodology called Aspect-Oriented Component Engineering (AOCE) to support novel web services systems specifications as a solution to create more autonomous and efficient web service based systems, and at the same time try to minimize and curtail the losses and overheads involved in development processes. The building blocks of our software systems are aspect-oriented components and these are better characterised and categorised compared to traditional software components. We called these systems Aspect-Oriented Web Services (AOWS) and they are composed of a number of novel sub-systems that enable more dynamic discovery and integration to be achieved. We further designed and implemented an AOWS-based application using the AOCE methodology to demonstrate that these systems are indeed realizable. The implemented system, a collaborative Travel

Planner prototype, was also tested through various evaluation and validation techniques. We also used Alloy, a formal modelling language to model, analyse and verify the Aspect-Oriented Web Services system and its abstractions. We also designed and developed tools to support AOWS development through the use of AOCE. These tools make the development cycle of designing and creating large and complex novel aspect-oriented systems like AOWS easier to manage and control, thus increasing efficiency and effectiveness during the development process.

Acknowledgement

I wish to thank my supervisors, Professor John Grundy and Professor John Hosking for their invaluable support and guidance throughout and beyond the period of this research. Their advice and insights will definitely remain an illuminating force throughout my career.

I wish to thank my Ph D advisors, Associate Professors Robert Amor and Rick Mugridge, and other lecturers and researchers for their invaluable suggestions for the duration of my thesis.

I also wish to thank Mr Hugo Haas, the W3C's Web Services Activity Lead, a.k.a. the person in charge of the standardization of Web Services technologies at W3C, for his invaluable pointers and suggestions on the latest trends in web services technology.

I wish to acknowledge the tremendous encouragement given to me by my lovely wife Nimi and our two wonderful children Harveen and Ruveena who made this doctorate study both deeply meaningful and vastly enjoyable. Also, to the rest of my wonderful and helpful family members, relatives, and all my dear colleagues and friends, thank you for the enormous support and motivation, without which this research would have no virtue of a beginning nor any prospect of an ending.

Lastly but not least, I also wish to thank Microsoft Corporation for awarding me the inaugural Microsoft .NET Scholarship in 2003 and the Faculty of Science for their

scholarship awards for each and every year of my post graduate studies, including for this Ph D.

Supporting Web Services Systems Specification using Aspect-Oriented Component Engineering

Contents

Part I: Motivation and Background

Part II: Aspect-oriented Web Services

Part III: Dynamic Aspect-oriented Web Services

List of Figures

List of Tables

1 Introduction

Most new distributed systems now use internet technologies as a fundamental part of their remoting architecture. This has led to a demand for an open, stable, scalable and reliable software infrastructure for the development of applications for e-Businesses (Wiedemann 02). Most distributed system infrastructures and technologies, such as RMI, CORBA, DCOM, EDI and XML over TCP/IP, provide some useful techniques for abstracting remote component interfaces and supporting cross-organisational communication (Mowbray and Ruh 98, Grundy et al 98, Sessions 97). However most lack the ability to work over a wide variety of internet services with security constraints, lack adequate dynamic queryable descriptions and binding services, use proprietary solutions, or have limited cross-platform or cross-language support features, together with complex data structure representations that are specific to the language used.

One solution to overcome these problems has been the development of web services (Clark 05, Alur et al 05). These are basically remote component services described, located and accessed using a set of open standards from the World Wide Web Consortium, (W3C). As discussed in (Mockford 04), using web services gives rise to the very promising possibility of allowing heterogeneous and diverse application-to-application integration on a large-scale over both the internet and intranet. Web services have quickly become popular in large part because they build on a well known and widely accepted meta language, called the eXtensible Mark-up Language, or XML (Newcomer 02). They provide a basic communication infrastructure on which existing remote object systems, such as DCOM or CORBA, can operate, by

using HTTP as a de facto Web Service message carrier. Besides HTTP, which is currently its most popular protocol, communication can also be achieved using other protocols such as SMTP, FTP and IIOP. Web services provide a simple, standardised mechanism for describing services within service documents, and allow for locating these web services by indexing discovery agencies, and further allow for the co-ordination of cross-system processes. McKie (McKie 02) postulates web services to be the next wave in business process automation (BPA). This is because from a business perspective, web services provide a newer and better way to enhance, extend, and even reengineer the capabilities of current strategic business applications for BPA, including those expensively acquired software systems that deal with Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Supply Chain Management (SCM).

However, web services are still a maturing technology. It has been pointed out that many questions, including those pertaining to their performance, security and interoperability, are not yet answered (Hung 04 and Lee 04). In addition, most web service-based systems are currently designed using conventional object-oriented analysis and design approaches. During the development of a number of distributed systems, we have found that such design approaches do not adequately help developers to capture, reason about and encode higher level component capabilities and are especially poor with respect to addressing issues relating to cross-cutting component services.

The primary objective of this thesis is to research and propose a new breed of novel dynamic aspect-oriented web services systems that are better characterised and categorised, and to apply the Aspect-Oriented Component Engineering, (AOCE) methodology (Grundy 00, Grundy and Ding 02) to design, develop and provide

support for such systems. Furthermore, we will use a formal modelling language to model and analyse our novel aspect-oriented web services system and its abstractions, and then verify that this system and its abstractions are logically and mathematically correct. We discuss and describe in depth, key aspects of our research which provide ways to support better and more efficient description, dynamic discovery and integration mechanisms in our novel web services systems specification using AOCE. These features are either lacking or cannot be supported in existing web services technologies. The reason we chose the AOCE development methodology is that, currently used component-based systems engineering approaches for web services development are inadequate and tend to focus more on low-level software component interface design and implementation (Grundy and Patel 01). This has the problem of the techniques being both cumbersome and difficult to comprehend. This also limits, and in worst case prevents the reusability of software components produced. This results in unnecessary wastage in terms of time, effort and resources. The larger the system the more prevalent and critical these problems become, and can be an issue even in industries producing or refactoring code for commercial software tools. In this chapter we explain the motivation behind our research and give an overview of our approach of applying Aspect-Oriented Component Engineering to design and develop Aspect-Oriented Web Services (AOWS). We also provide a brief summary of the contributions of the research carried out in this thesis.

1.1 Motivation

Web services have become a very important and expanding enterprise system development technology. Their promise is to enable business-to-business integration seamlessly and dynamically irrespective of platform, language or culture (Torkelson et al 02 and Cerami 02). This has ignited the interest of researchers, developers and entrepreneurs. As discussed in (Birman et al 04), both commercial and non-commercial organizations want to use web services to enhance their software systems' capabilities by extending and integrating with internal and external systems, without rewriting or copying code, and with business logic that is already available in the intranet or internet. We will first give an overview of generic web services systems to get a firm idea about this technology. Web service based systems will be explained in greater detail in the next chapter.

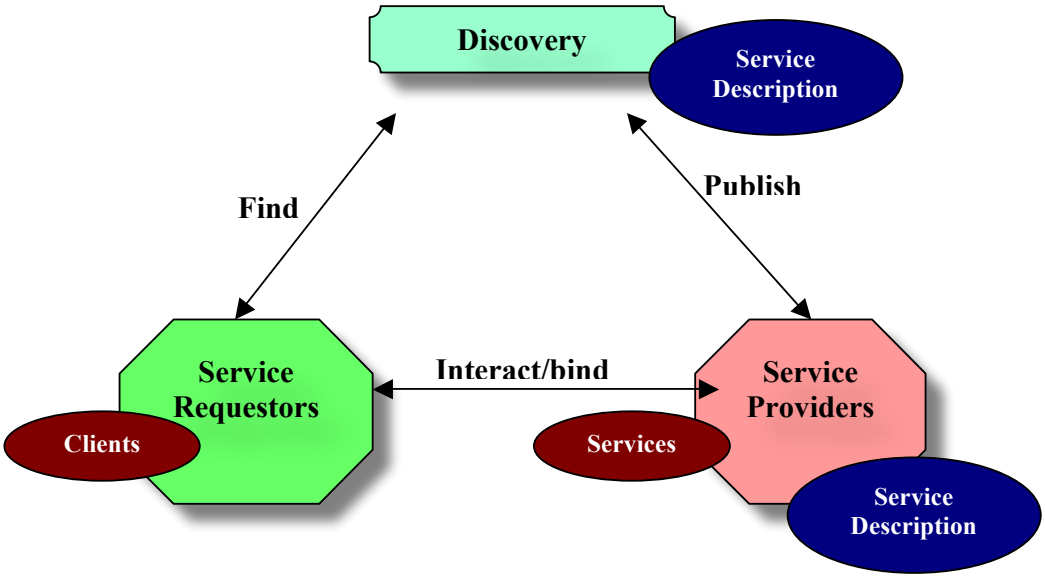


Figure 1.1: Generic web services architecture

Figure 1.1 illustrates a generic web services architecture adapted from (Booth et al, 04) comprising a variety of web service providers, web service requestors and discovery agencies. The web service providers publish their provided network services descriptions in service documents which include information about their location, exposed APIs and technical details relating to their discovery, integration and consumption. The service documents are written in XML and follow standards, e.g. set by W3C, and as such they can be easily constructed, edited, parsed and understood by users. Examples of service description documents include the Web Services Description Language (WSDL) and Business Process Execution Language for Web Services (BPEL4WS). In addition, the Web Service Choreography Interface (WSCI) is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services (Arkin, A et al 02). WSCI describes the dynamic interface of a Web Service participating in a given message exchange by means of reusing the operations defined for a static interface and hence needs to work in conjunction with WSDL descriptions.

Web service providers publish their services with discovery agencies. They do this by registering and depositing their service documents with known discovery agencies. This agency can be private and restricted to intranet access only, or exposed to selected business partners over an extranet or made public over the entire Internet so that anyone can have access to it and use it. An example of a discovery agency is the Universal Description, Discovery and Integration (UDDI) tool, e.g. the SAP UDDI Business Registry (SAP'sUDDI 05) and Microsoft's UDDI Business Registry (UBR) node (Microsoft'sUBR 05).

Web service requestors, i.e. clients, can query the discovery agencies to discover web service providers and find out about the services provided. The most popular protocol used for communication and transport of such requests and responses is the Simple Object Access Protocol (SOAP) running over the HTTP protocol. SOAP has an XML format and parsers and engines are available for its use in a variety of languages and platforms. Furthermore, a Message Exchange Pattern (MEP) (Booth et al, 04), which is a template devoid of application semantics, describes a generic pattern for the exchange of messages between parties, is used to describe the relationships (e.g., temporal, causal, sequential, etc.) of multiple messages exchanged in conformance with the pattern. MEP is also used to describe the normal (and abnormal) termination of any message exchange conforming to the pattern during the transactions between the providers and requesters. Currently, SOAP has its own MEPs described in its specifications.

If the required service is found, the web service requestors can integrate with the web service provider by constructing the appropriate proxies and connections. They can then consume the services provided for their own use by making remote procedure calls (RPC) through the proxy without going through the hassle of laboriously reinventing the wheel, i.e. they need not painstakingly rewrite the code. Also since we are using XML as the medium of communication, the requestors can be written in any language, for instance using Microsoft's C# and can be running on any platform e.g. using the Windows operating system while the providers too can be written in any language for instance written in Sun Microsystems' Java and can be running on any platform e.g. Linux machines within the internet or intranet. This notion of having language and platform independence within the web service-based systems is very attractive as we need not, at any stage, translate existing code to other languages to

make it usable and the services can even reside in remote machines running on any platform of choice.

The key benefit of web services is in developing software as a service, standardized through universally agreed specifications enabling dynamic business interoperability (Chappel and Jewell 02, Pallmann 05). They also bring about increased accessibility and efficiencies, expand and create new market opportunities and provide a mechanism to achieve legacy integration, irrespective of language or platform of the interacting remote sub-systems. It has been predicted that the number of organizations making use of web services is going to increase tremendously in the next few years (Cerami 02).

However, the currently used component-based systems engineering methodologies, including those for web service-based systems, tend to focus on low level software component interface design and implementation. This has the great disadvantage that this often results in development of components whose services are both difficult to understand and hard to combine (Grundy 00, Grundy and Hosking 02). This impacts adversely on the description, discovery and integration functionalities of current web service-based systems and further limits and hinders the extension and incorporation of dynamic capabilities into such systems. Most current development approaches also make too many assumptions about other components related to a particular web service system, constraining and limiting their reuse (we will explain this in greater detail). Furthermore the component documentation is often too low level which again makes it hard to understand at higher levels. Maintenance and refactoring of these software systems can as a result be very difficult. For remote systems like web

services, precise description and documentation is extremely important so that prospective clients wanting to consume the services can search for and understand the descriptions more correctly and accurately. Consistent and coherent high level descriptions may also assist in carrying out dynamic searches and integration more effectively and efficiently. As we mentioned above, web services are still a relatively new and maturing technology, there are still many unanswered issues concerning web services design and implementation, including those relating to security, performance, collaboration and interoperability. We propose the use of Aspect-oriented Component Engineering (AOCE) to develop novel aspect-oriented web services to help to overcome these limitations and address the issues raised.

1.2 Our Approach

The Aspect-Oriented Component Engineering approach describes functional and non-functional characteristics of software *components* using aspects, including their details and properties (Grundy and Ding 02). In this thesis we extend this earlier work to the use of AOCE to design, develop and refactor novel aspect-oriented *web service-based* systems the building blocks of which are aspect-oriented components. We show how we identify and capture rich cross-cutting concerns for highly distributed software systems and how we use this captured information as descriptors in novel service documents to enable better description and more autonomous discovery and integration than the currently available systems. We also use a formal modelling language called Alloy to construct formal models of our AOWS abstractions and analyse and verify that these models are formally correct, serving as a means of formally evaluating our software approach.

Aspect-Oriented Component Engineering (AOCE) (Grundy 00) uses a concept of different system capabilities called aspects to categorize and reason about inter-component operations. Our work on AOWS extends this to do the same for inter-component services. These services can be either *provided* or *required*. AOWS supports identification, description and reasoning about high-level component functional and non-functional requirements grouped by different systemic aspects. These requirements can then be refined into design-level software component service implementation aspects for the web services system. Aspect-oriented discovery agencies and web services description languages are also developed using AOCE

concepts to support efficient and effective description, discovery and integration of the highly characterised, componentised and integrated aspect-oriented web services.

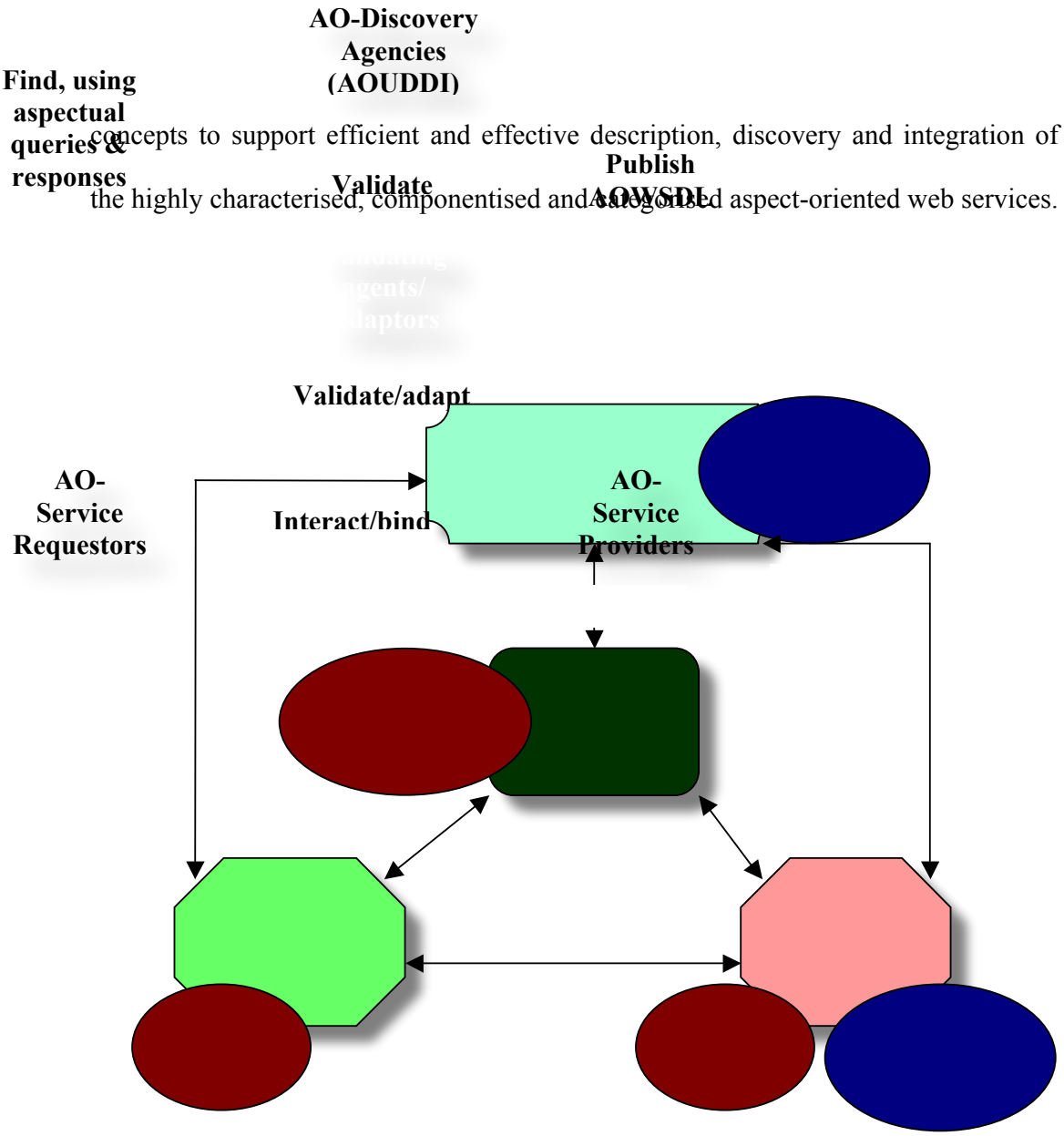


Figure 1.2: Aspect-oriented web services architecture (AO denotes Aspect-oriented)

Figure 1.2 shows how we augment the standard web services architecture to use aspects to design and characterise the web services systemic components, and to provide improved support for discovering, validating and integrating web service components at run-time. Here the whole web services system is designed and implemented using AOCE techniques, and the resulting subsystems produced including the providers, requesters, adaptors, testing and validating agents and discovery agencies are all composed of aspect-oriented components. The aspect-

oriented service providers expose aspect-oriented component functionalities that are richer and better characterised and categorised than in standard web services, through the use of Aspect-Oriented Service Description documents.

The interesting features of web service-based systems include the self-description of their web service components via Web Services Description Language (WSDL) (Cerami 02) and the ability to dynamically discover new web service components to integrate at run time via the Universal Discovery, Description and Integration (UDDI) registry (UDDIwebsite 05). We have developed the Aspect-Oriented Web Services Description Language (AOWSDL) to better describe our aspect-oriented web services because the existing Web Services Description Language (WSDL) is inadequate as it does not support the description of rich aspect-oriented features. As shown in Figure 1.2, we have designed and implemented an Aspect-Oriented Universal Discovery, Description and Integration (AOUDDI) agency to better discover and integrate the highly characterised and categorised aspect-oriented web services system and to interpret the AOWSDL document. Conventional UDDI agencies are unable to utilise the high-level aspectual information available within our AOWSDL documents.

Throughout this thesis we also use a running example to illustrate the application of our novel approach to the design and development of a prototype aspect-oriented web service-based system. This is a collaborative travel planner application that can be used to make comprehensive travel arrangements for holidays, work etc. by clients. We describe the design and implementation of this highly distributed travel planner application using AOWS concepts to explain our novel ideas more clearly. We also evaluate our approach's strengths and weaknesses, and identify key areas for future research regarding our approach based on our knowledge and experiences.

1.3 Thesis Overview

The structure of this thesis is as follows:-

- Chapter 1 provides an introduction to this thesis and its objectives. It explains the motivation behind this research and gives an overview of our approach of applying Aspect-Oriented Component Engineering for web services development. We also provide an overview of the contributions of our refereed full papers at international conferences based on the research and development carried out for this thesis.
- Chapter 2 discusses and analyses work that is related to web services development based on current Component Based Software Development (CBSD) methodologies. The advantages and disadvantages of these methodologies are discussed together with a detailed description of what web services are, how they work and the current approaches to developing them. We also discuss related work on Inversion of Control techniques, Multi-Agents and Formal Modelling that is central to our research.
- In Chapter 3, the concept of "aspects" prevalent in software systems is explained with reference to current aspect oriented programming techniques. Aspect-oriented Component Engineering is also discussed here. We explain how AOCE can be applied to address web services software development issues that cannot be solved using the currently available development methodologies.
- Chapter 4 details the Requirements Engineering of our prototype collaborative Travel Planner application that is based on aspect-oriented

Web Services system. This is developed using AOCE techniques and covers both functional and non-functional requirements.

- Chapter 5 illustrates and explains the Analysis, Designs and Architecture of our collaborative Travel Planner application using the AOCE methodology. We also describe how efficient components and subsystems are constructed using AOCE to address cross-cutting issues.
- Chapter 6 explains in detail the Aspect-Oriented Web Services Description Language (AOWSDL) document that is used to describe aspect-oriented web services. Locating and integrating these highly characterised and categorised aspect-oriented web services by using our Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI) registry is also explained here. We also describe the architectural and technical details of the AOUDDI tool that we have developed using AOCE
- In Chapter 7 we explain another novel, convenient yet vital interlinking aspect-oriented object that can be used to inter-connect and allow for the efficient and effective flow of information and instructions between the various AOWS subsystems that we were discussed in the previous chapters. This object called the AOConnector is based on a novel pattern of Inversion of Control to allow for clients to be engineered in a light-weight fashion and thus make them easy to design, develop and maintain.
- In Chapter 8 we describe an alternative implementation to the AOConnector approach. This incorporates the extensive use of multi-agents based on AI techniques and intelligent agents co-operating and negotiating with each other to dynamically execute tasks that enable

autonomous AOWS description, discovery, integration and subsequent consumption of the services.

- In Chapter 9, Alloy, a formal modelling language is used to model our AOWS abstractions and its subsystems. The Alloy Analyser tool is used to dynamically analyse and verify our AOWS abstractions and subsystems so as to prove that they are logically and mathematically correct.
- In Chapter 10 we describe the software tools that we have developed to support AOCE and AOWS development. These tools make the development cycle of designing and creating large and complex systems like AOWS easier to manage and control, thus increasing efficiency and effectiveness during the development process.
- Chapter 11 describes our experiences in using AOCE to design and develop aspect-oriented web services systems. It also includes testing and evaluation of the systems. The results produced are analysed and discussed here.
- Chapter 12 summarises the contributions of this thesis and gives the conclusions. It also proposes future work in the AOCE research field related to AOWS.

1.4 Contributions of our Refereed International Publications

The titles and the summarised novel contributions of four refereed international publications based on the research carried out in this thesis are as follows:-

1. **"An Approach to Developing Web Services with Aspect-oriented Component Engineering" NCWS 2003 (Grundy et al 03):** In this paper, we described our initial research into applying AOCE to web service development. This included categorising web service operations into components and characterising the cross-cutting functional and non-functional aspects of these components for web services. We also discussed the pros and cons of using AOCE for this purpose and our experiences with this novel approach.
2. **"Developing .NET Web Service-based Applications with Aspect-Oriented Component Engineering" AWSA 2004 (Singh et al 04):** We provided an in-depth description and example of how AOCE can be used to design and implement Web Service-based systems using .NET technology. This includes discussions of aspect-oriented enhancements to the Web Services Description Language and the Universal Discovery, Description and Integration standards. We also described a prototype web services system we designed and implemented using AOCE to demonstrate and evaluate our techniques.
3. **"An Architecture for Developing Aspect-Oriented Web Services" ECOWS 2005 (Singh et al 05):** In this paper we presented a novel software architecture called aspect-oriented web services (AOWS) to address problems associated with the many limitations in current web services systems, especially with their description, discovery and integration mechanisms. We introduced a new and novel reusable subsystem called an aspect-oriented connector to support light-

weight clients and also enable richer dynamic discovery and seamless integration. Besides describing our novel new architecture, we also gave a formal specification of it using a formal modelling language called ALLOY and an implementation of it using .NET web services technology.

4. **"Deploying Multi-Agents for Intelligent Aspect-Oriented Web Services" PRIMA05 (Singh et al 05):** In this paper we presented a novel software architecture called intelligent aspect-oriented web services (IAOWS) that use a combination of Aspect-Oriented Multi-Agents, AI techniques, and aspectual service descriptors for aspect-oriented web services to cater for more complete and thorough descriptions of services. This supports more autonomous discovery of both services and components, and dynamic integration and consumption by clients. We also describe an initial implementation to engineer and deploy Multi-Agents and capture the rich cross-cutting aspects together with their behavior and interaction within this novel highly distributed system.

2 Related Work

Component-based software development (CBSD) methodologies have been used in the analysis, design and development of a variety of software applications, including some very highly complex systems (Kvale et al 05, Vitharana et al 03, Allen and Frost 98). In this chapter, we discuss what CBSDs are, the reasons why they are used, and examine some current CBSD methodologies used either in industry or for academic research, that can be applied to web services development of any complexity. We also explain what constitutes software components and describe how components are produced and utilised in these methodologies. We further discuss the strengths and weaknesses of these methodologies. A detailed description of what web services are and what makes them so appealing, how they work and the current approaches to developing them is also included here.

2.1 Software Components

Software systems have evolved to become even more complex, larger, more difficult to understand and harder to control than ever. They have also become harder to refactor, reuse, maintain and scale. This has resulted in high development costs, unmanageable software quality, low productivity and higher risks when migrating to newer technologies. As a result there is an urgent need for better and more efficient software development methodologies (Grundy 00, Lee et al 05). Some of the best and most successful solutions available today to address these problems are the application of component-based software development (CBSD) methodologies

because they modularise software in a more manageable, effective and understandable manner and promote its reuse (Ran 03, Mei 04).

Components are already well established and widely used in all other engineering disciplines. In recent years, software engineers and developers have started to apply the idea of using components in software design and development.

A software component generally has three main features:
(a.) It constitutes an independent and replaceable part of a software system that has a clear function to fulfil.
(b.) It works within the context of well defined software architecture.
(c.) It communicates with other components through its interface definitions (Brown and Wallnau 98).

Several component-based software development methodologies have also evolved as a result of this. These include the TopCoder ™ (TopCoder 05), The Architecture Based Component Composition Approach (Mei 04), The COMO approach (Lee et al 99), The Select Perspective™ (Allen and Frost 98) and The Catalysis™ (D'Souza et al 99) approach. Others like OMG's Model Driven Architecture (Siegel 02) have an extensive collection of tools that can generate interface definitions and application code that is vital for components. Software components have extensively been used in technologies applied to distributed computing systems. These include the use of CORBA, RMI and DCOM. Both Java's J2EE and Microsoft's .NET platform support component based architecture for web services systems.

The main motivation behind component-based software development is to increase the reusability and portability of software pieces and make the whole development process more efficient and understandable through the use of components (Grundy et al 98, Heisell et al 02). Here well defined software components are designed, constructed and assembled together as building blocks to develop the respective, more modularised software application. The following subsection explains the characteristics common to Component Based Software Development methodologies.

2.2 Characteristics of Components Based Software Development methodologies

Component-based software development (CBSD) methodologies can be recognized from their basic underlying activities that are common to each other. The four major activities that characterize component-based software development as mentioned in (Haines et al 97) and adapted from (Brown and Wallnau 96) approaches are listed below:-

1.) component qualification (sometimes referred to as suitability testing),
2.) component adaptation,
3.) assembling components into systems, and
4.) system evolution.

Figure 2.1 below depicts the 4 stages involved in component-based software development approaches. The first stage of CBSD called component qualification is a process whereby previously developed relevant components are ascertained and selected with the view to apply and reuse them in the new software system to be built.

This involves looking for relevant components from other software systems developed using CBSD or getting Commercial Off-The-Shelf (COTS) components (Haines et al 97, Kvale et al 05). This can be a tedious and time consuming process, especially if the documentation is missing, wrong or not clearly prepared.

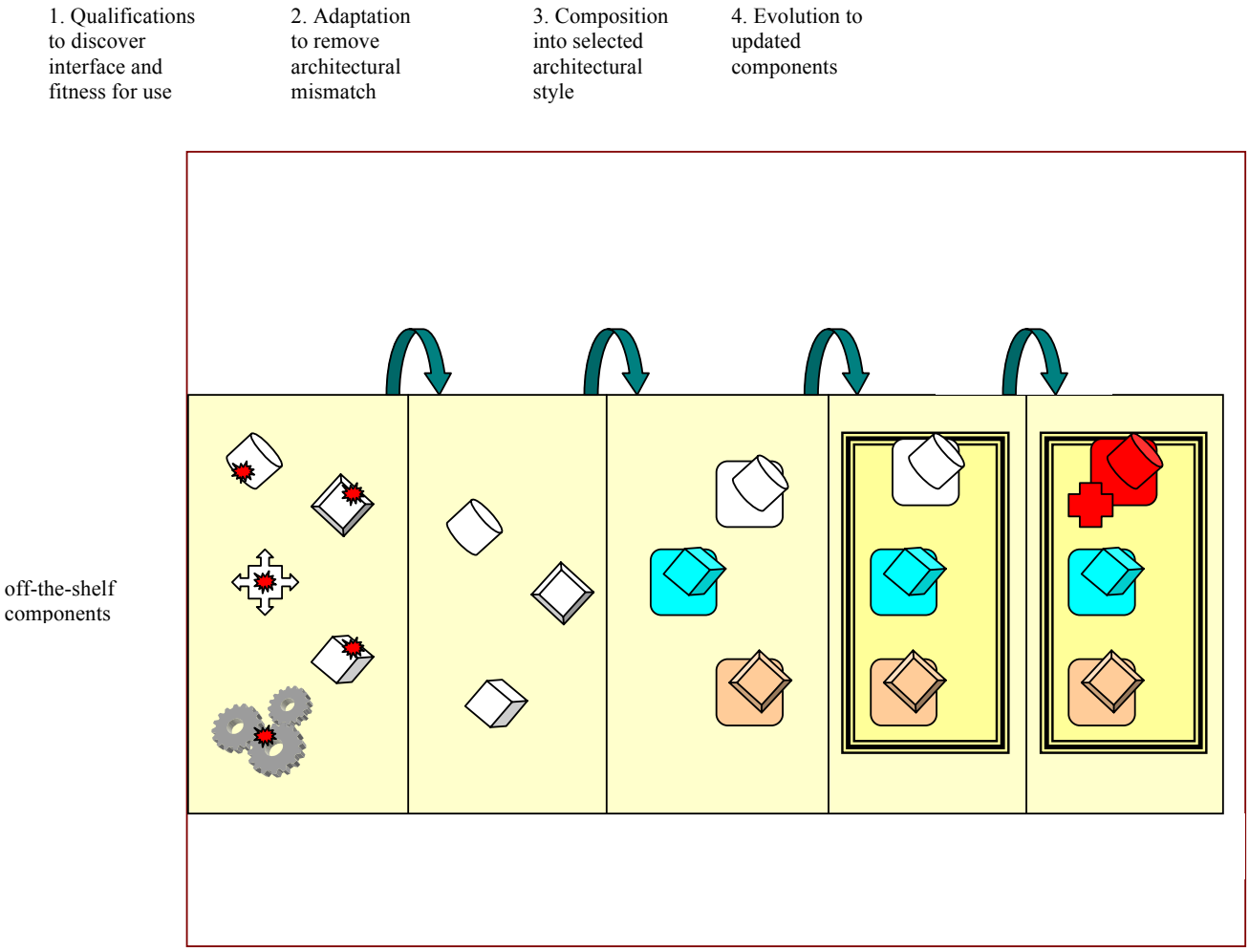


Figure 2.1: The 4 activities involved in Component-Based Development

Component adaptation is carried out in the second stage. Here components are adapted so that they can be made compatible and subsequently integrate with each other. This component adaptation process is necessary because the components were initially written to meet different requirements. They are also based on differing assumptions about their context. Components therefore often need to be adapted when applied in a new system. This is a very important stage and requires the skills of very good software developers. Very rarely do we find components that satisfy the software requirements and compatibility tests completely.

The third stage involves assembling the components into the software system. Some well-defined infrastructure is used to integrate the components that were adapted from the previous stage. This infrastructure provides the binding that forms a system from the various components. The components communicate through well defined and clear interfaces. These interfaces serve as binding contracts for the components to follow and conform to in order to use each others functionality.

The last stage which is called the system evolution stage is essentially used to correct errors and update components by swapping the defective ones with better ones. Also when functionality that is missing is required, a new component is created with this functionality and plugged into the system. As such, this stage basically does the final touch ups by further modifying and updating a system that is almost fully functional.

2.3 Current Component Based Software Development methodologies

Many Component-Based Software Development (CBSD) methodologies have been developed and tested, these include the TopCoder ™, The COMO approach, The Select Perspective™ , The Catalysis™, OMG's Model Driven Architecture and the Architecture Based Component Composition Approach (ABC) approach that were mentioned earlier. The reason we chose these six methodologies are that they are a good representation of existing CBSDs and collectively contain all the concepts and techniques used in current component based software development methodologies, including having good techniques and ideas that are used in all CBSDs. They are a good sample of the many different types of the methodologies that can be used to represent the current CBSDs used to develop the software for component based web services systems. We will discuss these Component-Based Software Development approaches paying attention to their strengths and weaknesses.

2.3.1 The ABC (Architecture Based Component Composition) Approach

In this Architecture Based Component Composition Approach (ABC) (Mei 04), it is proposed to use Software Architectures (SA) to compose prefabricated components to solve the key issue of component-based reuse. The downside is that SA provides a top-down approach to realizing component-based reuse, but doesn't pay enough attention to the refinement and implementation of the architectural descriptions, thus

it is not fully able to automate the transformation or composition to form an executable application.

ABC uses SA to play a central role in the whole software lifecycle, that is, SA description is used as the blueprint and middleware technology as the runtime scaffold for component composition, maintenance and evolution. It introduces software architectures into each phase of software life cycle, takes SA as the blueprint of system development, shortens the distance between high-level design and implementation by supporting tools and mapping mechanisms, realizes the automated system composition and deployment on runtime component operating platforms, and makes architecture available at runtime for software maintenance and evolution. Major issues in ABC include architecture-oriented requirement analysis, architecture design, architecture based composition, architecture based deployment, architecture based maintenance and evolution of software systems.

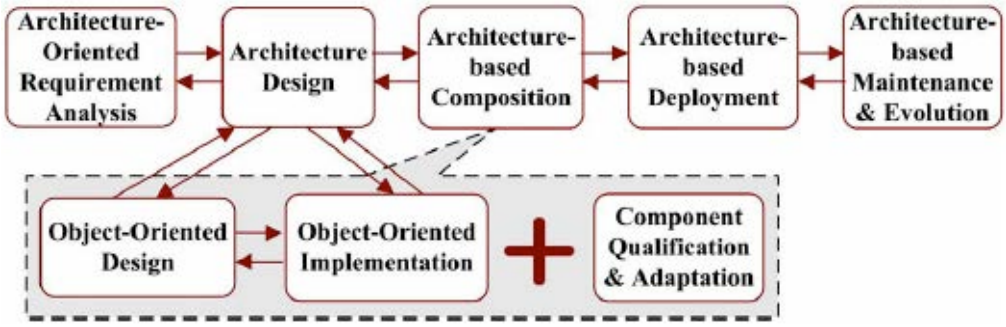


Fig.2.2 The Process Model of ABC

The process model of ABC is shown in Fig 2.2. To achieve the traceability and consistency between requirement specifications and system design, ABC introduce concepts and principles of software architecture into requirements analysis and specifications. In this phase, there is no actual SA but only the requirement specifications of the system to be developed, which are structured in the way similar to SA. It consists of a set of component specifications, connector specifications and

constraint specifications and will be used as the basis for software architecting. In the phase of software architecting, the requirements specifications are refined, and some overall design decisions are made. To produce SA meeting functional and non-functional requirements of the target system, the architects may study the requirement specifications, refine components and connectors in the problem space, create necessary artificial components and connectors, produce dynamic and static SA models, build mapping relationships between requirement specifications and SA, check SA and so on. In the phase of component composition, the components, connectors and constraints in the reusable assets repository will be selected, qualified and adapted to implement the logic entities produced in architecting. However, there are still some elements unable to be implemented by reusable assets. These elements have to be implemented by hand in object-oriented languages or other ones. Being implemented and tested, the elements will be stored into the repository and then composed into the target system as reusable assets. In that sense, the design view of SA can be fully implemented by the reusable assets. But ABC does not take into consideration the cross-cutting issues prevalent in its systemic components that give rise to tangling code (Kiczales 97, Lieberherr 99, Grundy 00, Grundy and Hosking 02) in the programs developed. It also tends to focus more on the lower-level features of the component/system. This can make the SA designs hard to understand at abstract levels or during code refactoring. Higher level systemic component descriptions such as persistency, user interfaces, security, transaction processing, performance etc. are all lacking. Such high-level features are important for understanding and using systemic components and their functionalities, especially in complex systems. ABC may sound simple but in effect it is a complicated CBSD methodology with many strict development rituals to abide by, and for other

developers who were not involved in the initial development of the particular software, this will be complicated and difficult to do as they will be trying to decipher a multitude of architectural diagrams and designs obtained from different phases that are not exactly helpful for reuse, refactoring, maintenance or scaling purposes.

2.3.2 The TopCoder ™ Methodology

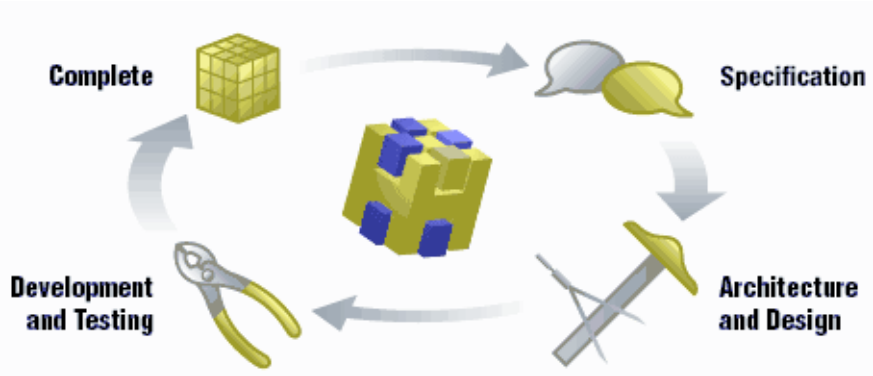


Figure 2.3: The four stages in the TopCoder ™ Component-Based Development as illustrated in (TopCoder 05).

TopCoder ™ (TopCoder 05) is Component-Based Software Engineering methodology that is used to commercially develop software in industry. It is amongst the most comprehensive and practical Component-Based Software Development (CBSD) methodology and is made up of 4 phases. Figure 2.3 above shows these four phases/stages in TopCoder ™, i.e. these are its Specification; Architecture/Design; Development/Testing; and Completion/Certification stages. The analysis, design, development and release of each and every software component must go through each of these four phases, through multiple iterations if necessary. The component is also made to undergo and pass rigorous testing before it can be certified as fit for use. We will discuss each of the four Topcoder CBSD stages below:-

Specification stage

During this stage TopCoder Project Managers (PM) moderate the TopCoder Customer Forums to gather requirements about a new Component Project. Once the project scope has been determined, the PM creates a Requirements Specification for the Design Phase of the project. New projects are regularly posted to the TopCoder Software Development site and emails are sent out to members notifying them of the new projects.

Architecture and Design stage

Here, the PM create an Architecture Review Board made up of three TopCoder members who regularly submit designs for TopCoder Software Projects. Submitted designs are scored using a standardized scorecard. The design with the best score above the TopCoder Software Minimum Score is chosen for the project. The winning designer is given additional time to incorporate suggestions from the Review Board. If the time frame is not met, the designer will be disqualified and the next design in order of score will be declared the winner. Once a winning design is determined, the PM posts the Development Phase of the project on the TopCoder Software Development site.

Development and Testing stage

Continuing from the earlier two stages, the PM creates a Development Review Board made up of three TopCoder members who regularly submit development proposals for TopCoder Software Projects. Submitted development solutions are scored using a standardized scorecard. The development submission with the best score above the TopCoder Software Miniumum Score is chosen for the project. The winning

developer is given additional time to incorporate suggestions from the Review Board. If the time frame is not met, the developer is disqualified and the next design in order of score will be declared the winner.

Completed and Certification stage

During this stage a complete TopCoder Software Component is added to a Component Catalogue and is ready for download by subscribers. It has been thoroughly tested to ensure acceptable performance, accuracy of results, and ability to handle bad data and incorrect usage. Extensive documentation is downloaded along with the component. Customers, the PM and other TopCoder members interact in the Customer Forums to discuss the current complete version of the component, as well as make suggestions for future versions of the component.

To release each component, it must go through each of these four phases and if any phase fails an acceptance test, the phase is restarted. This methodology is not only tedious, it also focuses on lower-level features of the component/system. This can make its designs and implementations hard to understand at higher abstract levels or during code revisits and refactoring. Higher level systemic component descriptions such as persistency, user interfaces, security, transaction processing, performance etc. are all lacking. Such high-level features are important for understanding and using systemic components and their functionalities, especially in complex systems like web service. In web services these features and information can also be used gainfully in service documents to achieve better description, and more autonomous discovery and integration.

2.3.3 The COMO Approach

This is a component development methodology that is based on practical object-oriented techniques for software development. Though it's not a relatively new development methodology, COMO can be used to design and develop web services systems. This approach defines concrete guidelines for modelling workflow between tasks and focuses only on component development. It extends currently used UML notations to incorporate a component model and also includes message flows and classes into their component diagrams. These message flows between components are further mapped into interfaces in the component diagrams. The reliability of the component modelling is also taken into account in the COMO approach (Lee et al 99). This methodology utilises two clustering techniques called the "Use Case Clustering" and "Use Case cum Class Clustering" techniques to design and develop software components based on the requirements engineering of the software system to be built. The clustering techniques aid in identifying and determining the operations that will be supported in the interfaces and implementations of the software components.

The COMO approach and its clustering techniques can be used to develop complex web services systems but its techniques only focus on functional requirements when identifying the main requirement sets. Non functional issues are not addressed in the COMO approach. It only focuses on lower level functional decompositions of components. These functional decompositions vertically slice the software systems based on clustering techniques. Cross-cutting issues such as persistency, distribution, performance, and transaction processing are not addressed. These issues cross-cut

through the systemic components and give rise to tangling code in the programs developed (Kiczales 97, Lieberherr 99, Grundy 00, Grundy and Hosking 02). The COMO approach neither describes any of these systemic high level features nor any of their non-functional properties. It does not address the cross-cutting problems that exist in the software. As such, the absence of a lot of very important and rich aspectual features of components limits the reuse and understandability of its components.

2.3.4 The Select Perspective™ Approach

The Select Perspective™ (Allen and Frost 98) approach is another component-based approach that incorporates the developments in the Unified Modelling Language (UML) and extensions to the UML to develop enterprise systems. An enterprise system is defined as one that can handle complex business processes necessary to manage and operate large businesses. These type of systems need to be delivered promptly and according to the specifications required, with resort made to the identification and reuse of existing components to enable this to be achieved within the allowable time frame.

The UML notation used in this methodology has also been streamlined and enhanced, albeit minimally to cater for enterprise systems. The Select Perspective incorporates six core models based on the Unified Modelling Language (UML) to achieve its development goals. These models are the Use Case Model, Class Model, Object Iteration Model, State Model, Component Model and Deployment Model. They are used in conjunction with two more models called the Business Process Model and the Logical Data Model which are not typically found in software modelling techniques. All these modelling ideas and their particular techniques can be applied to both the development of reusable components and for specific solutions.

The Select Perspective™ methodology can be used to develop web services systems but again it mainly focuses on the functional properties and services of a software system. But only low-level systemic characteristics and services are captured in this

approach. Also, it just focuses on functional perspectives. As such the components designed do not contain high-level systemic aspect information for component aspects. This means that high-level aspectual features, for instance for persistency, security, performance, distribution and transaction processing services for components are not defined. The non-availability of these high-level aspectual descriptors in the components makes it difficult to understand the components produced using this methodology because they are defined at a lower language. The complexity of this problem is further compounded because there are a multitude of models involved, each adding low-level and hard to understand descriptors of their own.

2.3.5 The Catalysis™ Approach

The Catalysis™ approach (D'Souza et al 99) incorporates the use of the Unified Modelling Language (UML) to model software systems based on objects and components. This approach mainly focuses on lower-level component functional decomposition and their interfaces. It is based on ideas borrowed from other technologies and can be applied to the development of Component-based systems, high integrity designs and also towards the reengineering of existing software.

The Catalysis™ UML models used are described in a number of specific ways. Here actions are described in terms of their effects on objects and can be illustrated with snapshots and defined with post conditions. Abstract specifications can be made very precisely, thus avoiding ambiguities. Actions, collaboration-schemes and objects can be abstracted, refined and their relationship traced all the way from business goals to

program code. Furthermore the components and objects, which are also similarly designed as above, can be presented using different views. Like all components, the Catalysis™ components can also be designed to plug into each other or into the software framework.

Though it is quite a detailed CBSD technology and can be applied to design and develop web services-based systems, the disadvantages are that all the techniques used in the Catalysis™ approach tend to focus on lower-level features. This can make component-based designs very hard to understand at higher levels. Identifying explicitly higher level systemic component descriptions such as persistency, user interfaces, security, transaction processing, performance etc. are missing from the Catalysis™ approach. These high-level features are very important for understanding systemic components and their functionalities.

2.3.6 OMG's Model Driven Architecture for Web Service Development

The Object Management Group (OMG) has its own methodology for developing complex web services systems. They call their approach the Model Driven Architecture (MDA) (Siegal 02). It has a comprehensive set of MDA tools that can be used to generate interfaces definitions and application code to construct components. MDA was formulated with the objective of simplifying the process of modeling, design, implementation, and integration of applications, including large and complex web services, by defining software fundamentally at the model level. This is expressed in OMG's standard Unified Modeling Language (UML).

The MDA base model of the application specifies every detail of its behavior and business functionality in a technology-neutral way. This is called the application's Platform-Independent Model. Using the Platform-Independent Model, MDA tools follow an OMG-standard mapping to generate an intermediate model. This is tailored to the target middleware implementation platform. This intermediate product produced is called a Platform-Specific Model (PSM). It reflects non-business, computing-related details e.g. those affecting performance and resource utilization that are added to the Platform-Independent Model by the web services' architects. The PSM generated from the MDA tool may not be perfect and may require some manual "tune-up" before it can be used for the next stage. However, the PSM is extremely detailed and contains the same information as a fully-coded application but in the form of a UML model instead of as code.

In the final development phase, MDA tools use the PSM to generate interface definitions and application code, makefiles, and configuration files for components and the PSM's middleware platform. The components produced can also be reconfigured and refactored manually to produce the desired results.

Though it is a development methodology that can be applied to large scale development of web services systems, the disadvantages are that all the techniques used in this approach again tend to focus on the lower-level component type and behaviour features of the system. This can be very hard and complicated to understand at higher levels or during reengineering/refactoring processes. The higher level systemic cross-cutting issues such as persistency, user interfaces, security, transaction processing, performance etc. are all missing from this approach. It does

not address the cross-cutting issues that exist in the software. These issues also cross-cut through the systemic components and give rise to tangling code within the software (Kiczales 97, Lieberherr 99, Grundy 00 and Grundy and Hosking 02). OMG's Model Driven Architecture does not describe any of these systemic high level features or any of their non-functional properties. These high-level features are very important for understanding systemic components and their functionalities, especially in the context of large and complex systems. These features are also essential and very crucial for more efficient and effective description, discovery and integration of web services because they rely heavily on clear and precise definitions of the web services advertised.

2.4 Web Services and its Service Oriented Architecture

Web services can bring additional functionalities and information into our applications from the intranet or internet in much the same way that browsers can make information available to end users. Web services promise to enable business-to-business integration seamlessly and dynamically irrespective of platform, language or culture (Torkelson et al 02, Cerami 02). The W3C document on web services defines it as a software system that is identified by a Uniform Resource Indicator (URI) whose public interfaces and bindings are defined and described using the eXtensible Mark-up Language (XML) (Booth et al, 04). Its definition can be discovered by other software systems that can interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocol.

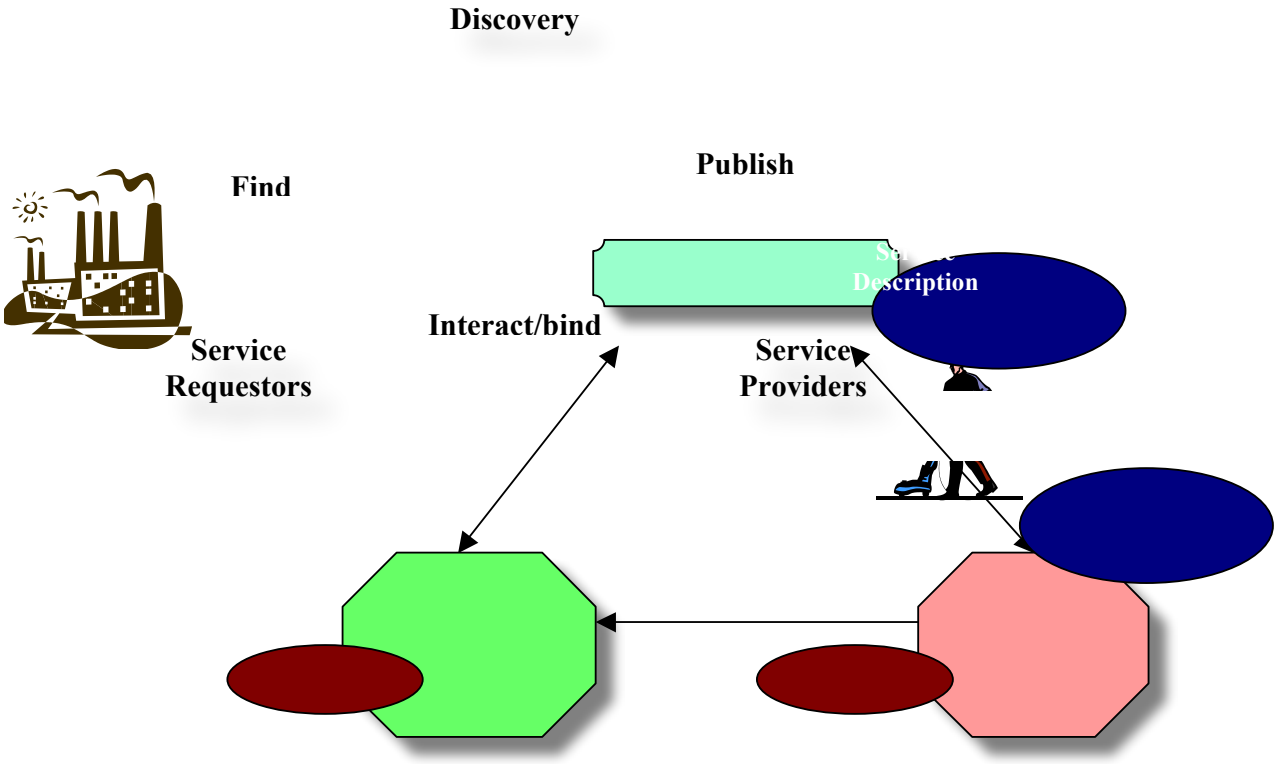


Figure 2.4: Generic web services architecture

The purpose of a web service as such is to provide some functionality on behalf of its owner which can be a person or organization, such as a business or an individual. Web services provide a Remote Procedure Call (RPC) interface as explained in (Pallman 05) for client applications, the owners of which can again be a person or organisation, to call class methods on the server side. A generic web services architecture (Booth et al 04) comprising web service providers, web service requestors, their respective owners and discovery agencies is depicted in figure 2.4. The web service providers publish their services which includes their description, location, exposed APIs and technical details relating to their discovery, integration and consumption on discovery agencies.

The Universal Description, Discovery and Integration (UDDI) registry, as stated by (Newcomer 02), is an example of a web services discovery agency that can be used by

web service providers to publish details about their location, services and related technical details. The UDDI registry is a business and service registry which is also an open industry initiative to enable businesses utilising web services technology to describe, discover and integrate with each other as mentioned in (Ran 03 and Adams and Boeyen 02). Web service providers further make use of the Web Services Description Language (WSDL) (Cerami 02) as the XML format to describe their services. It contains all the details necessary to interact with the service, including message formats, transport protocols and location (Christensen et al 01). The nature of this web services description is such that it hides the implementation details of the service so that it can be used independently of the programming language, hardware or software platform on which it is implemented.

Web service requestors, i.e. clients, can then query the discovery agencies to discover the web service providers and find out about the services provided (Microsoft .NET 05 and Sun Microsystems 05). If the required service is found, the web service requestors can integrate with the web service providers and consume the services provided for its own use without going through the trouble of rewriting the code.

Web services are meant to expose functionality of a server to other applications as discussed in (Strahl 01). The client applications in this case may be a Fat Client Windows application; a Fat Server Web application that runs on a standard web backend such as Active Server Pages (ASP); a browser based client application using script code or even Java applet running in a browser on a UNIX machine. As long as a client application has support for the protocol used by the service provider, for instance the Simple Object Access protocol (SOAP) (Cerami 02) it can call the

remote Web Service and return data for it, assuming the user is authorised. The SOAP protocol is responsible for routing the RPC message from the client to the server and return the result back to the client application. SOAP is based on XML and follows a relatively simple design that is easy to implement. SOAP's simple protocol has contributed to its wide support on just about any platform and development environment (Chappell and Jewell 02).

The syntax of SOAP messages are that they must be encoded using XML and must use a SOAP Envelope and SOAP Encoding namespace, but they must neither contain a DTD reference nor any XML Processing Instructions (w3schools_soap_example website 05). A Body element within the Envelope contains the request or response information. An optional SOAP Header element that contains application specific information (like authentication) about the SOAP message, if present, must be the first child element of the Envelope. To illustrate these, figure 2.5 below shows an example of a SOAP request made to obtain the quote for an item from a remote server. The request is "Persistency_GetToyPrice" and it is enclosed within the Body element of the root SOAP Envelope element as shown. This request has a ToyName as its parameter and it will invoke the "Persistency_GetToyPrice' operation on the server to return the price of the toy specified. The namespace for the function is defined in "http://www.toy.org/toy" address. Figure 2.6 below shows the response with the Price parameter showing the value of the toy queried.

```
POST /InToy HTTP/1.1
Host: www.toy.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.toy.org/toy">
    <m:Persistency_GetToyPrice>
      <m:ToyName>AIBO</m:ToyName>
    </m: Persistency_GetToyPrice>
  </soap:Body>
</soap:Envelope>
```

Figure 2.5: Example of a SOAP Request

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.toy.org/toy">
    <m: Persistency_GetToyPriceResponse>
      <m:Price>200.5</m:Price>
    </m: Persistency_GetToyPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Figure 2.6: Example of a SOAP Response

As the SOAP protocol is very comprehensive and yet very versatile and flexible, there exist SOAP clients for practically every development environment, including clients for COM, .NET, Perl, Java, C++ and PHP (Cerami 02, Microsoft .NET 05 and Sun Microsoft 05). These protocols should identify its agreed Message Exchange Pattern (MEP) set. As explained in (Booth et al 04), a MEP is actually a template that is devoid of application semantics and describes a generic pattern for the exchange of messages between parties is used to describe the relationships (e.g., temporal, causal,

sequential, etc.) of multiple messages exchanged in conformance with the pattern. MEP is also used to describe the normal (and abnormal) termination of any message exchange conforming to the pattern during the transactions between the providers and requesters. SOAP implementations provided by vendors, which has its MEPs described in its specifications, typically consist of two pieces, i.e. a client side proxy that handles the SOAP message creation and a server side piece that implements the Web Service logic. Proxies, also termed Web References, are the key to make Web Services functional, easy to use and consume because they can be dynamically generated (and dynamically destroyed if necessary), reside locally and represent the objects/systems residing on remote machines that communicate over the wire using XML as mentioned in (Strahl 01).

We can further use the Business Transaction Protocol (BTP) which is an XML-based protocol that serves as a standardized Internet-based means of managing complex and ongoing business-to-business (B2B) transactions between multiple organizations (Dani et al 05). It is being developed by the Organization for the Advancement of Structured Information Standards (OASIS). BTP is intended to be especially useful in a web services environment and is attractive because it can be layered over transport technologies, including the Simple Object Access Protocol (SOAP).

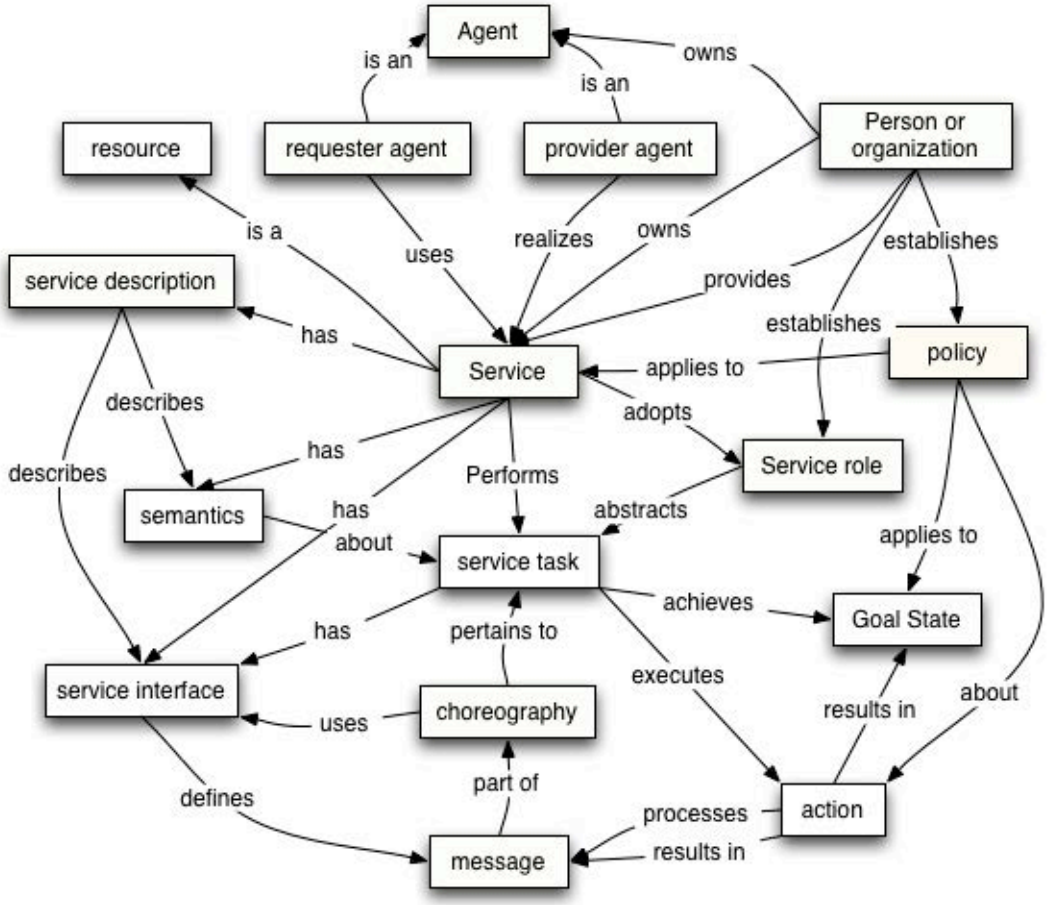


Figure 2.7: Service Oriented Model as depicted in (Booth et al 04)

Web services are also based on the Service Oriented Model (SOM) as shown in Figure 2.7 above that was extracted from (Booth et al 04) above as proposed by world bodies, in this case by W3C's Web Services Architecture Working Group. SOM focuses on those aspects of the architecture that relate to service and action, it depicts the relationships between agents and the services they provide or request. In this model, an agent is a concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided. To illustrate this distinction using our prototype Travel Planner application, we might implement a particular web service, e.g. Flights web service, in our application using one particular agent written in a particular

programming language, e.g. Java, and a different agent the next time (perhaps written in a different programming language, e.g. C#) with similar functionality. As can be seen, although the agent may have changed, the web service remained the same. As shown in the figure above, each agent may realize one or more services or may request one or more services.

An action, for the purposes of this Service Oriented Architecture (SOA), is any action that may be performed by an agent, possibly as a result of receiving a message, or which results in sending a message or another observable state change. Example of an action is a requester choosing the optimum service provider to integrate with from a list returned by the UDDI containing a number of different web service providers that provide similar services and their ranking based on matched criteria.

At the core of the concept of service is the notion of one party performing action(s) at the request of another party. From the perspective of requester and provider agents, an action is typically performed by executing some part of a program.

Software agents are also proxies for the entities that own them. This is important as many services involve the use of resources which also have owners with a definite interest in their disposition. For example, services may involve the transfer of money for payment to purchase train tickets and the incurring of legal obligations as a result.

A choreography is a model of the sequence of operations, states, and conditions that control the interactions involved in the participating services. The interaction prescribed by a choreography results in the completion of some useful function. Scenarios from our Travel Planner include the request and confirmation of a flight ticket to a particular destination at a certain time, continuing with information about

its payment and delivery, or putting the system into a well-defined error state if the request is unsupported.

A choreography can be distinguished from an orchestration. An orchestration defines the sequence and conditions in which one web service invokes other web services in order to realize some useful function.

A choreography may be described using a choreography description language. A choreography description language permits the description of how Web services can be composed, how service roles and associations in Web services can be established, and how the state, if any, of composed services is to be managed.

A service description is a set of documents that describe the interface to and semantics of a service. A service description contains the details of the interface and, potentially, the expected behavior of the service. This includes its data types, operations, transport protocol information, and address. It could also include categorization and other metadata to facilitate discovery and utilization. The complete description may be realized as a set of XML description documents.

There are many potential uses of service descriptions: they may be used to facilitate the construction and deployment of services, they may be used by people to locate appropriate services, and they may be used by requester agents to automatically discover appropriate provider agents in those case where requester agents are able to make suitable choices.

A service interface is the abstract boundary that a service exposes. It defines the types of messages and the message exchange patterns that are involved in interacting with the service, together with any conditions implied by those messages.

The semantics of a service is the behavior expected when interacting with the service. The semantics expresses a contract (not necessarily a legal contract) between the provider entity and the requester entity. It expresses the intended real-world effect of invoking the service. A service semantics may be formally described in a machine readable form, identified but not formally defined, or informally defined via an "out of band" agreement between the provider entity and the requester entity.

Knowing the type of a data structure is not enough to understand the intent and meaning behind its use. For example, methods to make payments and receive reimbursements during a financial transaction typically have the same type signature, but with a different effect. The effects of the operations are the semantics of the operation. It is good practice to be explicit about the intended effects of using a web service; perhaps even to the point of constructing a machine readable description of the semantics of a service. Machine processable semantic descriptions provide the potential for sophisticated and autonomous usage of web services. For example, by accessing such descriptions, a requester agent may autonomously choose which provider agent to use based on a list of available choices of similar service providers.

Apart from the expected behavior of a service, other semantic aspects of a service include any policy restrictions on the service, the relationship between the provider entity and the requester entity, and what manageability features are associated with the service. In the rest of this thesis, we will refer to a provider and requester to mean the service provider agent and requester agent respectively.

The currently used component-based systems engineering methodologies for web service-based systems, including the TopCoder ™, The Architecture Based Component Composition Approach, The COMO approach, The Select Perspective™, The Catalysis™ approach and OMG's Model Driven Architecture tend to focus on low level software component interface design and implementation. This has a great disadvantage in that it often results in development of components whose services are both difficult to understand and hard to combine. The current development approaches also make too many assumptions about other components related to a particular web service system, constraining their reuse. Furthermore the component documentation is often too low level which again is hard to understand at higher levels. Maintenance and refactoring of these software systems can be very difficult. Precise description and documentation for web services, especially in service documents, is also extremely important so that prospective clients wanting to consume the web services can locate and understand the descriptions more accurately. As web services are still a relatively new and maturing technology, there are still a lot of unanswered issues about web services design and implementation, including those relating to security, performance, collaboration and interoperability as discussed in (Cerami 02, Hung 04 and Lee 04). More accurate, consistent and coherent high level descriptions will also allow for dynamic searches and integration to be done more effectively and efficiently.

All the current CBSD approaches discussed above cannot produce service documents containing elements identifying and describing the high level cross-cutting features inherent in web services because they do not address such issues. These high level features and their properties are also very useful for enabling better understanding to

achieve dynamic discovery, integration and consumption of web services. As such we need a better CBSD approach that can be used to identify and address such issues and further incorporate them into service documents and develop systems capable of effectively and efficiently locating, interpreting and using them. We propose using a CBSD methodology called Aspect-Oriented Component Engineering (AOCE), described in the next chapter, to develop novel Aspect-Oriented Web Service based systems (AOWS). In the next section we will discuss a novel technique called Inversion of Control (Fowler 04) and related work in this area of research. We also give a description of the use of Inversion of Control in a connecting subsystem in AOWS, called an AOConnector. The AOConnector make clients more lightweight and was developed using AOCE to address the issues discussed above that cannot be addressed using current CBSD methodologies.

2.5 AOConnector and Inversion of Control

In Chapter 1 (with reference to Figure 1.2), we introduced our approach of using Aspect-Oriented Component Engineering (AOCE) to develop Aspect-Oriented Web Services (AOWS) systems. In this subsection we give an overview of a novel technique called Inversion of Control (Fowler 04) and related work in this area of research. We also give a brief description of its use in AOWS through a connecting object, called an AOConnector, (shown in figure 2.8 below). We had used our own version of IoC in the AOConnector object to make our clients more lightweight so that they are more understandable, maintainable, scalable, controllable and usable.

Inversion of Control (IoC) patterns are used to develop highly decoupled, mobile, lightweight and unit-testable software (Mathew 05). The main idea behind IoC is that an object exposes its dependencies via some form of contract, and other objects or subsystems can use this object to access and use functionality without actually knowing how this object accesses other classes to execute its operations or extract/manipulate data. Dependencies can be anything that an object needs to perform its designated function but is not concerned with its implementation, i.e., it may include the object's interface, system resources etc. In a nested object graph, each object in the call chain exposes its dependencies to the outer caller that uses it, which in turn exposes those dependencies including any of its own to its caller and so on, until all dependencies manifest itself at the top. The top-object then assembles the dependency graph before activating the objects. The top-object is generally an entry point into the software system, e.g. in our Travel Planner client, its main method that serves as the entry point.

But the currently used IoC techniques as employed in the Spring framework and Pico Containers and HiveMind (Spring_framework_homepage 05,

PicoContainer_homepage 05, HiveMind_webpage 05) have again mainly focused on the lower-level functional properties and services of a software system. Only low-level systemic characteristics and services have so far been captured using these approaches. As such the objects designed do not contain high-level systemic aspect information for component aspects. This means that high-level aspectual features, for instance for persistency, security, performance, distribution and transaction processing services for components are not defined. The non-availability of these high-level aspectual descriptors in the objects makes it difficult to understand the objects because they are defined at a lower language. The complexity of this problem is further compounded because there are a multitude of different cross-cutting concerns involved, each adding low-level and hard to understand dependencies of their own, thus making IoC more complicated and difficult to understand.

The AOConnector object, which also serves as an interlinking subsystem as shown in Figure 2.8 below, uses a new pattern of Inversion of Control that we modelled, designed and developed ourselves. We used IoC so that our AOWS system is more loosely coupled and efficient as a whole. This figure illustrates the inter-relationships between the client (requester) and the other subsystems (shown enclosed within the red circle) of AOWS via the AOConnector. In this system, the AOConnector object exposes its public functionalities via its interface and these act as its dependencies (Fowler 04).

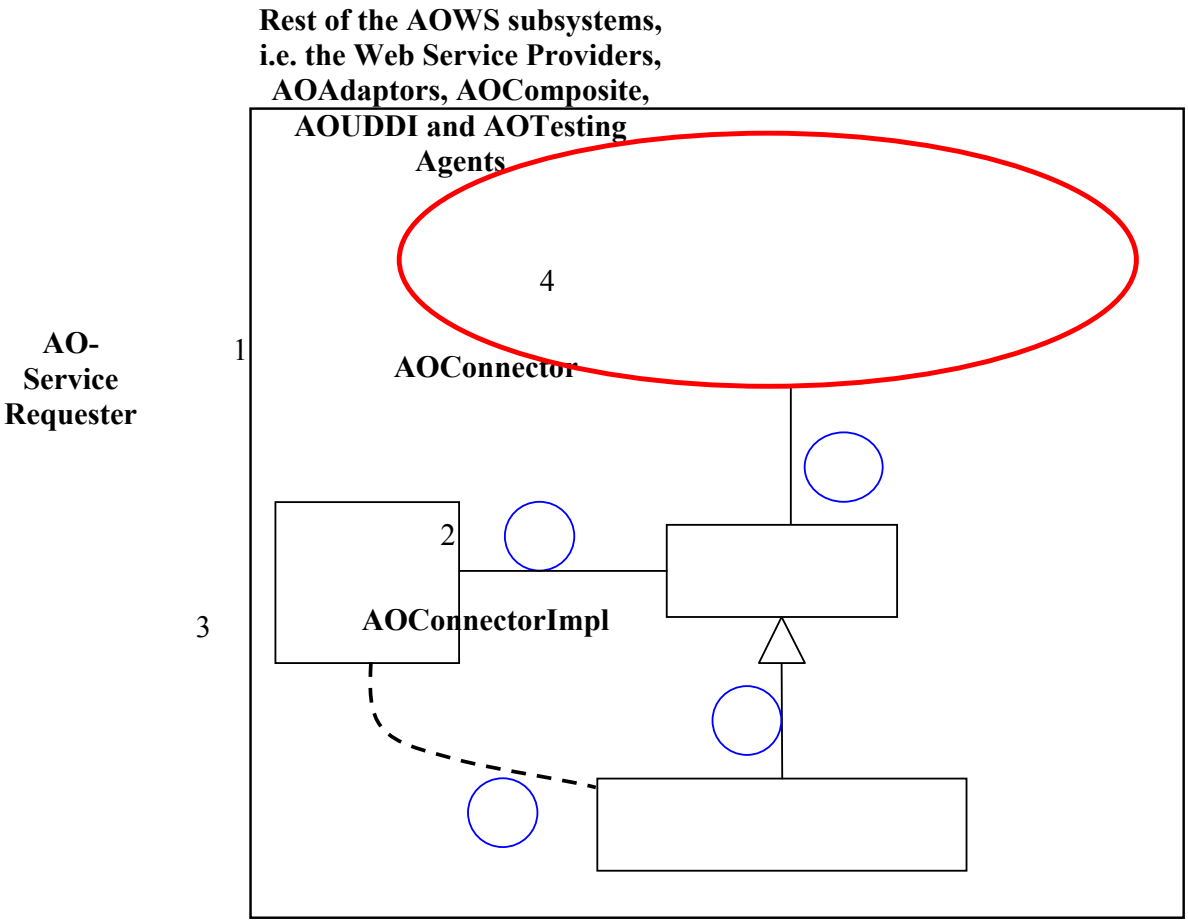


Figure 2.8: The AOConnector uses the Inversion of Control mechanisms

The client (service requester), as shown in the above figure, links as depicted through item numbered (1) to the AOConnector and utilises it to integrate or make queries, remote service calls etc. to the appropriate subsystem in AOWS. Through this IoC pattern, the client does not concern itself with which subsystem or service is called by the connector to execute its tasks. To illustrate an example, the client uses a method of the AOConnector called "TransactionProcessing_ExecuteMethod" and let the connector decide which service the connector has integrated with to use to serve the clients request best. The connector is implemented by the AOConnectorImpl concrete class that implements this method of its interface. The full signature of this method is given in Figure 2.9 below:

```
public                                          object
TransactionProcessing_ExecuteMethod(string
MethodName, object[] Param)
```

Figure 2.9: The TransactionProcessing_ExecuteMethod method's signature

It takes two parameters, the name of the method of the remote web service to be invoked which is a string object called "MethodName" and an array object of the parameters to be used as input parameters by the service. The AOConnector's interface is implemented in the AOConnectorImpl (2.) which is its derived class (sub-class) and it can be accessed by the client through its constructor etc. as shown by item numbered (3) in the figure. The connector object then looks up and interacts (4.) with the appropriate sub-system i.e. the AOUDDI, AOAdaptor, AOValidating agents or relevant service providers that in turn executes the method and returns an appropriate response to the connector that further processes it if necessary and then relays it to the requesting client.

If the particular method does not exists within the currently consumed web services, the connector will query the AOUDDI for a service that exposes such a method and, integrate with the service provider, consume the service to execute the method and return the results. If such a method does not exists even within all the services registered with the AOUDDI, then the AOConnector will transmit a message back to the requester that such a method does not exists and will suggest similar methods, for instance similar methods with less or more parameters, if such methods are available. As such the use of Inversion of Control in the AOConnector object allows for separation of business logic calls to be handled by the connector thus making the clients more light-weight, modular and easy to maintain and refactor. Clients do not

need to bother themselves with the server side logic and remoting connections and transactions; all these are handled by the connector object.

Concrete examples on how to use the AOConnector and its other supported methods are explained in Chapter 7. We also give an in-depth discussion on its analysis, design and implementation of the connector object in that chapter.

2.6 AOWS and Multi-Agents

As an alternative implementation to the AOConnector approach that was described above, we further researched and incorporated multi-agents into AOWS so as to support more autonomous discovery, integration and consumption of our web services systems. The agent-based AOWS is a novel extension to our thesis and a viable alternative to the AOConnector in its own right and is an architecture that incorporates extensive use of multi-agents based on AI techniques and intelligent agents co-operating and negotiating with each other to dynamically execute tasks that enable autonomous AOWS description, discovery, integration and subsequent consumption of the services. We call this novel software architecture Intelligent Aspect-Oriented Web Services or IAOWS to differentiate it from the one that does not use intelligent agents to negotiate and coordinate tasks within and between the various subsystems involved.

The characteristics of Multi-Agents (Sycara 98) within a system are that (i.) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (ii.) there is no system global control; (iii.) data are decentralized; and (iv.) computation is asynchronous. The objective of using Multi-Agents in web services systems is that they allow for more autonomous description,

discovery, integration and interactions based on the comprehensive use of AI techniques. So far, limited success has been achieved in the various areas of autonomous web service-based applications, including areas relating to facilitating automation in a particular part of the system like specific types of automated discovery for web services (Gannod and Bhatia 04), or composing whole intelligent web services systems (Heuvel and Maamar 03). These systems have many limitations because they have all overlooked and lack the capability to address high level issues (like aspects and highly characterised and categorised aspect-oriented components) that enable better categorisation and characterisation of the services advertised, nor do they have mechanisms that enable the discovery and integration with systems that support such high level descriptors/facilities or components. We propose the use of multi-agents that can interpret and use the high-level descriptors used in our autonomous systems. The agents themselves are composed of highly modular aspect-oriented components comprising independent units with each agent having clearly defined functions/tasks to individually or collectively negotiate and execute within the context of the other co-operating intelligent agents.

Multi-agents have been used in intelligent software systems to execute tasks through negotiation, cooperation and/or competition. But they have not been used in aspect-oriented software systems including aspect-oriented web service based systems. IAOWS uses the concept of multi-agents and aspects, in this case aspects that impact on different parts of not only the web services, but also the agents.

Figure 2.10 below shows an example from the prototype travel planner system that we refactored and restructured based on the IAOWS concept of incorporating multi-agents into our software system. As shown, by using discovery agents the client/requester looks-up various services from a registry (1). We deployed Discovery

Agents to coordinate with the client's Requesting Agents to search the repository of the AOUDDI, and return results best matching the web service descriptions requested for, including descriptions for their components, aspects, aspect details and provided/required aspectual features

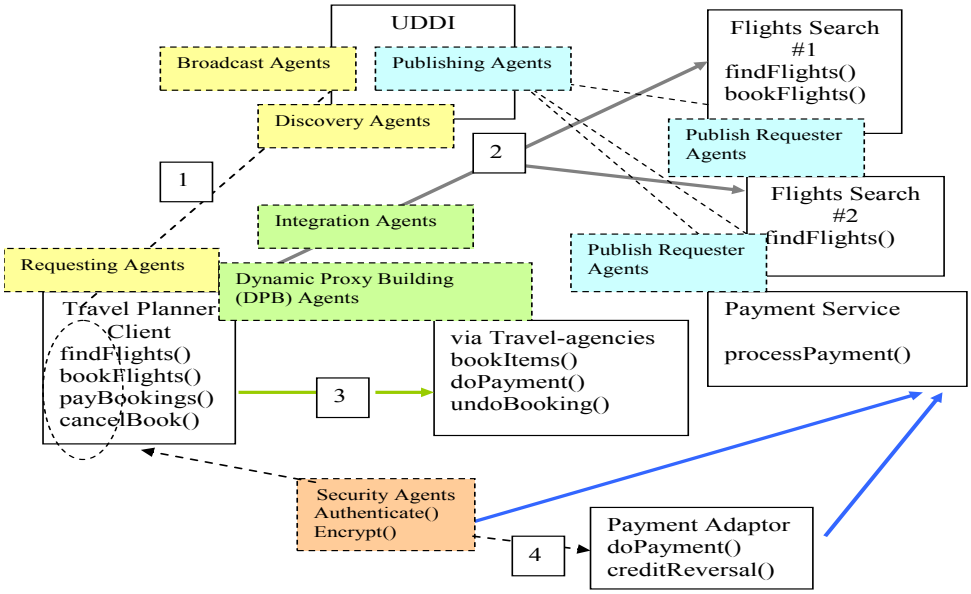


Figure 2.10. Example of web-service based travel planner utilizing multi-agents.

As shown in this example, flight searches for clients are performed by dynamically integrating with various discovered flight service providers (2) using the integration and proxy building agents. Bookings can be made directly or through agents (3), and if required payment subsequently made through a web-service based allowable mode of payment (4), this series of transactions can be verified by security agents. The Travel Agencies (3) are used as a back-up manual measure for those who do not have the time to search, book etc., and are more comfortable paying others to do these activities for them. Security issues handled by security agents may include a need for user authentication and data encryption/decryption. In specifying client needs and web services providing them, we need to specify these security requirements, and the relevant Multi-Agents will interact, coordinate and negotiate with each other to

produce an optimal solution. Aspectual constraints and their required/provided properties are used in testing and validating any discovered service.

To support better dynamic discovery, integration and subsequent consumption of services in web-service based systems, we designed and developed Intelligent Aspect-oriented Web Services (IAOWS) using Multi-Agents. This research further extends our AOCE work in which we developed extensions to the object component model to support component design, de-coupled implementation and run-time discovery and integration using component aspects (Dong et al 03, Coyler and Clement 04). Component aspects are cross-cutting concerns impacting on components, including persistency management, distribution, security, transaction processing and resource use. Components provide capabilities to others or require services from them across these different system aspects. Aspect details capture functional and non-functional properties and allow design-time reasoning and run-time component description and adaptation.

Key aspects that multi-agents use when discovering web services to interact with include security model, transaction management, performance measures for operations, and fault and exception-handling approaches. As such, when building web services we may describe data persistency approach, database transactional behaviour for operations, resource utilization, communications infrastructure, monitoring and logging, etc. During discovery and integration, we may need to locate adaptors, transaction managers, and security managers, and compose (or orchestrate) services. We aim to better support this range of activities when designing, implementing and deploying web services using IAOWS. We have developed a model of IAOWS-based systems, together with a variety of multi-agents, and proof-of-concept implementation of the model with .NET web services.

2.7 Alloy and Formal Modelling, Analysis and Validation

We also used Alloy (Jackson 02), a formal modelling language to model our AOWS system, so that we can use it to analyse and verify that the abstractions that we modelled are logically and mathematically correct. Alloy is a higher level parallel programming language appropriate for programming massively parallel computing systems. It is based on a combination of ideas taken from functional, object oriented and first-order logic programming languages (Language_list 05). It is similar to Z or OCL, the Object Language of UML, but Alloy is designed more for automated model analysis and verification compared to the other two languages. Alloy is specifically targeted at the creation of micro-models of complex software systems because they can then be automatically checked for correctness (Jackson MITLab, 02). The usefulness of Alloy is evident in the fact that it does not require the mechanism to describe the effect of a behaviour. This conveniently allows us to divide our aspect-oriented web services model, which is really one very large model, into several smaller and simpler models, which will be much easier to analyze and verify.

Alloy is based on first-order logic that can be used for structural modelling by expressing complex structural constraints and behaviours. Alloy treats relations as first class citizens and uses relational composition as an operator to combine various structured entities. The essential constructs (Alloy tutorial 05) of Alloy that we used for AOWS modelling, analysis and validation purposes are its signatures, functions, predicates, facts, and assertions. A signature (sig) is a paragraph that introduces a basic type, a collection of relations (called field), and a set of constraints on their values that can be defined in our AOWS. A signature may further inherit fields and constraints from another signature. A function (fun) evaluates the first order

expressions into a value. It is a parameterized function that can be used in other expressions. A predicate (pred) captures behaviour constraints in our AOWS and evaluates them into true or false. It is a parameterized formula that can be further applied in other constraints. A fact (fact) imposes global constraints on the relations and objects. A fact is a formula that takes no arguments and need not to be invoked explicitly. It acts as axioms in the model, which is always true. An assertion (assert) specifies an intended property in the AOWS system and it is a formula whose correctness needs to be checked, assuming the facts in the model. Alloy also has a wide collection of semantics (Alloy tutorial 05) that we used for modelling the AOWS in this thesis. The semantics are denoted by symbols or a sequence of characters that are recognized as semantic tokens, for example to represent formal language operators like ‘unions’, ‘intersections’, ‘subsets’ and ‘not’.

We had used the constructs and semantics extensively to model our AOWS system. After modelling we pass the models through the Alloy Analyze which is a tool for analyzing models written in Alloy. Given a finite scope for a specification, the Alloy Analyzer translates it into a propositional formula and uses Boolean Satisfiability (or SAT) problem solving technology based on complexity theory solutions to generate instances that can satisfy the facts and properties expressed in the specification. In other words, given a formula and a scope, a bound on the number of atoms in the universe, it determines whether there exists a model of the formula (i.e., an assignment of values to the sets and relations that makes the formula true) that uses no more atoms than the scope permits The Alloy Analyzer provides two kinds of risk analysis for our AOWS system. The first risk is that the constraints given are too weak. Flaws of this sort are found by the Alloy Analyzer by the checking assertions,

in which a consequence of the specification is tested by attempting to generate a counterexample. The second risk is that the constraints given are too strong; in the worst case, the constraints contradict one another and all possible states are ruled out. Flaws of this sort are found in simulation in which the consistency of a fact or function is demonstrated by generating a snapshot showing its invocation.

Alloy and its Analyzer have been used primarily to explore abstract software designs. Its use in analyzing code for conformance to specifications and as an automatic test case generator is being investigated in ongoing research projects (Alloy_homepage 05).

The key features of Alloy are that it allows for the expression and modelling of complex structures to be defined and constructed with just a few powerful operators (Dong et al 03). It can be applied to both static and dynamic structures and is highly declarative, i.e., it has a full logic, including conjunctions and negations, and describes a system as a collection of constraints. It is analyzable, i.e. it caters for both simulation and checking. The Alloy tool is fully automatic, with no user intervention required and can generate concrete samples and counterexamples. But just like any testing, it is sound but not complete. Alloy's analysis can execute a model forwards or backwards, even without test cases, and has no ad hoc restrictions on logic. As such, the Alloy and its Analyzer is a very useful tool and suits us very well in modelling, analyzing and verifying Aspect-Oriented Web Services systems which have some very complex and sophisticated structures, both static and dynamic.

2.8 Summary

Web services promise to enable business-to-business integration seamlessly and dynamically irrespective of the type of platform or language used. Web services-based systems can be very large and have become more complex and harder to understand and control. This can result in high development costs, unmanageable software quality, low productivity and higher risks. As a result, component-based software development (CBSD) methodologies have evolved to try to address these problems. These CBSD methodologies include the TopCoder ™, The Architecture Based Component Composition Approach, The COMO approach, The Select Perspective™, The Catalysis™ approach and OMG's Model Driven Architecture.

Though these development methodologies can be applied to large scale development of web services systems, the disadvantages are that all these techniques tend to focus on the lower-level component type and behaviour features of the system. This can be very hard to understand and manage at higher levels. The higher level systemic component descriptions such as persistency, user interfaces, security, transaction processing, performance etc. are all missing from these approaches. These systemic features also cross cut through the software components and are not addressed in the current CBSD methodologies giving rise to tangling code in their programs. These high-level features are very important for understanding systemic components and their functionalities, especially in the context of large and complex systems including web service based systems. These features are necessary for the efficient and effective description, discovery and integration of web services. As such there is an urgent need for a better and more efficient software development methodology. We propose using

a CBSD methodology called Aspect-Oriented Component Engineering (AOCE) to develop novel Aspect-Oriented Web Service based systems (AOWS).

In our Aspect-Oriented Web Service based systems, Inversion of Control techniques are used in a novel subsystem called the AOConnector that links the requester to the rest of AOWS subsystems to allow for clients to be engineered in a light-weight fashion and thus make them easy to design, develop and maintain. As an alternative implementation to the AOConnector approach, we further researched and incorporated multi-agents into AOWS so as to support more autonomous discovery, integration and consumption of our web services systems. Alloy, a formal modelling language is used to model our AOWS abstractions and its subsystems. The Alloy Analyser tool can be used to dynamically analyse and verify our AOWS abstractions and subsystems so as to prove that they are logically and mathematically correct.

3 Aspect-oriented Component Engineering

Aspect-Oriented Component Engineering (AOCE) is a new, novel and efficient component based software development methodology that has the aim of equipping and enabling software engineers develop more reusable, understandable, maintainable and scalable software systems by creating better characterised and categorised aspect-oriented software components that are used as the systemic building blocks (Grundy and Ding 02). These aspect-oriented software components are also more efficient and reusable as they also take into consideration and address the cross-cutting issues involved. As was discussed in the preceding chapter, component based software engineering techniques have been used to vastly increase and improve the reusability and portability of software parts. But the presently used component-based software development (CBSD) techniques, including those for web services tend to focus on low level software component interface design and implementation. This has very great setbacks in that it often results in development of components whose services are hard to understand and difficult to combine (Haines et al 97, Grundy et al 98).

Current component-based software development methodologies also make too many assumptions about the interrelationships of the components within the system. Furthermore the component documentation is too low level which again is hard to understand at higher levels. Also, the higher level systemic component descriptions such as persistency, user interfaces, security, transaction processing, performance etc. are not stressed on in these approaches. These high-level features, which can be efficiently described using AOCE (Grundy 00, Grundy and Ding 02), are very important for understanding systemic components and their functionalities. These

high-level features cannot be usefully incorporated into any of the existing CBSD's requirements engineering, architecture, design and implementation phases because the existing methodologies do not cater for these concerns nor do they provide support for them in their components. Aspect-oriented component engineering as such can overcome the limitations of current CBSD methodologies in these respects.

In this chapter, we will initially give an overview of how AOCE is applied to develop aspect-oriented web services so that we can understand the issues involved in this thesis in more detail. Then we trace the use of aspects in popular programming technologies like Aspect-Oriented Programming (AOP) (Kiczales et al 97) and Programming with Aspectual Components (Lieberherr 99). We will finally describe the AOCE methodology and explain how it can be applied to develop software systems.

3.1 Overview of applying AOCE to Aspect-Oriented Web Services development

The AOCE methodology brings about increased understandability and consistency of approach in developing software systems from inception right through to designs, implementation, deployment and any subsequent maintenance (Grundy 00, Grundy and Ding 02) The components produced are also better characterised and categorised resulting in them being more reusable, efficient and reliable. The main steps involved in applying the AOCE methodology to develop aspect-oriented web service (AOWS) systems are shown in Figure 3.1 below. We used existing UML modelling tools like

Rational Rose (Rational Rose homepage 05) and Microsoft's Visio (Visio homepage 05) to carry out the requirements engineering and draw the design and architectural diagrams for the development of the collaborative travel planner system.

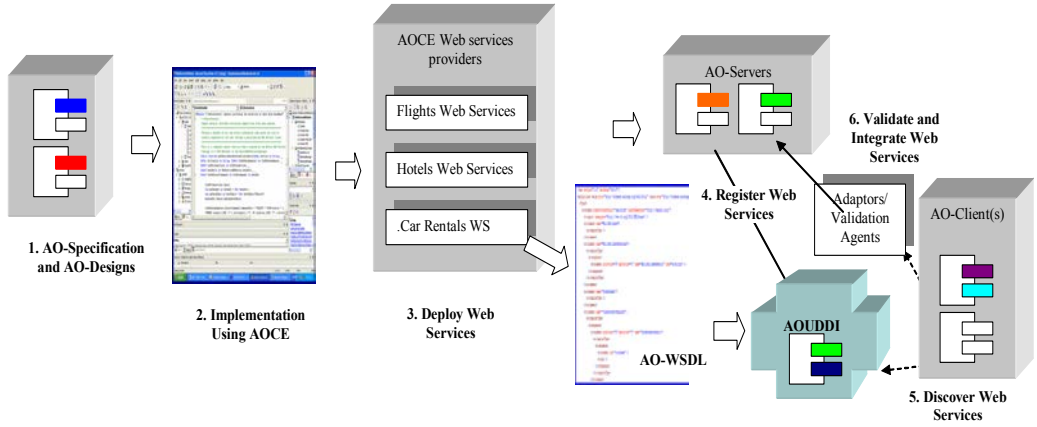


Figure 3.1: Using AOCE to develop aspect-oriented web service-based systems.

Figure 3.1 shows the development stages involved in the process of developing aspect-oriented web services systems using AOCE. The stages will be described in detail in the next chapters where they will be used to develop the AOWS-based Travel Planner application. First, aspect-oriented specifications and designs were considered and drawn to capture the functionality of desired web service clients and service components. The AO-specifications stage involves requirements engineering that take aspects and aspect-oriented components into consideration and is termed aspect-oriented Component Requirements Engineering (AOCRE) (Ding 02). AOCRE is described in detail in the last subsection of this chapter.

Aspect-oriented analysis involves analysis of the software components' interfaces and determines what each component's function is in the software system. Aspect-oriented designs involve aspectual considerations in designs and how the aspect-oriented component fits in with the rest of the components in the software to provide

the functionalities of the system. The aspectual characterisations in these designs also provide non-functional characterisations of clients and web services including their inter-relationships in components. The identification of functional and non-functional characteristics of system services a web service or client provides or requires is also depicted using the aspects.

The implementation stage can be carried out using any suitable IDE that can support web service-based application development, depending on the language chosen to implement it, for instance using Microsoft's Visual Studio .NET (Microsoft .NET website 05) or Eclipse IDE (Eclipse homepage 05) as the development IDE for C# and Java respectively. The web service providers were implemented with the need for them to be run-time discovered, validated and integrated, and to have associated aspect-enhanced service descriptions. To complement this, the web service clients were implemented such that they could discover their required web service providers dynamically. This was done with the aid of aspect-enhanced web service characterisations supplementing the conventional existing web service category and provider descriptions.

An Aspect-Oriented Web Service Description Language (AOWSDL) which is an aspect-enhanced Web Service Description Language (WSDL) (Christensen et al 01) was specially created here to support the description of aspect-oriented web services in XML format. The AOWSDL describes both the web services functional and non-functional properties based on aspects information. An Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI) tool was also implemented to deal with aspectual information relating to aspect-oriented web services. The AOUDDI

can also function like a normal UDDI (Adams and Boeyen 02) in that it can be used to describe and discover web service providers and integrate them with the web service requestors. But AOUDDI has extra features in that queries regarding the aspects and their details can be made to it by clients to obtain better and more informative descriptions of the aspect oriented web services. This is done to obtain the correct web service through multiple queries and to support dynamic discovery of web services by clients by taking advantage of the rich aspectual information available in the AOWSDL.

Aspect enhancements were also used to assist in discovering service adaptors. For instance, in order to communicate with a discovered service using a different protocol, the client will need to locate an appropriate adaptor to access it. Remote services may also need to be dynamically tested to ensure they meet advertised characteristics e.g. performance, security protocols and transactional behaviour. We used aspect characterisations to support dynamic testing agents running validation checks on discovered web services. Now that we have stirred the curiosity and interest of our readers in aspects and related technologies, we discuss these issues in more detail in the following sections.

3.2 Aspects and AOP

Gregor Kiczales and his research group at Xerox PARC conceived the idea of using aspects (Kiczales et al 97) in a novel programming technology that they called Aspect-Oriented Programming (AOP). Its main purpose was to address cross-cutting

issues that were spread out in the designs and implementation code of software system. These cross-cutting issues "mangled" the system's functionalities making them hard to understand, modify and control, and in some cases, ultimately, remaining as redundant code because no one wants to delete, edit or to have anything to do with it because understanding and dealing with it might take up too much time and resources. Neither procedural nor object-oriented programming techniques are sufficient to clearly capture these cross-cutting issues in software designs and implementations and neither of these programming techniques be used to address cross-cutting issues.

Concerns are said to crosscut if the methods related to those concerns intersect. AOP deals with crosscutting concerns and descriptions, designs, and implementations for those concerns. Functions used to describe, design, and implement a given concern are called methods. In AOP, a method is said to be related to a particular aspect if the method contributes to the description, design, or implementation of a concern that cross cuts through the objects or components of the software.

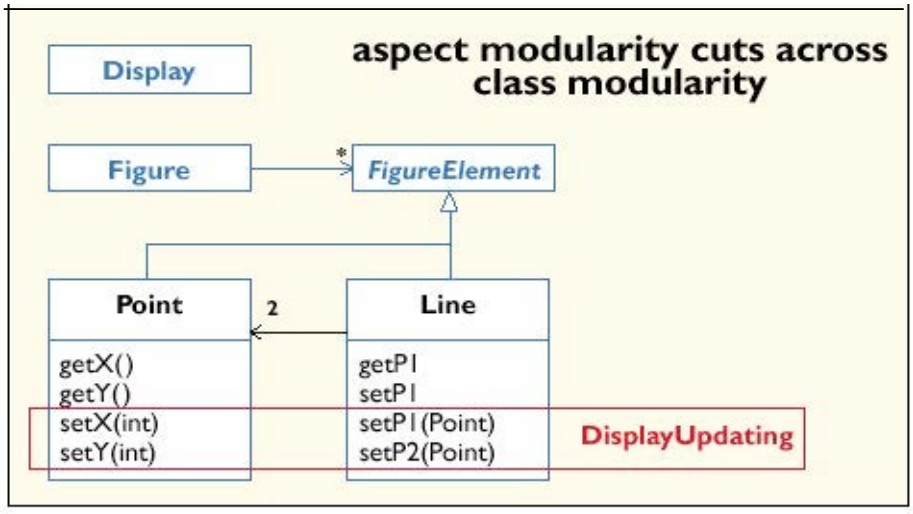


Figure 3.2. Aspects crosscutting classes in a simple figure editor (Elrad et al 01).

As an illustration for understanding more about aspects, we consider a UML diagram for a simple figure editor, as depicted in the figure 3.2, in which there are two concrete classes called the Point and Line classes of the FigureElement super class (Elrad et al 01). These classes manifest good modularity, in that the source code in each class is closely related and each class has a clear and well-defined interface. But consider the concern that the screen manager should be notified whenever a figure element moves. This requires every method that moves a figure element to do the notification so that the display gets updated (DisplayUpdating).

The red box in the figure is drawn around every method that must implement this DisplayUpdating concern, just as the Point and Line boxes are drawn around every method that implements this cross-cutting concern. Notice that the box for DisplayUpdating behaviour fits neither inside of nor around the other boxes in the figure, instead it cuts across the other boxes. This is why we call it a crosscutting concern. The bigger the application, the more pronounced and cluttered the cross-cutting issues become (Kiczales et al 97). Using just Object Oriented programming, the implementation of crosscutting concerns tends to be scattered throughout across the system, just as it is shown in the figure above. But by using the mechanisms of AOP, the implementation of the DisplayUpdating behaviour can be captured and modularised into a single aspect. Since we can implement this behaviour in a single modular unit, it makes it easier for us to think about it as a single design unit. In this way, having the programming language mechanisms of aspects lets us think in terms of aspects at the design level as well (Elrad et al 01).

These cross cutting issues are called aspects, and other terms like tangling, intermingling, mangling and interleaving units have also been used to describe them. A better description of aspects is attributed to Gregor Kiczales (Kiczales et al 97) where he states that an aspect is a modular unit that cross cuts the structure of other modular units and that it can encapsulate state, behaviour and behaviour enhancements in other units. It is stated that the goal of AOP is to make designs and code more modular, meaning the concerns are more localized using AOP rather than scattered, and have well-defined interfaces with the rest of the system (Elrad et al 01). This provides us with the benefits of modularity, including making it possible to reason about different concerns in relative isolation, making them pluggable and amenable to separate parallel development during the software development process.

There are two types of aspects, design aspects and program (or code) aspects. Modular units of design that cross-cut the structure of other parts of the design are called design aspects. Similarly modular units of programs that cross-cut other modular units of programs are called aspects. Both these types of aspects give rise to cross-cutting issues in designs and implementations.

The AOP methodology formulated techniques to solve cross-cutting issues and these include isolation of aspects, reuse of aspect code and composition of aspects from the onset. AOP also uses an approach which is called code weaving to tackle cross-cutting issues in programs. This technique is carried out by composing the aspects properly, identifying the regions where the aspects appear and weaving the aspects into the regions so as to produce the desired results.

AOP also clearly distinguishes components from aspects. In this methodology, a component can be cleanly encapsulated in a generalised procedure. Furthermore components tend to be units of the systems functional decompositions. For instance, in a collaborative travel planner system, the booking and system customer are both components because they can be both cleanly encapsulated in a generalised procedure and are units of the systems functional decomposition. On the other hand, aspects cannot be cleanly encapsulated in a generalised procedure and tend not to be units of the systems functional decomposition but rather are to be regarded as properties that affect the performance or semantics of the component in the system. These make distribution issues, persistency and performance to fall into the category of systemic aspects.

Gregor Kiczales and his team also developed the first, and still most popular, AOP language, AspectJ (Hannemann and Kiczales 02, AspectJ homepage 05). AspectJ is actually an aspect-oriented extension to the Java programming language and the Xerox PARC group's work is now integrated into the currently popular Eclipse Java IDE (Ecllipse-AspectJ homepage 05). There are 3 basic constructs in AspectJ, i.e. the join points, pointcuts and advice. The joint points are the points where the crosscutting occurs, a pointcut defines a set of execution points for the joint points and the advice represents what to do in the cross-cutting area. Other commercial Aspect-oriented frameworks include JBoss (JBoss homepage 05), AspectWerkz (AspectWerkz homepage 05) and Spring AOP (Spring framework homepage 05). All these have also helped popularise AspectJ that has become one of the most widely-used aspect-oriented languages to address cross-cutting issues. AOP techniques have been used in other languages and platforms as well. AOP has been applied to

metadata and their interceptors in the Aspect Builder (Shukla et al, 02) application, where services between clients and other objects were stacked semi-seamlessly in COM and seamlessly in .NET. Microsoft has also announced that it has been developing a state-of-the-art aspect-oriented programming tool called Aspect.NET (Safonov and Grigoryev 05) which can be integrated with the latest Visual Studio IDE. In this project Microsoft aims to make AOP ubiquitous for .NET software engineers, develop the most adequate ways of representing aspects and lay the foundation for future research and development work on spreading AOP among .NET users.

Therefore the Aspect-Oriented Programming methodology together with ideas from its vast research carried out so far has contributed immensely to our understanding of these cross-cutting modular units called aspects in design and implementation (Kiczales 05). They have also helped software developers gain a better understanding in separating components from aspects.

3.3 Adaptive Programming

Adaptive Programming (AP) (Lieberherr 99) which is attributed to Karl J. Lieberherr is a programming methodology that describes components and aspects as views, and they need not be clearly separate from each other. The goal in AP is to separate views by minimising dependencies between views so that modifications in one view will have minimum impact on other views.

Adaptive Programming can be regarded as a special case of Aspect-Oriented Programming where one of the aspects is expressible in terms of graphs and the other aspects or components refer to the graphs by using traversal strategies. In this case, when one of the aspects is viewed in terms of graphs, the other aspects or components follow defined ‘strategies’ to control and interpret references to these graphs.

The coexistence of multiple views is also regarded as very crucial in Adaptive Programming. The views here are constructed in such a way that they refer to each other. These include references to internal parts of other views as well and as such give rise to unavoidable issues of cross-cutting. In Adaptive Programming, the views are complementary and collaborative so that each view can address a different concern of the application.

Karl Lieberherr and his group proposed the Programming with Aspectual Components (Lieberherr et al 99). This technique is a merger of convenience between AOP and component-based programming. Aspectual components are produced here and they are programmed in a generic data model. These aspectual components are essentially effective tools for AOP. Aspectual components extend adaptive plug-and-play components with a modification interface that turns them into tools for AOP. A key ingredient of aspectual components is that they are written in terms of a generic data model, called a participant graph, which is later mapped into a data model that can be used with AOP.

3.4 Identifying Aspects in Components using AOCE

Unlike Aspect-Oriented Programming and Programming with Aspectual Components, Aspect-Oriented Component Engineering (AOCE) is not a programming technology. AOCE is a new software methodology for developing more reusable, scalable, extensible and dynamically adaptable aspect-oriented software components. We use the term “aspects” to denote modular cross-cutting units that horizontally slice the overall software system that was vertically componentised into software components. Figure 3.3 below illustrates this concept of aspects cross-cutting components in software systems built using AOCE. These aspects characterize specific cross-cutting functional and non-functional properties of the components. Examples of aspects include all those cross-cutting features for security, persistency, configuration, collaboration, transaction processing, distribution, user interface, performance and resource utilization.

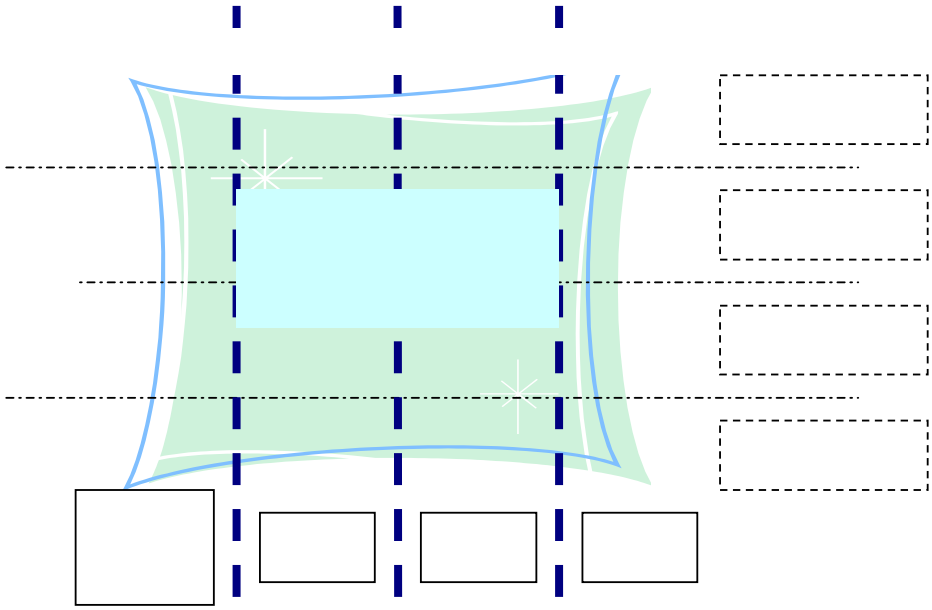


Figure 3.3 Components and component-aspects in AOCE

AOCE is as such not a programming technology like Aspect-Oriented Programming or Programming with Aspectual Components. AOCE focuses on the whole software developing lifecycle, including requirements engineering, analysis, design, implementation, testing and deployment. It also encompasses subsequent refactoring and maintenance of the software system.

The aspects themselves may vary with different domains (Grundy 00), for instance security-critical systems have various security related aspects; safety-critical systems have redundancy and high assurance aspects whereas real-time control systems may have more priority for performance (response time), resource utilization (memory management) and concurrency aspects.

Aspects can thus be clearly identified for each software component within a system. There will normally be more than one aspect for each component that provides functional or non- functional services for other components within the system or end users to use. Similarly, the component itself may require one or more aspect-oriented functional/non-functional services from other components in order to function properly.

3.5 Aspect and aspect details in AOCE

Each aspect that is provided or required has some "aspect details" which are used to more precisely describe the systemic properties of the component related to the aspect. For instance, one component may provide data encoding or encrypting

services of the security aspect while another component require data retrieving and storage services of the persistency aspect. The data encoding, encrypting, retrieving and storage services are all aspect-details.

The aspect details may have one or more aspect detail properties that are used to further characterize the aspect information (Grundy and Patel 01). The aspect detail properties will relate to functional and/or non-functional characteristics of the aspect detail. For instance, a security aspect might have aspect detail properties for data encoding algorithms, encryption algorithm, encryption key length and key type properties. All these detail properties make the aspects richer and more useful and meaningful to the components.

Besides functional constraints, aspects in AOCE are also used to capture non-functional constraints of components. An example of a non-functional constraint would be the performance of a persistency aspect, i.e. the speed of retrieving data or the maximum data size that can be retrieved.

The main types of Aspects identified based on this PhD research and used in this thesis are tabulated in Table 3.1 below:

	Aspect	**Aspect Details**	**Description**
1.	Transaction Processing	Send/receive data Commit Rollback Lock	Components requiring or providing transaction processing capabilities.
2.	User Interface	GUI/BUI	Components requiring or providing

		Views (process views) Forms/Frames Feedback Responses Extensible parts	user interfaces. This includes extensible and composable interfaces for several components.
3.	Performance	Robustness Speed Accuracy	Components requiring or providing for speed, accuracy or robustness.
4.	Security	Encoding Decoding Access control Key distribution Authentication	Components requiring or providing inter-component security, data encoding and cryptography
5.	Persistency	Locate data Lock data Store data Retrieve data Storage media Reliability	Components requiring or providing data persistency management facilities. Detail properties could include select, update, delete and insert data. Usually provided/required by data managers.
6.	Distribution	Locate objects/data Send/receive data Object transfer Allocate resources/processing	Components requiring or providing data or object distribution facilities. Usually provided/required by middleware components like CORBA/RMI.

7.	Resource Utilization	Memory usage Threads/processes involved.	Memory resources, threads and processes provided/required by the components.
8.	Configuration	For configuration of components.	Provided/required by the components for configuration purposes.
9.	Collaboration	For collaboration with other components or systems.	Provided/required by the components for collaboration with other components/systems.
10.	Web Services	Provides/requires web service functions. Searching and locating web services.	This kind of aspect is only present in web services related components/sub-systems. It is the Web Service function (method) in the code. It can exist independently or as a Composite Aspect. In a Composite Aspect several other aspects like Distribution aspects and Web Services aspects are so entwined with each other that they are simpler to be considered as a single aggregate aspect.
11.	Debugging Aspects	Provides/requires debugging functions. Includes break points in code, printing to the IDE Console.	These are mostly transient in nature in that they only live during the development of the system. Once the system is fully developed these aspects loses their debugging function

			and are discarded. But some Debugging Aspects are purposely left in the system and they become Message Aspects.
12.	Message Aspects	Provides useful messages/information to the user. These aspects extract information from other aspects.	They serve to provide information to the user and the user cannot change them. For instance, information given to the user during the start-up of an operating system. Initially this information could have been used for debugging purposes, but since it was useful it was left in the application.

Table 3.1: Description of Component Aspects and Aspect Details

3.6 Aspect-oriented Component Requirements Engineering

The aspect-oriented Component Requirements Engineering (AOCRE) (Ding 02) is a subset of AOCE and is used to clearly identify and specify functional and non functional requirements in relation to the aspects of a system. These aspects are the crosscutting systemic services that are either provided or required in the systems components. Also components may have different number of aspects, some may have many aspects while others will just have a few. Some components may even share aspect services.

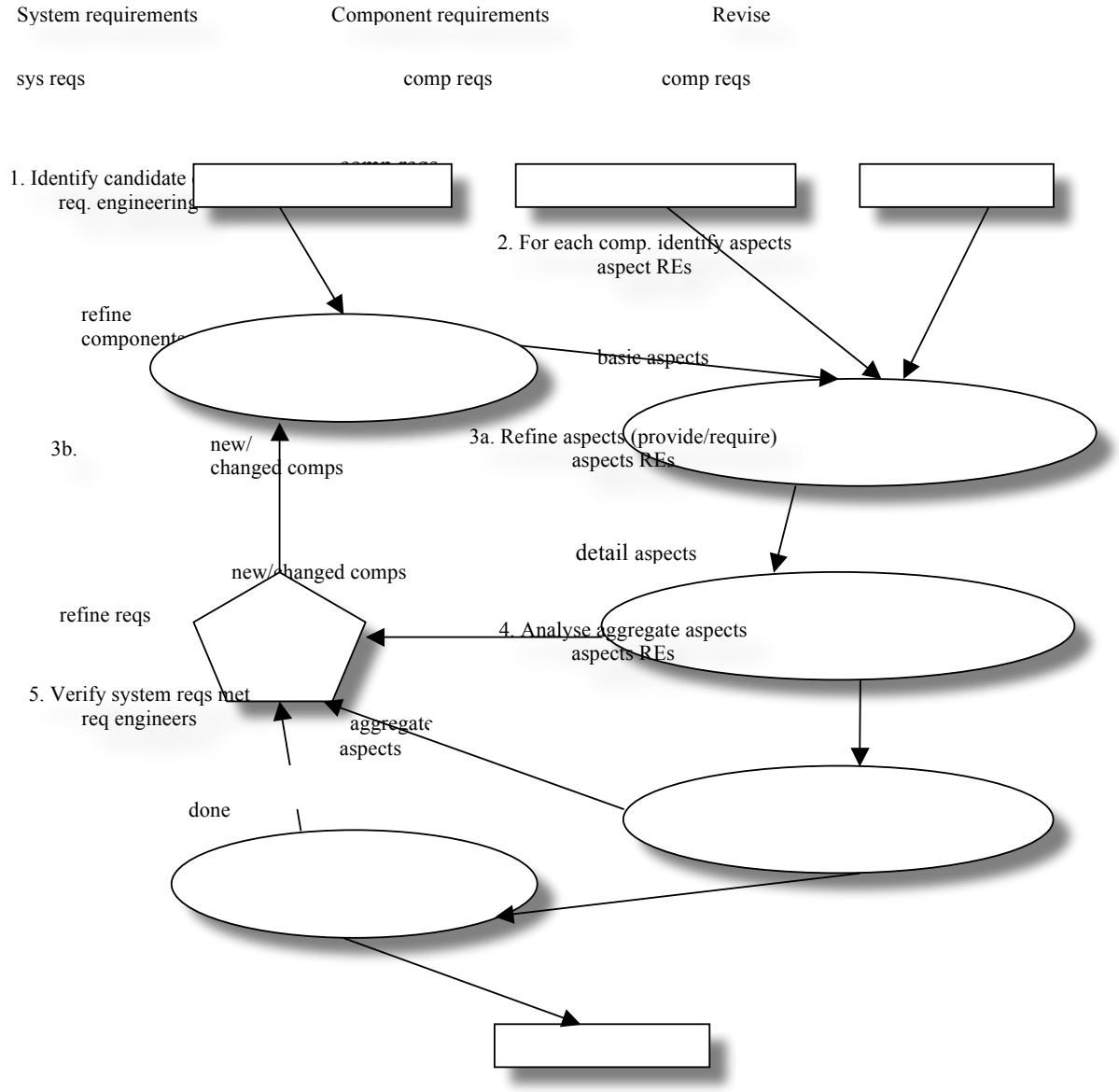


Figure 3.4 Basic AOCRE process flow

Requirement engineers will find it very helpful to utilise and understand not only the individual disparate aspects in components but also all overlapping aspects identified in different components. These overlapping aspects are the reflection of a components high level systemic characterisation. Figure 3.4 above illustrates the basic process flow of AOCRE. It starts with several system application requirements or where individual or clusters of components requirements are analysed. As shown in the figure, requirements are carried out iteratively until the system requirements are met, which then moves on into the design phase.

3.7 Summary

Aspect-Oriented Component Engineering (AOCE) is a complete software development methodology that is used to develop better characterised and categorised aspect-oriented software components that are also more efficient, maintainable and reusable as compared to non aspect-oriented components. Aspects are cross-cutting modular units that are spread out in the designs and implementation code of software systems. If not addressed, they can "mangle" the system's functionalities making it hard to understand, modify and control them, both in the development process and in the software being developed. We apply AOCE to the development of aspect-oriented web services (AOWS) by using the concept of components providing services for one or more systemic aspects and requiring one or more aspect services from other components.

AOCE also focuses on factoring the web services components' crosscutting systemic issues into component interfaces so that components can be run-time located, reconfigured, dynamically composed and reasoned about. AOCE makes components provide richness through multiple perspectives and structures component requirements and designs clearly. AOCE also allows for better dynamic configuration and decoupled components interactions.

An Aspect-Oriented Web Service Description Language (AOWSDL) which is an aspect-enhanced Web Service Description Language (WSDL) is needed to support the

description of aspect-oriented web services in XML format. The AOWSDL describes both the web services functional and non-functional properties based on aspects information. An Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI) tool was also implemented to use the aspectual information in the AOWSDL relating to aspect-oriented web services. AOUDDI allows for the better description, discovery and integration of web services.

4. Requirements Analysis

Requirements Analysis involves the collection and interpretation of information and instructions from the client/clients regarding the specifications and requirements of the software system that needs to be designed and developed (Bennet et al 99). In AOCE, as discussed in the previous chapter, it involves using AOCE techniques (Grundy 00, Grundy and Patel 01, Grundy and Ding 02) to gather the information regarding the specifications and requirements of the system which will then be further analysed and refined in the analysis and design stages and then subsequently implemented. We call this process of information gathering for AOCE purposes Aspect-oriented Component Requirements Engineering (AOCRE) and it is used to produce the use cases for AOCE (Ding 02).

We will first give an overview of our software system, a web services based collaborative Travel Planner application, that we developed using AOCE so that its requirement engineering can be better described and understood. We then explain the requirements analysis that we carried out for this collaborative Travel Planner as regards the type of operations the application need to support This requirements engineering is divided into functional and non-functional requirements. We will further build on these requirements engineering and specifications in later chapters to complete the analysis, design and development of our novel AOWS-based system. That is, the specification, analysis and design of the AOUDDI and AOWSDL is described and discussed in Chapter 6 where we explain aspect-oriented description, discovery and integration mechanisms of AOWS systems in detail. The novel AOConnector object is

then explained in the next chapter so that we can illustrate and describe how it connects the various subsystems that were explained earlier and how it fits in and functions with the rest of our system.

4.1 Overview of the Web Services Based Collaborative Travel Planner Application

We considered the analysis, design and development of a collaborative travel planning application built from dynamically discovered web services providing travel item searches (e.g. for flights, cars, trains, hotel rooms etc), booking, payment, event scheduling and itinerary management. The application was developed to enable dynamic discovery of appropriate services providing these functions. Multiple, alternative service providers may also be discovered. The web services discovered may provide limited or comprehensive functionality. Some services may be free, others may require payment. Also they may be from “trusted” providers or unknown 3rd parties. Some may support business transaction models, respond faster than others to requests, or support security models that others don’t. During service discovery it may be necessary for validation of a web service to be performed to ensure it actually meets its advertised characteristics.

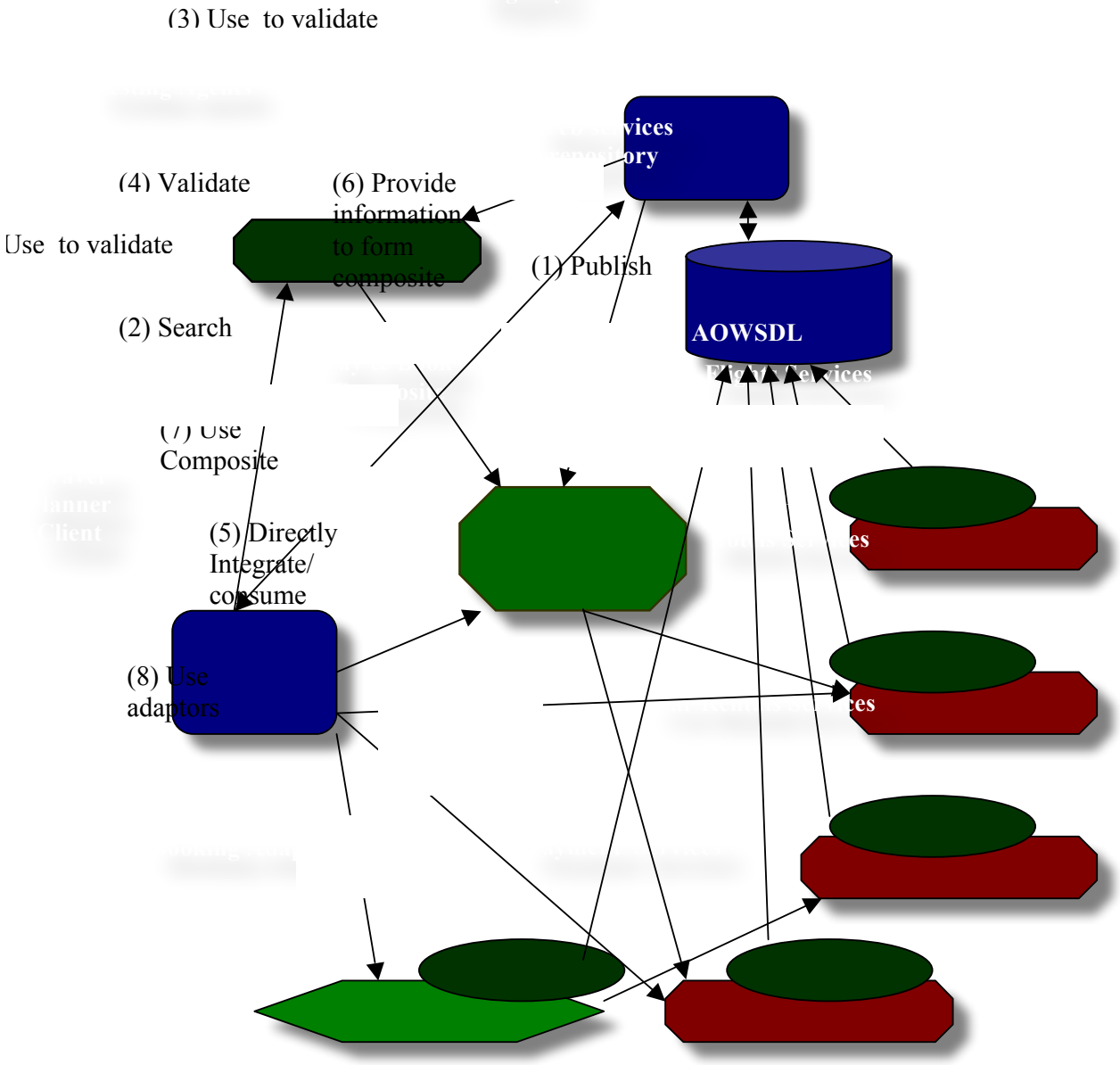


Figure 4.1: Collaborative Travel Planner architecture based on AOWS.

The architecture of our Travel Planner system is shown in Figure 4.1. It is based on the Aspect-oriented web services architecture discussed in Chapter 1 (Figure 1.2). Some of the aspect-oriented web service providers are shown in the above diagram, e.g. the Flights web service, Hotels web service, Car Rentals web service and Payment web service etc. The numbered items shown in the figure depict the sequence of events when applying AOWS concepts in our Travel Planner. These service providers first publish (1) their service documents (AOWSDL) on the AOUDDI relaying information about

their exposed aspectual APIs, the services they provide, their location and other technical details including on how to consume them.

The web service requester here is the Travel Planner Client. It will query (2) the AOUDDI to locate the web service providers that it requires. The AOUDDI and clients can use (3) the Testing Agents and aspectual information to further verify and validate (4) that the services actually satisfy/provide the required service that the client wants, e.g. check whether it conforms to a required performance level or returns a data set containing all the required items based on a particular query.

If the client discovers a suitable service provider it can integrate directly (5) with the provider and consume it to utilise the exposed aspect oriented APIs. However if the client can't locate a single provider, but there exist a number of service providers that if bundled together can provide all services requested for, then a composite object (6) is constructed that is composed of all these providers. The client will then integrate (7) with the composite object and use it to consume the services. In the diagram shown, a composite object is constructed composed of a payment service provider and a hotels service provider. This composite will enable clients to use it to make reservations in hotels and enable payments to be made towards it. Adaptors can also be located (8) so that they can be used, for instance, to consume web services using different protocols. Here a booking adaptor is used to map the car rental services protocol (that was not written in SOAP) to the SOAP protocol so that it can be understood by the clients. Adaptors can also be used to map extended WSDL documents containing all the aspectual elements of the Aspect-Oriented Web Service provider (but not written in the correct format/order of the AOWSDL) to proper AOWSDL documents.

In the following subsections, we will refer to and explain the type of operations our Travel Planner application described above need to support. We will discuss these operations under the functional and non-functional requirements subheadings below. The use cases and their descriptions that are described below are in high level language so that they can be easily understood by both the software developer and client alike.

4.2 Functional Requirements

Functional requirements (Bennet et al 99) describe what a system does or is expected to do. The system described here is the collaborative web based Travel Planner system as discussed in the examples above. It consumes a number of web service providers with functionalities pertaining to booking flights, hotels, trains, making payment etc. Listed below are the functional requirement specifications for the Hotels Web Service. The functional requirements for the other web services are included in the appendix.

1. Users should be able to consume the web service using a variety of clients, including thick clients, thin clients and other web services.
2. The clients can be implemented in any language or platform.
3. The web services should allow users access to any of its exposed API's.
4. The system should allow users to search for hotels based on city or country queries.
5. The system should allow users to search for vacant rooms in a particular hotel.
6. Users should be allowed to make bookings of the vacant rooms of their choice for periods ranging from one to more days, depending on availability.
7. The system should allow the users to change, cancel or update their bookings.

8. The system should be able to disclose information about places and activities of interest in the vicinity of the hotel.

9. The system should allow users to be able to view their bookings.

10. The system should allow users to be able to make payments for the occupancy of the rooms based on a method of their choice.

11. The system should only allow specified staff or management to access and manage the systems database.

12. The system should permit selected staff to view, add, delete or update booking information.

13. The system should permit selected staff or management personnel to view, add, delete or update the customer information.

14. The system should allow the customers to view, add, delete or update only their own respective customer information that they have entered into the system.

15. The system should allow the customers to view, add, delete or update only their own respective booking information that they have made.

16. The system should keep information about customers, their bookings and payment details.

17. The system should permit selected staff to view and add, delete or update hotels, rooms and related information.

18. All users should be able to be authenticated through the use of unique logins and passwords combinations before they use the system.

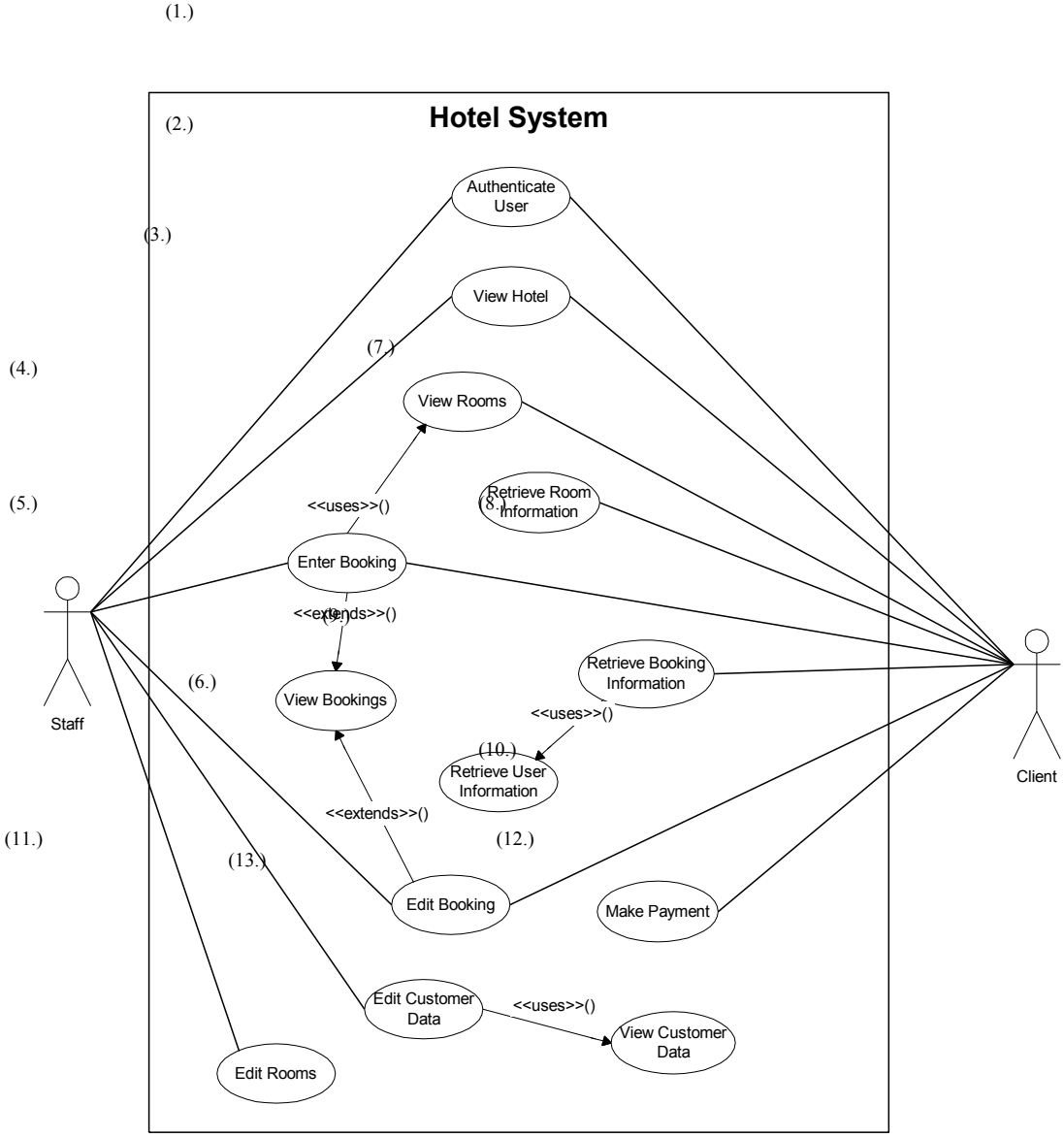


Figure 4.2: Use Case diagram for the Hotel System

Figure 4.2 depicts the use case diagrams that describe the requirements of the Hotels web service system. An authentication process (1) is required for security purposes so that users, both staff and clients, using the system need to be checked to be given authorisation for user specific and sensitive information manipulation, for instance, to make bookings, payments, edit profiles or itinerary etc. By utilising this system a user should be able to view all hotels (2) in a particular city or country of his interest. Users can then select a hotel of his/her choice and view the rooms (3) and other facilities

available in the vicinity of the hotel. The user can further make bookings of the rooms available (4) if he is satisfied. He can view all his bookings (5) and subsequently edit the bookings (6) if he desires. The user should be able to retrieve information of all the rooms that he has stayed (7) so far and information on all the rooms he has booked (9) so far as all these will serve as important and convenient sources of information for future bookings. He should also be able to retrieve information about his own profile (8) so that he can get it edited, updated etc. to be correctly identified and contacted and his preferences, likes, dislikes etc. can be passed on to third parties if he agrees to allow the company to do so for promotion of goods and services that may be of interest to the user. The user should also be able to make payment (10) for any bookings that he has made.

As shown in the above figure, staff should also be able to access and view (12) the customer data and edit (13) it on behalf of the customer. They should also be able to edit (11) information about the rooms to keep up with updates etc. These are in line with staff being given more rights so that they can maintain the proper state of the data in the system and make any changes/amendments to the data based on requests by customers.

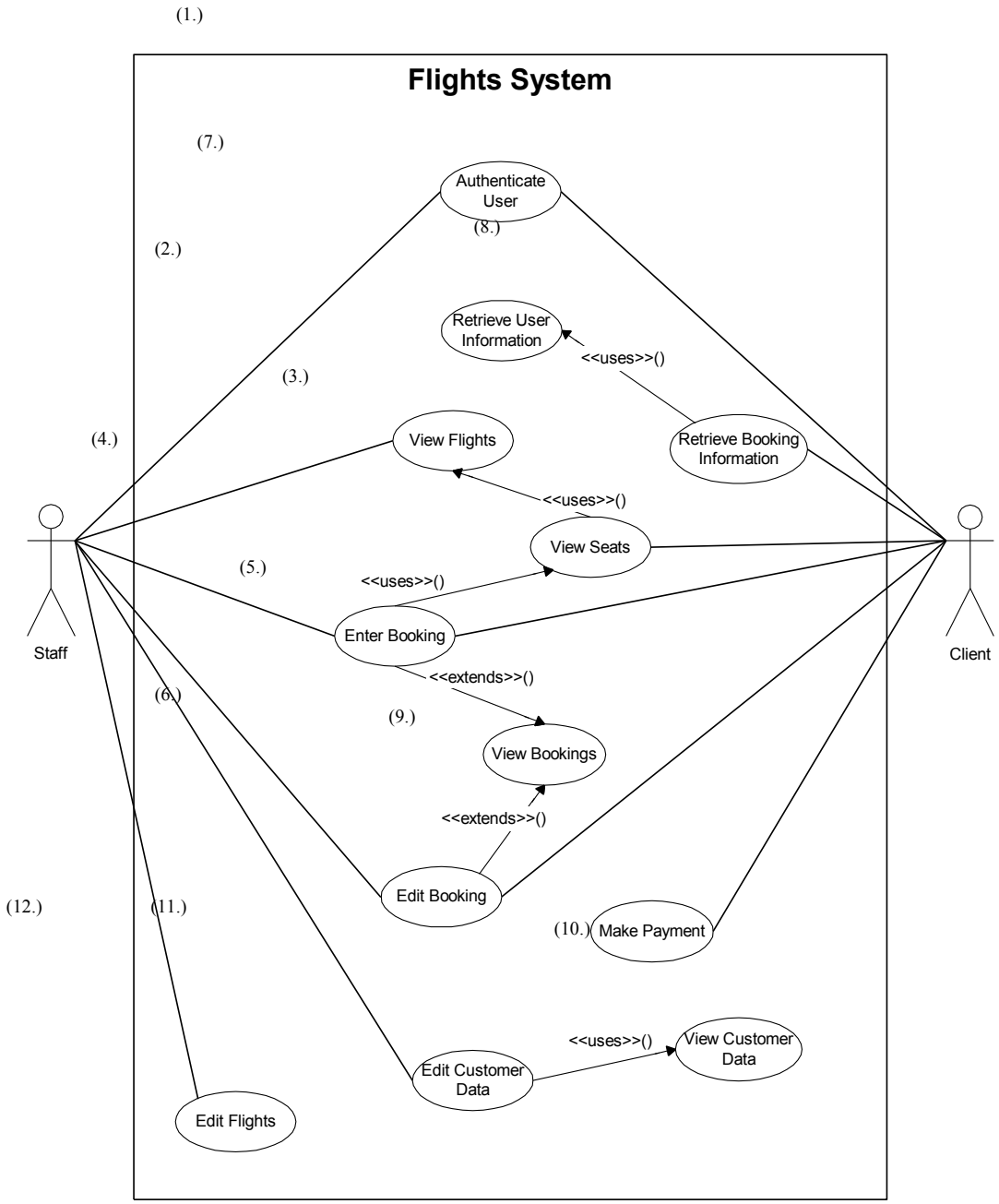


Figure 4.3: Use Case diagram for the Flights System

Similarly Figure 4.3 above illustrates the use cases that describe the requirements of the Flights web service. It's a comprehensive system whereby the various flights can be viewed based on queries relating to timing, departure and destination details. Seat details can be viewed for specific flights and bookings subsequently made if found to be

satisfactory. These bookings can be further edited if change of plans occurs. The use cases shown are discussed below.

As shown, users need to be given authorisation through authentication processes (1) to do personal tasks like making bookings for flights, editing their itinerary, user profiles, arranging to make payments etc.

Users can view (2) all available flights from a particular place of origin to other destinations at the times specified by the user. Users can then select a particular flight suitable to him/her and view the seats (3) facilities available on the flight. The user can further make bookings of the seats available (4) on the flight if it serves his purpose. He can also view all his bookings (5) and subsequently edit the bookings (6) if he wishes to do so. The user should be able to retrieve information of all the flights that he has taken (7) so far including information on all the seats he has booked (9) so far as all these will serve as important and convenient sources of information for future bookings/travel. Again, just as with the hotels booking system, the user should also be able to retrieve information about his own profile (8) so that he can get it edited, updated etc. to be correctly identified and contacted and his preferences, likes, dislikes etc. can be passed on to third parties if he agrees to allow the flights company to do so. The user should also be able to make payment (9) for any bookings that he has made. Staff should also be able to access and view (10) the customer data and edit (11) it on behalf of the customer. They should also be able to edit (12) information about the flights to keep up with updates and latest information about their services.

4.3 Non-Functional Requirements

The non-functional requirements (Bennet et al 99), i.e. functions that relate to systemic features such as performance, maintainability and portability of the hotels web service are described below. Non functional requirements of other web services are described in the appendix.

Non Functional Requirements:

1 **Maintainability:** To be labelled as a quality software product, the system must be maintainable (Siegel 02). The AOCE development techniques used will be clear, consistent and coherent so such that it will be possible to refactor and enhance the system with extra functionality whenever the need arises to do so.

2 **Documentation:** The documentation for the application will be clear, concise, relevant and helpful. It should also contain a description of the application and any advance features or short-cuts to its usage that the application may support.

3 **Reliability:** To prevent inconvenience to users, the system will be required to trap/handle any errors. Here, the severity of the error will generally determine the course of action to be taken. The system will generate an error log for any errors occurring during the time of operation. If the error is recoverable, then the system will output a message notifying the user that an error has occurred and that the system is taking a recovery action. Also if the system receives invalid data through the user interface, it should handle it appropriately and display the necessary messages.

4 **Error Handling in Extreme Conditions:** Should the error be too severe to recover from, the system will save as much work as possible and halt execution after displaying an appropriate error message. The system will require an Uninterrupted Power Supply (UPS) in case of power failure.

5 **Robustness:** The system should be robust enough to be able to handle multiple simultaneous accesses by remote clients without crashing.

6 **Customer Security:** The system provides login and password facilities to be used by customers to access the information.

7 **Staff Security:** Login and passwords also are required by all staff members who use the system.

8 **Speed:** Response time should be less than 5 seconds when accessing and consuming the web service from a remote client.

9 **Accuracy:** Accuracy of information retrieved should be more than 90%.

10 **User Interface:**

I. **Simplicity:** The user interface should be simple, straight forward and any unnecessary complexities should be avoided. The user should be able to easily understand what the system is trying to portray and there should be no ambiguity in the user interface.

II. **Ease of Use:** The user interface should be comprehensible, clear, consistent and easy to use.

III. **Support:** Appropriate additional help should be available for the user where instructions become too complex and not easily understandable.

IV. **Able to Learn Quickly:** Users should be able to understand the system with ease and relatively quickly, including naïve computer users and those new to the system.

V. **Visual Appeal:** The user interface should be aesthetically pleasing for users and the layout should also be well spaced and not cluttered.

These functional and non-functional requirements are very important so that software developers get a clear idea and description of the system they are going to build.

4.4 Use case descriptions

Given below are two detailed use case descriptions and their event flow about the hotels web service specifications described above. More use case descriptions are listed in the appendix.

View Hotels	
Actor	Customer or staff using a web service client
Precondition	User enters name of city or country that the customer wishes to visit.
Post-condition	System displays the results of the search on the screen.
Description	User searches the information stored in the hotels database about the hotels the customer wishes to stay at in a particular country or city.
Basic course of action	User queries using name of a city or country where the hotels are located and the results of the query are returned in a dataset and are displayed on the screen.

Table 4.1: Use case descriptions for “View Hotels”

The Use Case event flow for “View Hotels” outlined in table 4.1 is explained below.

1. **Used by:** customers or staff via web service clients to search for hotels based on queries containing name of city or country where the hotels are located.
2. **Event flows:**
    2.1. Repeat until customer or staff finds hotels information or leaves the web service:
        2.1.1. User enters name of city or country where the hotels are located to query the web service.
        2.1.2. User clicks search button on a requesting client and a list of all hotels in the city or country are returned.
    2.2. If server error – error message displayed. Go to 2.1
    2.3. If no hotels found – error message displayed. Go to 2.1
3. **Related Actors/Use cases:** Used by Travel Planner client. Must use this function before searching for rooms in a particular hotel.
4. **Special conditions:** uses Web Services Technology and SOAP or HTTP protocol. Must be used by a web service requestor.

View Rooms	
Actor	Customer or staff using a web service client
Precondition	User enters name of hotel and its location after doing a hotels search first.
Post-condition	System displays the results of the search on the screen, it contains the details of all the rooms in the particular hotel.
Description	User searches for the details of all the rooms stored in the hotels database about the hotel that the customer wishes to stay at.

Basic course of action	User queries using name of hotel and its location after doing a hotels search first. The results of the query are returned in a dataset and are displayed on the screen.
Alternative courses of action	1.) If customer is a new customer, then the system will request the customer to register first. 2.) If user already knows the hotel name and its location he can query the web service about the rooms directly without performing a hotels search first. The results of the query are returned in a dataset and are displayed on the screen.

Table 4.2: Use case descriptions for "View Rooms"

The Use Case event flow for "View Rooms" as outlined in table 4.2 is explained below.

1. **Used by:** customers or staff via web service clients to search for vacant rooms based on queries containing name of the hotel and its location.
2. **Event flows:**

    2.1. Repeat until the customer or staff finds vacant room information or leaves the web service.

        2.1.1. User enters name of hotel and its location after doing a hotels search first to query the web service.

        2.1.2. User clicks search button on a requesting client and a list of all rooms in the hotel concerned are returned. The list clearly shows whether the rooms are vacant or not.

    2.2. If server error – error message displayed. Go to 2.1

    2.3. If no vacant rooms found – error message displayed. Go to 2.1

3. **Related Actors/Use cases:** Used by Travel Planner client. Must use this function to view and search for vacant rooms before booking the rooms in a particular hotel.
4. **Special conditions:** uses web services technology and SOAP over HTTP protocol. Will be used by a remote client to the web service-based Travel Planner application.

4.5 Summary

We captured the web services systems functionality of the collaborative Travel Planner described here from the specifications in the Requirements Engineering. Both the functional and non-functional requirements of the Travel Planner were discussed in detail here. Use case diagrams were provided together with use case descriptions to further clarify the Requirements Engineering process. Both the use cases and their descriptions are in high level language so that they can be easily understood by both the software developer and client alike. Based on the analysis of the Requirements Engineering of the web services, the system will be decomposed into several aspect-oriented components with functional services. These aspect-oriented components are also better characterised and categorised when compared with their non aspect-oriented counterparts.

5. Applying AOCE to the Analysis and Design of Web Services Components

The demand for using more efficient, understandable and reusable components in Component Based Software Systems increases by the day (Haines et al 97). The fuelling factor behind this is that efficient components are viewed as user-friendly, flexible and easily pluggable building blocks that communicate via their interfaces and as such they can be more easily reused in other parts or other software systems that require their functionalities. AOCE techniques can be used to design and construct these types of efficient components as aspect-oriented components are well defined, understandable and highly modularised. They can also be more easily refined or reused in other systems that need their functionalities. In this section we will discuss the analysis and design of the software components of the collaborative Travel Planner based on a web services system using the AOCE techniques and ideas explored here. We first give an example from the Travel Planner application to illustrate some of its operations and how aspects may cross-cut the system so that we can consider the impact of aspects when we analyse and design the software during the development process.

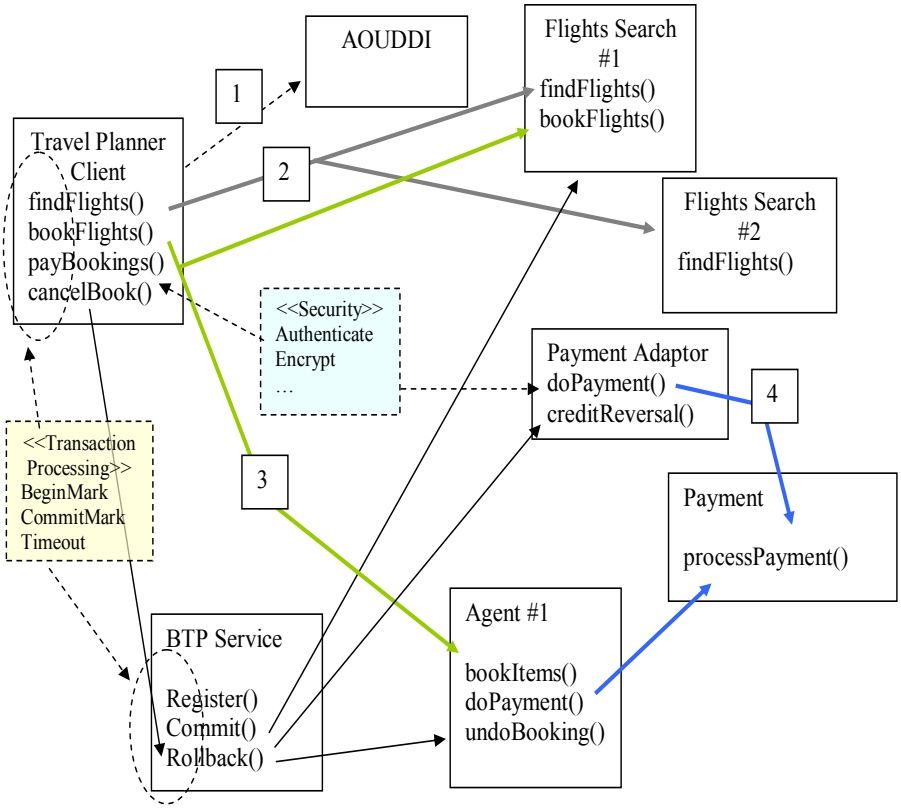


Figure 5.1. Examples of web service aspects.

Our Aspect-Oriented Web Services system (AOWS) uses the concept of aspects (Kiczales et al 97), in this case aspects that impact on different parts of web services. Figure 5.1 shows an example from the travel planner system that was developed based on AOWS, depicting some of the operations and aspects that cross-cut the system. The client discovers various services from a registry as shown by the broken line numbered (1) in the figure above. Flight searches are performed via various providers as depicted by (2), and bookings made directly or via agents shown by the green lines in (3), possibly using a payment system (such as credit card authorization) as represented by the blue lines (4). Two examples of web service aspects (dotted boxes) and their impact are illustrated. The security issues shown may include a need for user authentication and data encryption.

In specifying client needs and web services providing them, we need to specify these security requirements, and the clients and services requiring and providing them. Details such as authentication method also need to be specified. At service discovery and integration time, such constraints must be used in searching and validating a discovered service. Another example is support for a business transaction protocol e,g a. long-running transaction over several services. Here, flights may be found and booked, but not confirmed until paid. They may become unavailable or change during the long-running transaction, meaning transactional constraints must be described, services support them, and at discovery and integration time support validated.

Key aspects to describe when advertising web services for others to interact with include those for security models, transaction management, performance measures for operations, faults and exception-handling approaches. In addition, when building web services we may describe data persistency approach, database transactional behaviour for operations, resource utilization, communications infrastructure, monitoring and logging, etc. During discovery and integration, we may need to locate adaptors, transaction managers, and security managers, and compose (or orchestrate) services (Cibran et al 04). We can better support this range of activities by using AOWS to design, develop and deploy web services based systems.

While such a travel planning application and associated web services are often used to illustrate web services concepts, several problems are present when trying to engineer such applications with current web service development approaches and technologies: Key challenges faced by engineers developing such web services-based architectures include:

- How can appropriate web service components be identified and designed?

- How can such web services be appropriately described so clients requiring their functionality can discover and integrate with them? This includes describing extra behavioural characteristics to select between web services with compatible interfaces.
- How can web services be advertised to other components with enough information that useful discovery algorithms can be provided?
- How can adaptors to components be discovered or synthesised and composite component aggregates be found and initialised?
- How can appropriate web service components be modelled, analyzed and verified so that they can be correctly identified and designed? This includes specifying both functional and non-functional behaviour of the services for their clients.
- How can web service descriptions be used to validate that discovered services meet advertised characteristics at run-time? This includes characteristics such as security, performance and transactional behaviour.

We bear these questions in mind and address them when designing and implementing our prototype based on AOWS.

The aim of an Aspect-Oriented Analysis is to define what the system and its components do, including what types of aspects exist in the respective components of the software system and their properties. When dealing with components, this is best achieved by doing an aspect-oriented analysis of the interfaces of the components. The descriptions of the interfaces would simplify and streamline the efforts of the software designer and put him or her on the right track when undertaking the next step of aspect-oriented design for the components.

5.1 Aspect-Oriented Analysis and Design

An Aspect-oriented Collaborative Travel Planner comprising a web-services client, a hotels web services provider, a flights web services provider, adaptors and an Aspect-Oriented Universal Description, Discovery and Integration registry (AOUDDI) were designed and implemented using AOCE. The following sub-sections describe the aspect-oriented components analysis and design of the hotels web service and the flights web service. As aspect-oriented components communicate and interact with each other and other objects through their exposed aspect-enriched interfaces, AOCE places a lot of emphasis on these exposed contractual interfaces. The AOCE analysis and design of the AOUDDI, adaptors and related issues like the Aspect-Oriented Web Services Description Language (AOWSDL) are discussed in the next chapter. The AO-web service providers discussed here are designed such that, besides authorisation rights for security purposes, they need not have any knowledge of the clients consuming it nor do they keep any persistent data about the clients.

5.1.1 Hotels web service

The Hotels web service provides functionalities including performing searches for hotels in particular locations, viewing the facilities in their vicinity, searching for vacant rooms and subsequently making bookings. This web services system was componentised using AOCE techniques. The Travel planner client can locate this web service using the AOUDDI registry and access it based on the information contained in its AOWSDL file.

5.1.1.1 Aspect-Oriented Analysis

The aspect-oriented analysis of component interfaces in the hotels web services system is shown below in Figure 5.2 (a) – (d). The types of aspects involved are abbreviated and clearly shown within double angled brackets preceding the function name. A key to all the abbreviations used for the aspects is included in the diagram. A positive or negative sign preceding the aspect type within the brackets indicates whether the aspect is provided or required. For instance <<+Prs>> means that it is a persistency type of aspect and it is provided by the component. Any sign outside the angled brackets was automatically inserted by the Visio modelling tool. A positive sign is inserted before all functions by the modelling tool to indicate that the visibility of the function is public.

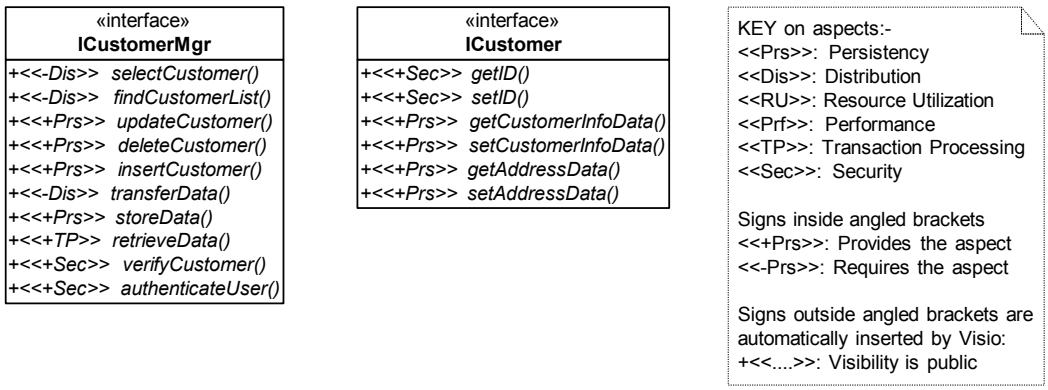


Figure 5.2 (a.): Interfaces of the Customer Component

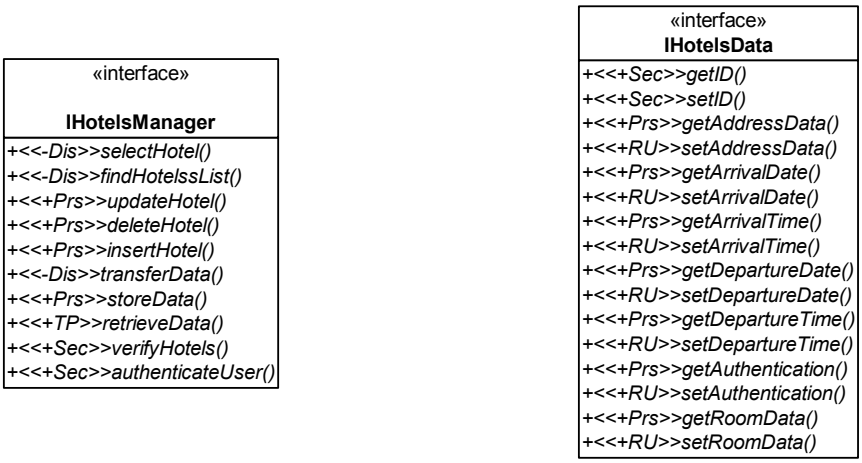


Figure 5.2(b.): Interfaces of the Hotels Component

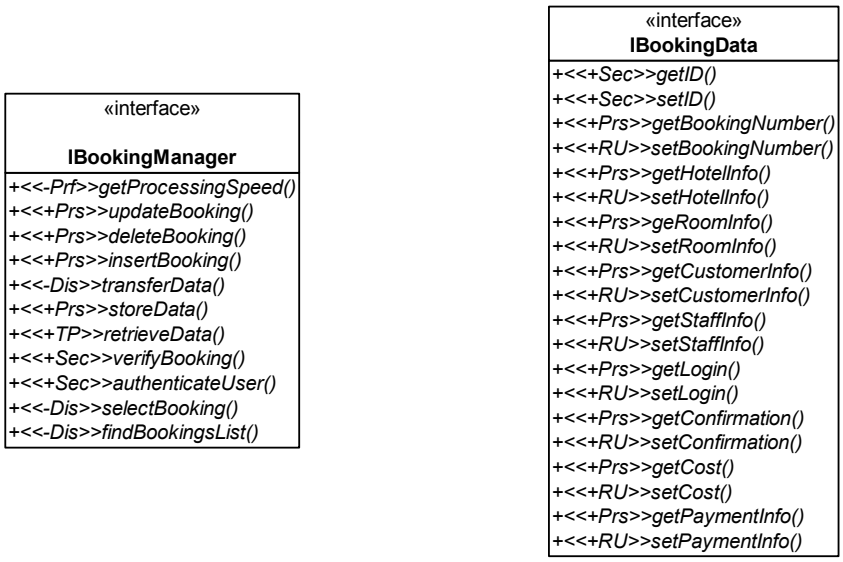


Figure 5.2(c.): Interfaces of the Hotels Booking Component

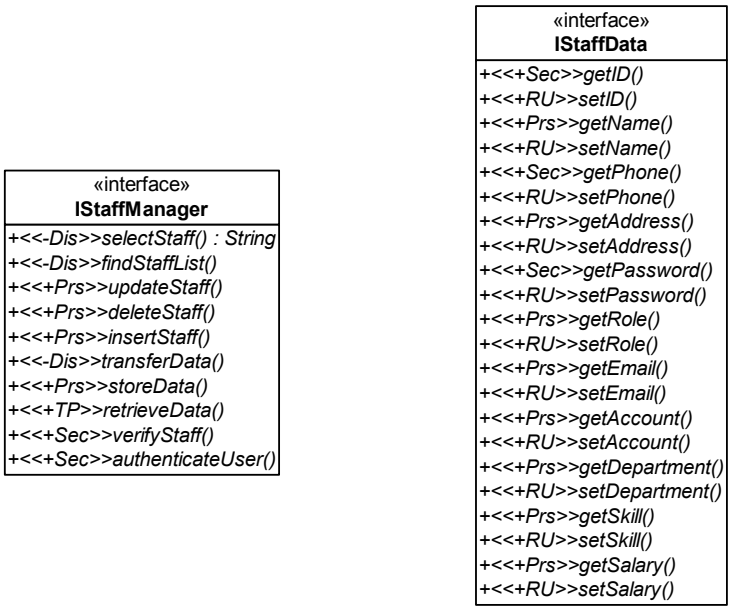


Figure 5.2(d.): Interfaces of the Staff Component

Figure 5.2 (a.) – (d): Aspect-oriented Analysis of components showing their interfaces in the Hotels Web Service System.

The aspect-oriented components and their inter-relationships in the hotels web service system are further shown in Figure 5.3 below. The interfaces belonging to the components are clearly shown in the figure with their aspect types together with the aspect-details sieved out and placed onto the components themselves. As can be seen, the Bookings Component and Hotels Component are both utilised by the web service providers to expose useful aspects to web service requestors, i.e. the travel planner client, so that the client itself need not implement the business logic involved.

The hotels aspect-oriented web service provider also publishes its services to discovery agencies that act as business registries for indexing and locating the web service. The discovery agency used here is the Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI), a prototype registry having

similar functions to a typical UDDI (Cerami 02). The web service clients can discover the aspect-oriented web service providers including their details about their services and location by querying the AOUDDI registry. The aspect-oriented components and their designs are described in more detail in the following sub-sections on component design.

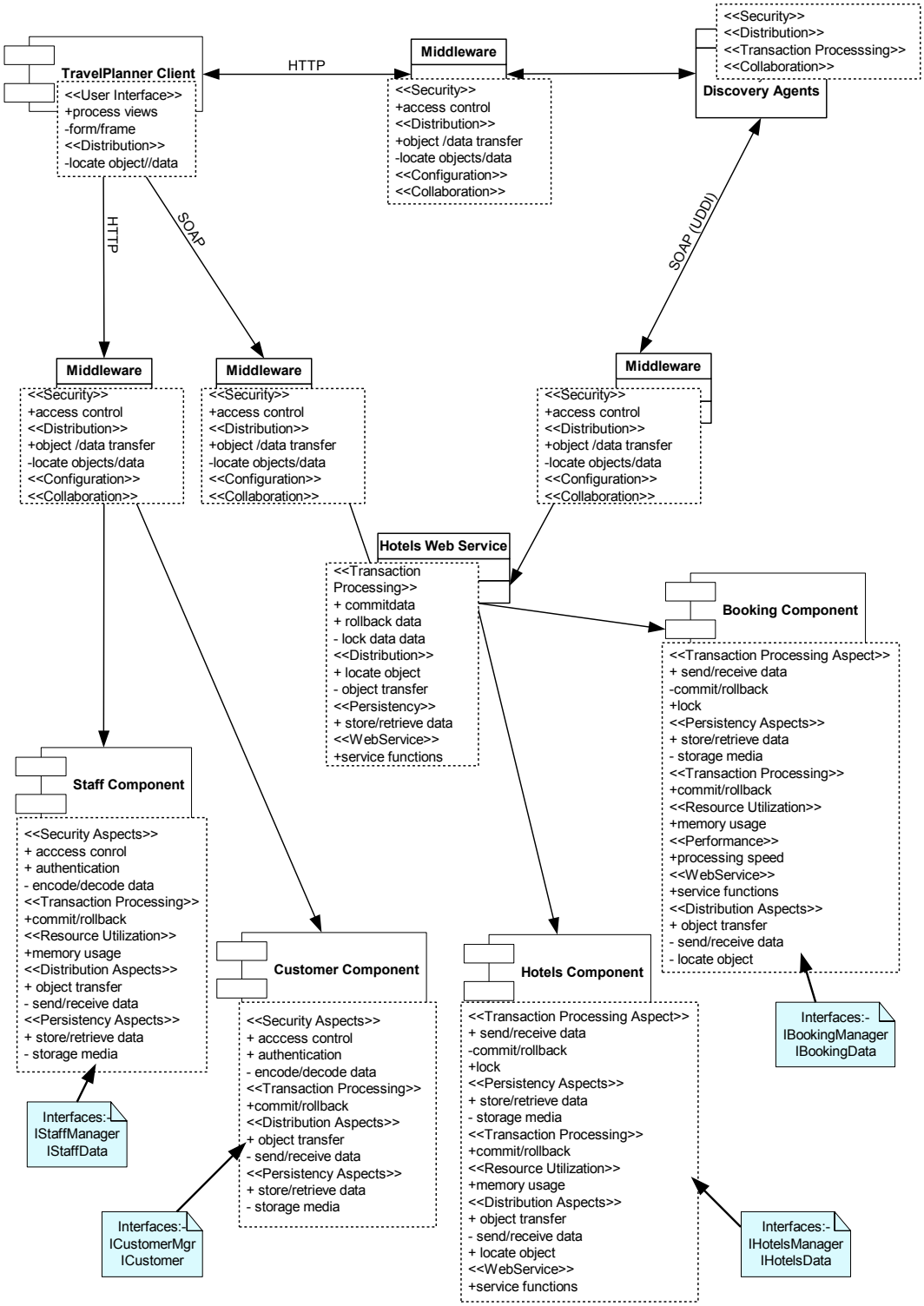


Figure 5.3: Aspect-oriented components and their interactions in the hotels web service system

5.1.1.2 Aspect-Oriented Design of Components

The design of efficient, reusable and understandable components is of paramount importance in any Component Based Development methodology. The aspect-oriented components and their designs are described in more detail in the following paragraphs and with reference to Figures 5.3 – 5.6. The components were designed using Microsoft's Visio, a UML Modelling Tool. Each component exposes its interface so that other components or objects can access and use their functions. These interfaces as such represent a contract that is binding between the participating components.

1. Customer Component

Figure 5.4 below illustrates the design of the Customer component, a very common, versatile and reusable component. It can be easily reused in other systems that require a customer component because of its generic and common nature, i.e. most customer objects essentially have the same attributes and functions with some slight modifications which can easily be achieved. The ICustomer interface which is implemented as the CustomerData object is designed to wrap relational database information. It has a lot of inbuilt methods to retrieve and set the values of various instances of the class as shown in the diagram as getter and setter methods. The CustomerData object also uses the AddressData and CustomerInfoData objects to manipulate customer information. The ICustomerMgrImp which is the implemented class of the ICustomerMgr interface manages all the information of the customer in databases and utilises the CustomerData object. This includes all updating, inserting, selecting, and deleting functions of the aspects concerned necessary to manipulate data in the customer databases.

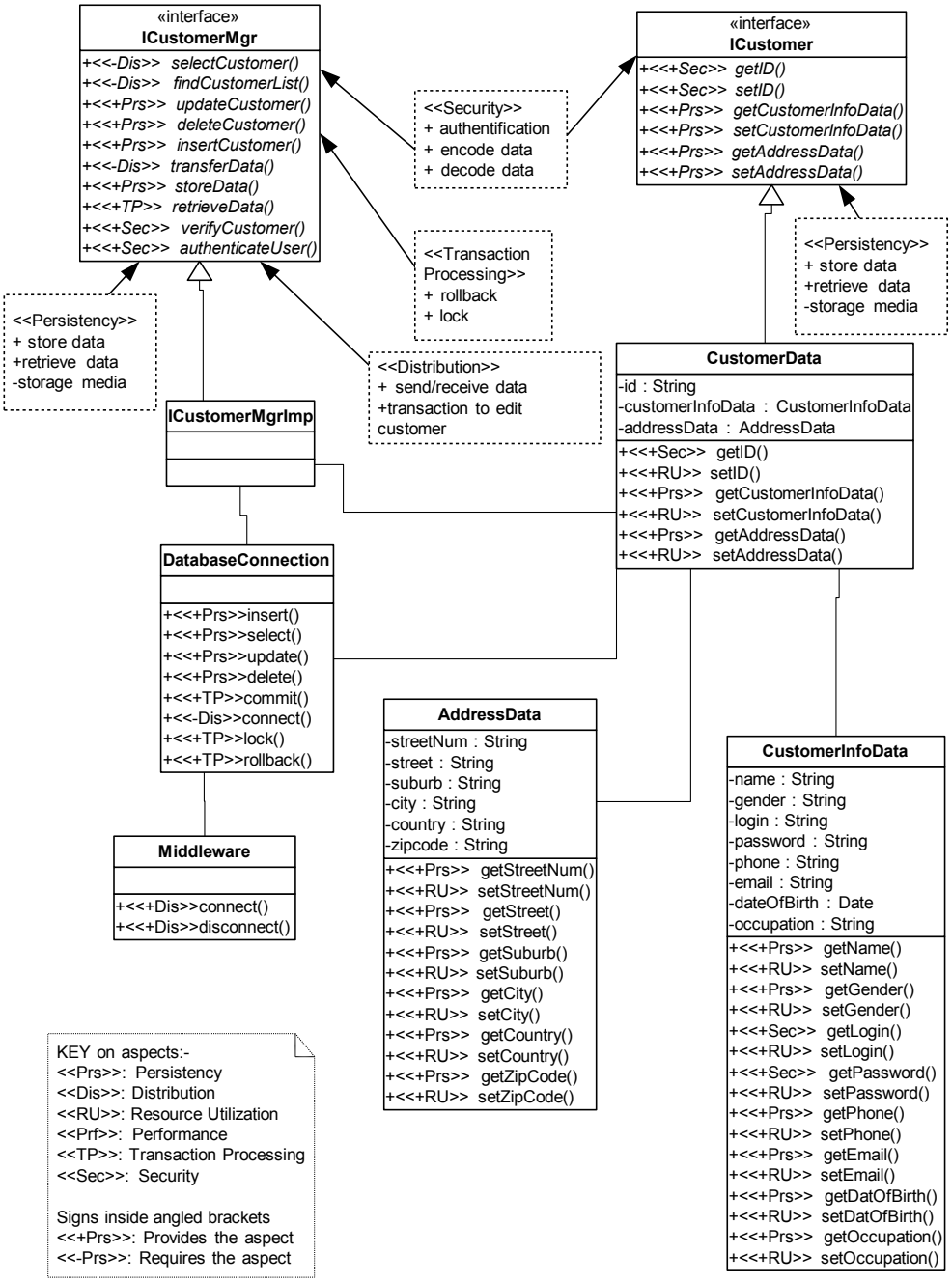


Figure 5.4: Customer Component with aspects information

2. Hotels Component

Figure 5.5 below illustrates the Hotels Component that contains all the information, attributes and functions necessary to access, retrieve and edit the hotels and rooms data in the database. It has a IHotelsManager interface that is implemented as IHotelsManagerImp object. This object is designed to manage all details that are required by the clients to view hotels and rooms, including information relating to their costs, availability and facilities. It also allows for the editing of all this data stored in the databases for those with authority to do so. The HotelData object which implements IHotelsData is designed to wrap relational database information about the hotels. All these objects in the hotels component together with their attributes and functions were defined and iteratively refined from requirements engineering and analysis. The Hotels component is easily pluggable into systems that require it because its interfaces are clearly defined using aspectual features which are easily understandable. These aspects also better characterise and categorise the hotels component enabling dynamic discovery of its advertised services to be performed.

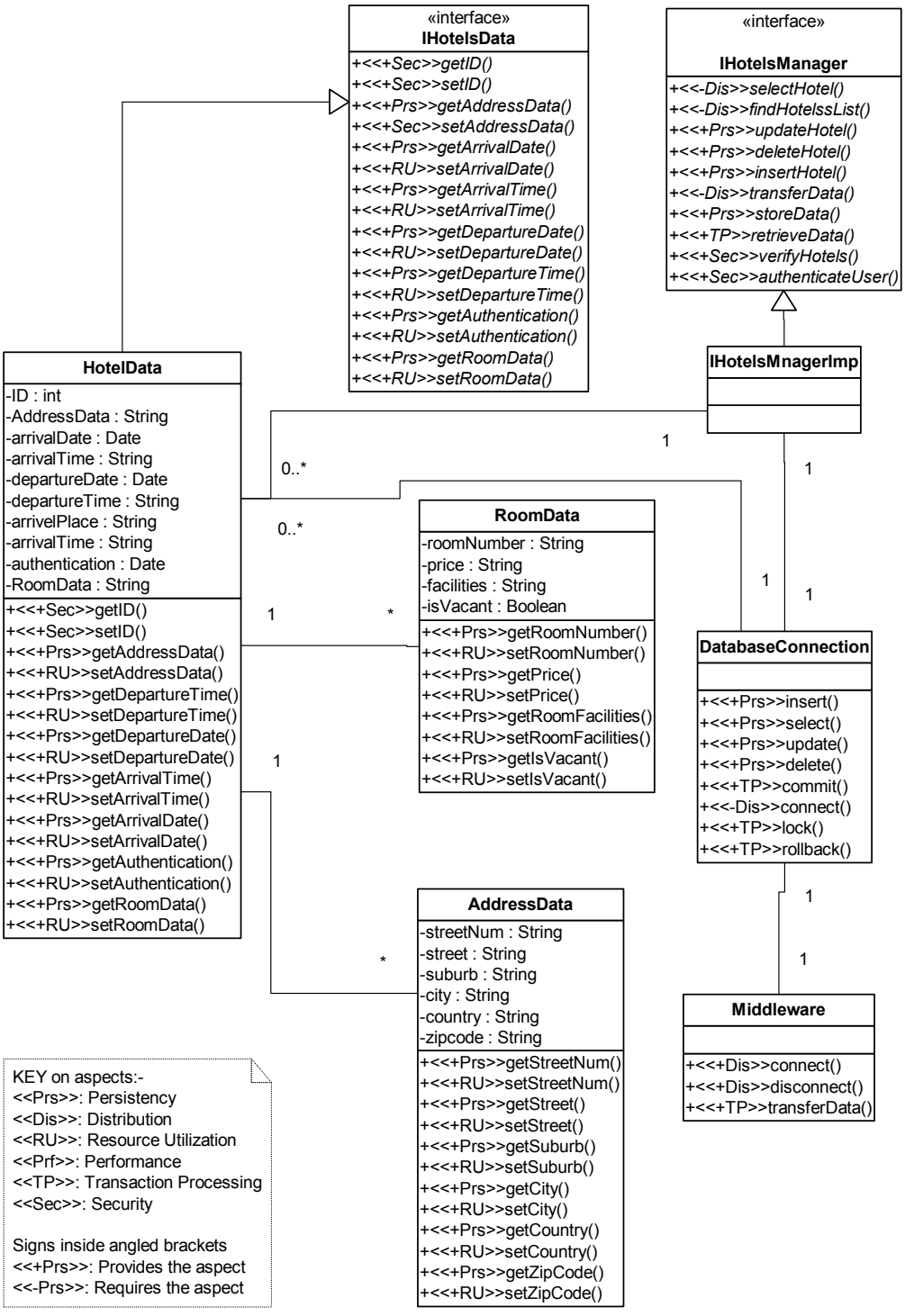


Fig 5.5: Hotels Component with aspects information

3. Hotels Booking Component

Figure 5.6 below shows the Hotels Booking Component which is used by clients to make bookings of the rooms in a particular hotel. It exposes the IBookingManager interface for software systems to access its functions and attributes. The IBookingManagerImp object implements this interface and it is designed to manage all information about bookings in the databases. The BookingData object is designed to wrap relational database information, its attributes and access methods. It acts as a helper object to the IBookingManagerImp so that the component is more modularised. Also the aspects in the IBookingManager interface enriches the component by giving it high level descriptors and at the same time addressing the issue of cross-cutting within the component.

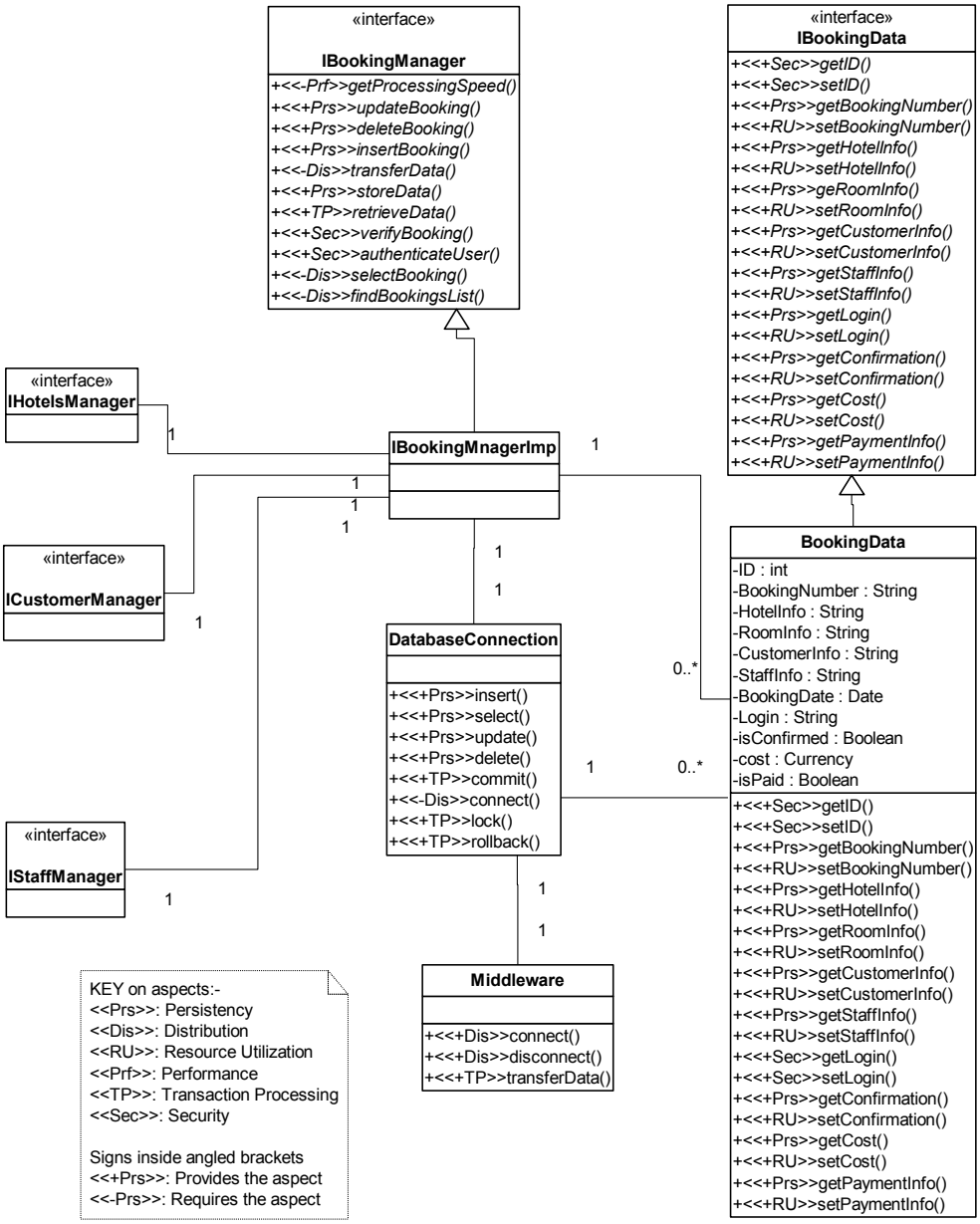


Figure. 5.6: Hotels Booking Component with aspects information

4. Staff Component

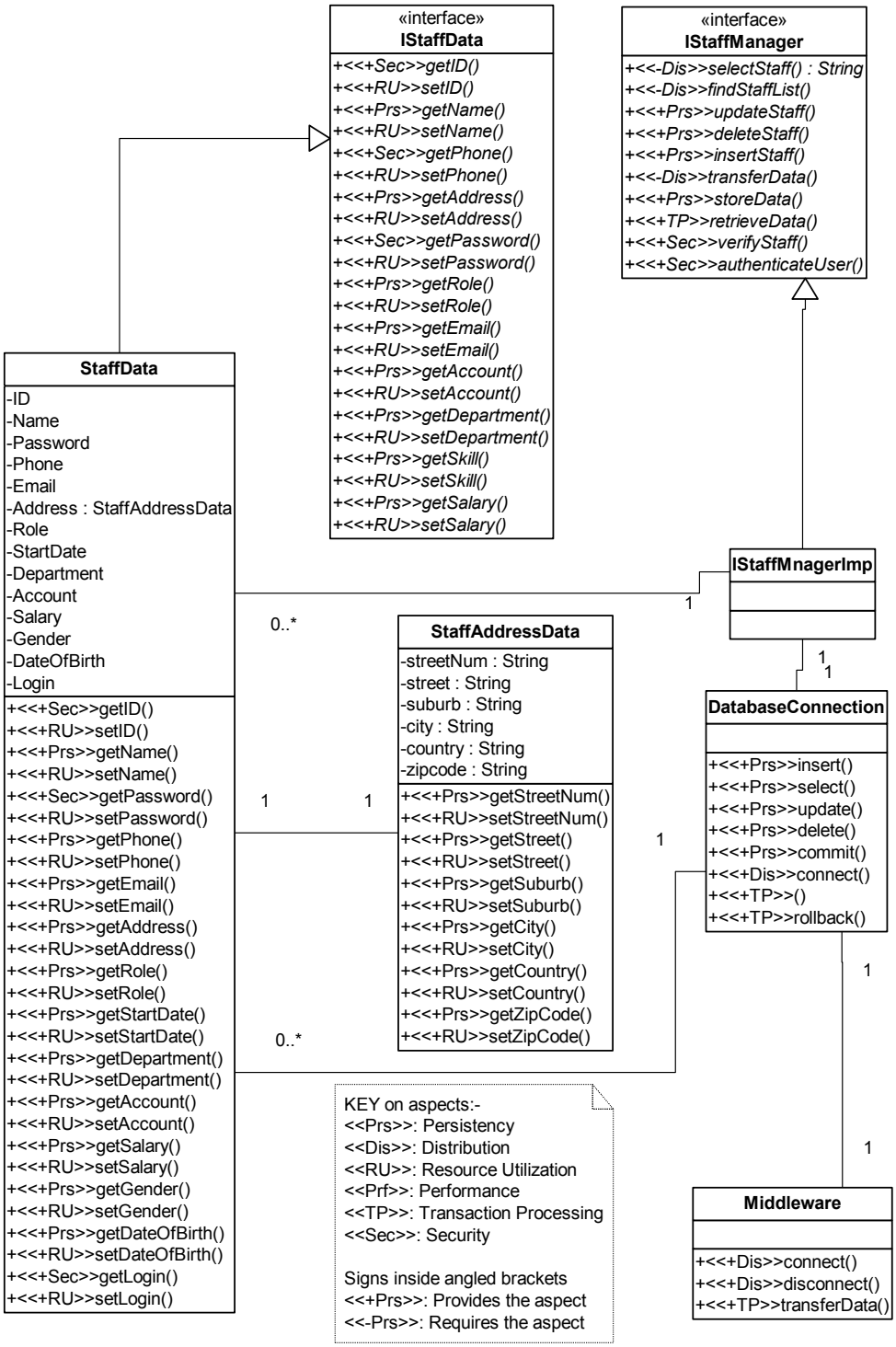


Figure 5.7: Staff Component with aspects information

The staff component shown in figure 5.7 above is used to access, retrieve and manipulate staff information. It has an IStaffManager interface that is implemented by the IStaffManagerImp object. This object wraps around all relational database information, the attributes and access methods relating to the staff. The information about all staff is maintained in the back end databases. These databases also contain all staffs' level of permission of access and other personal information relating to them. The IStaffData interface which is implemented as the StaffData object is also designed to wrap relational database information. This also includes all updating, inserting, selecting, and deleting aspectual functions necessary to manipulate staff data in databases. The StaffData object uses the StaffAddressData object to manipulate staff information. The staff component also uses the IStaffManager interface to communicate or interact with other components or software systems to manage the staff information.

5.1.2 Flights web service

The aspect-oriented analysis of components showing their interfaces in the flights web services system is shown below in Figure 5.8 (a) – (b). The customer and staff components are the same as those in figure 5.3(a) and 5.3(d) respectively and are already discussed above. These aspect-oriented components and their interactions in the flights web service system are further illustrated below in Figure 5.8. As shown, the Bookings Component and Flights Component are both utilised by the web service providers to expose useful aspects to web service requestors, i.e. the travel planner client, so that this client itself need not implement the business logic involved. The interfaces belonging to the components are clearly shown in Figure 5.9 with their aspect types together with the aspect-details sieved out and pasted onto the components themselves. This flights aspect-oriented web service provider also publishes its services to discovery agencies that act as business registries for indexing and locating the web service. The web service clients can discover the web services and their location and subsequently integrate with them by using the discovery agents.

Figure 5.8 (a.): Interfaces of the Flights Component

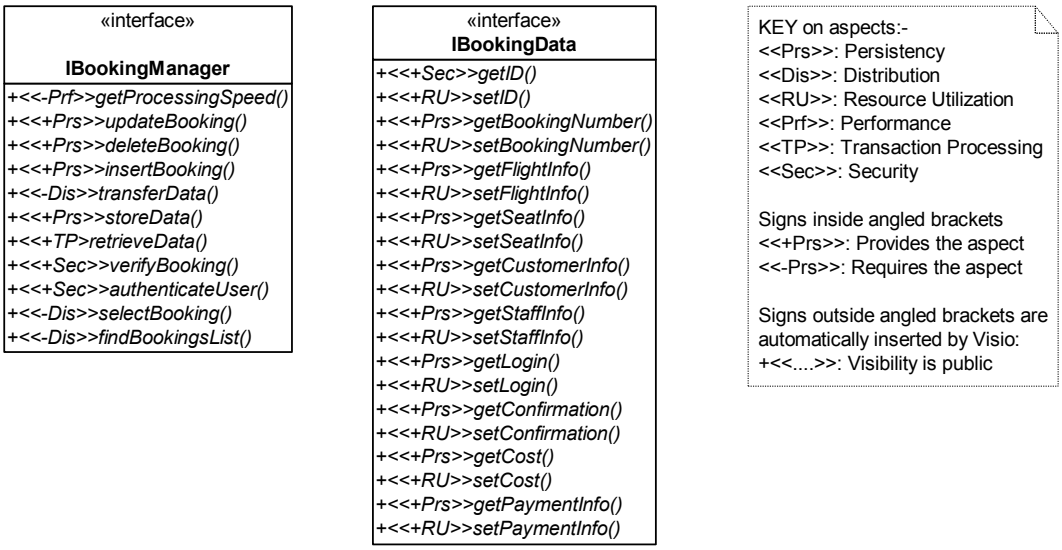


Figure 5.8 (b.): Interfaces of the Flights Booking Component

Figure 5.8(a) – (b): Aspect-oriented Analysis of components showing their interfaces in the Flights Web Service System.

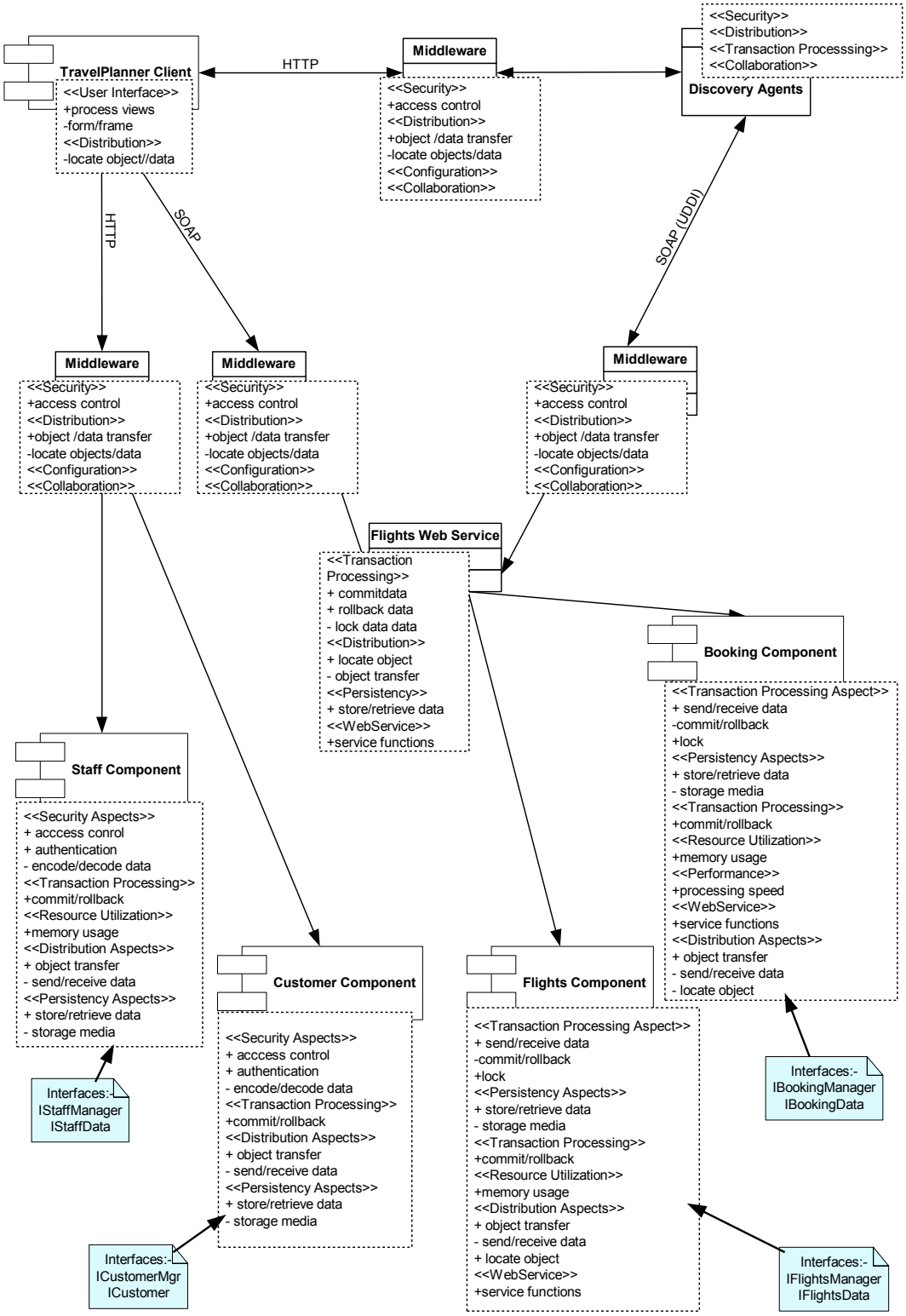


Fig 5.9: Aspect-oriented analysis of Flights Web Service components

1. Flights Component

The Flights Component shown in Figure 5.10 contains all the information, attributes and functions necessary to extract or manipulate the data relating to flights and seats in the database. It has a IFlightsManager interface that is implemented as IFlightsManagerImp object. This object is designed to manage all details that are required by the clients to view flights information, including all seats information and subsequently make reservations for seats that are available. The FlightsData object is designed to wrap relational database information about the flights. All these objects in the flights component together with their attributes and functions were defined and iteratively refined from requirements engineering and analysis. The Flights component is easily pluggable into systems that require it because its interfaces are clearly defined using rich aspectual descriptions.. The aspects make the components more understandable, better characterised and categorised.

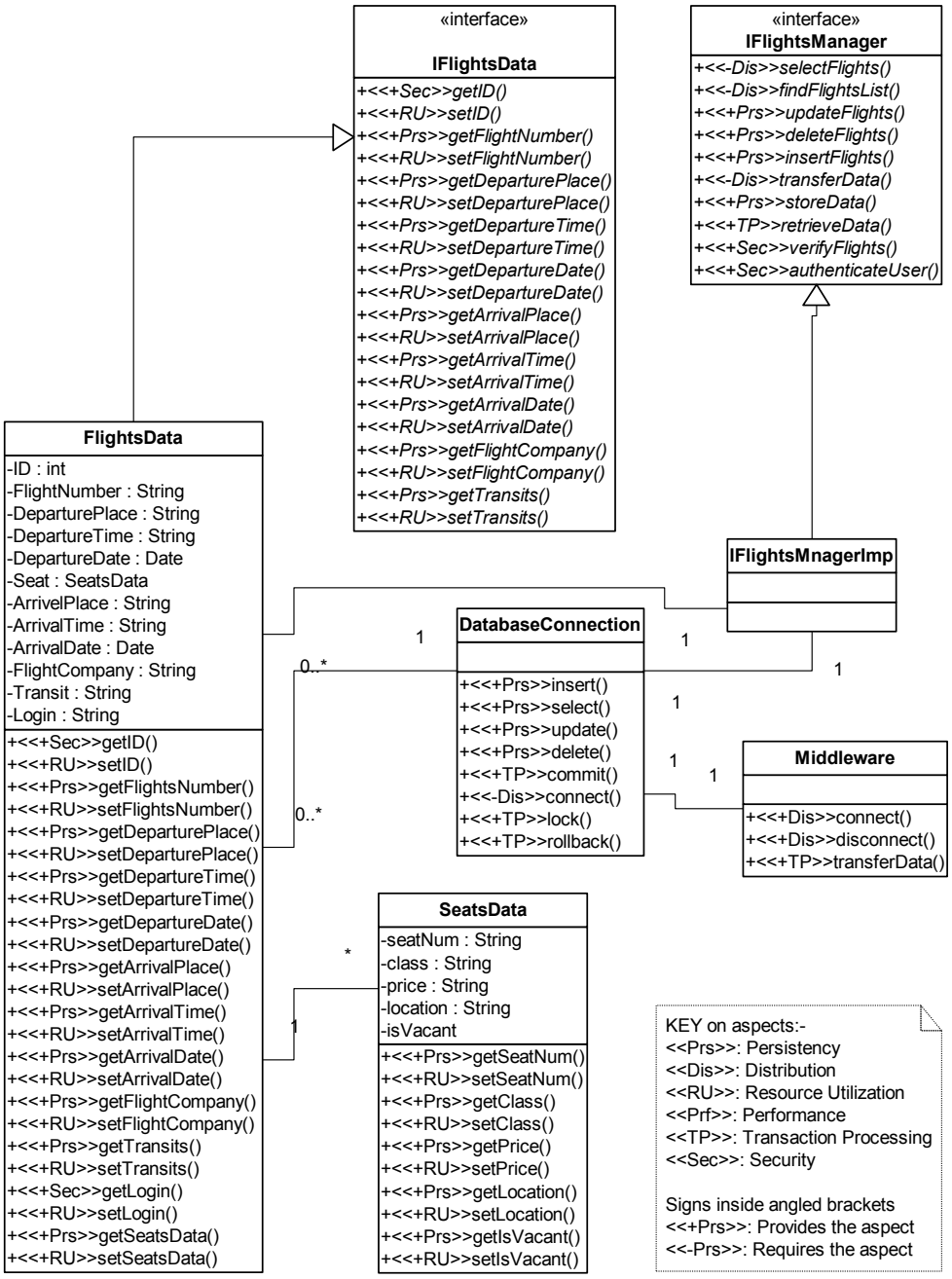


Figure 5.10: Flights Component with aspects information

2. Flights Booking Component

The Flights Booking Component shown in Figure 5.11 is used by clients to make bookings of the seats on a particular flight. It exposes the IBookingManager interface for software systems to access its functions and attributes. The IBookingManagerImp object implements this interface and it is designed to manage all information about bookings in the databases. The BookingData object is designed to wrap relational database information, its attributes and access methods. It acts as a helper object to the IBookingManagerImp so that the component is more modularised. Also the aspects in the IBookingManager interface enrich the component by giving it high level descriptors and at the same time addressing the issue of cross-cutting within the Flights Booking component.

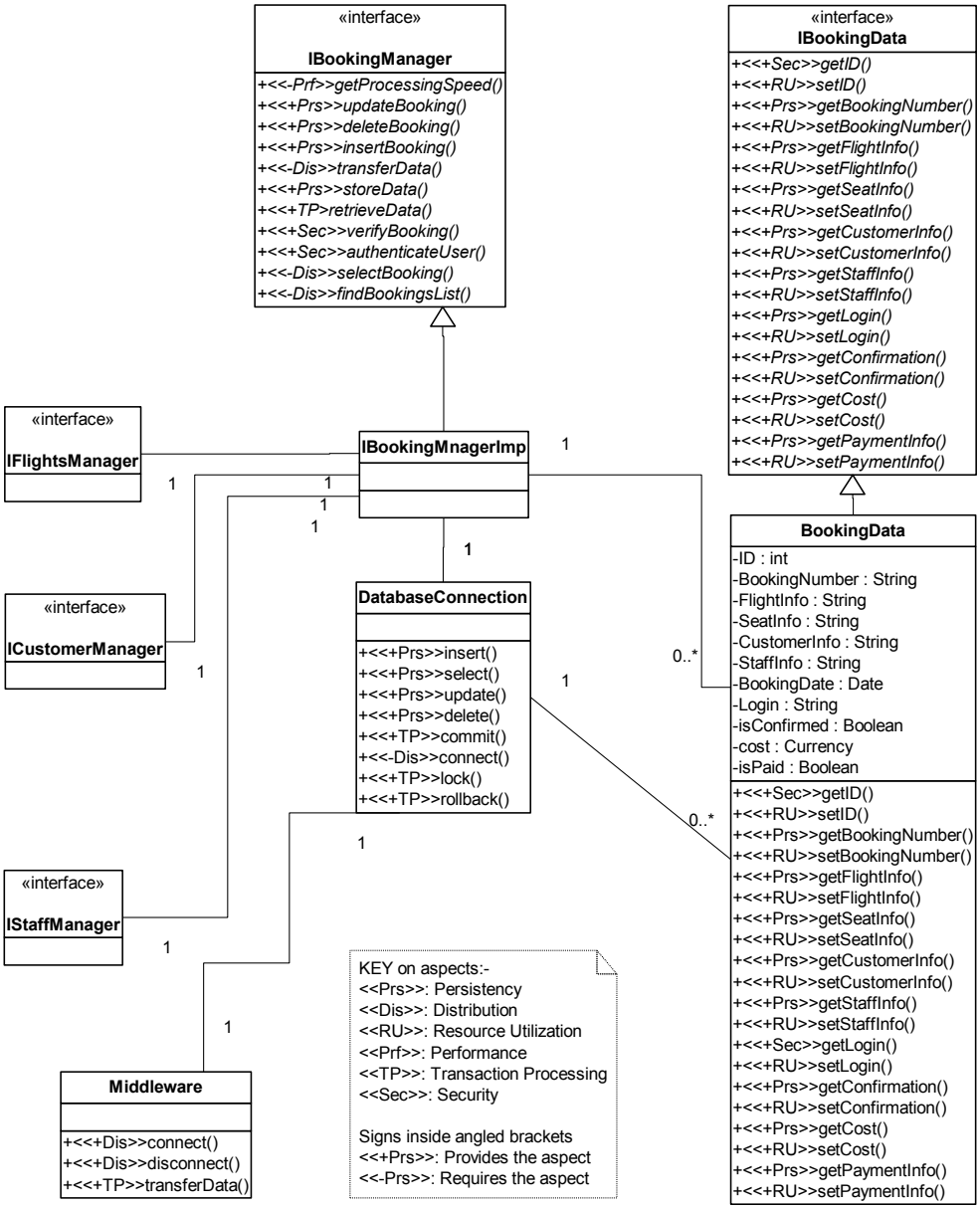


Fig 5.11: Flights Booking Component with aspects information

5.2 Sequence Diagrams with Aspects

A sequence diagram shows a series of interactions between objects arranged in a time sequence (Bennet et al 99). They represent dynamic systemic behaviours. Using the AOCE methodology we have additional aspectual characteristics included in the sequence diagram. These aspects increase the understandability of the sequence diagram by increasing the characterisation of the functions.

Figure 5.12 below shows the sequence diagram of the hotels web services when a client invokes method calls to search for vacant rooms. The aspects involved are clearly defined just below the objects in the sequence diagram. These aspectual features enable software engineers to better understand the sequence of events and objects involved.

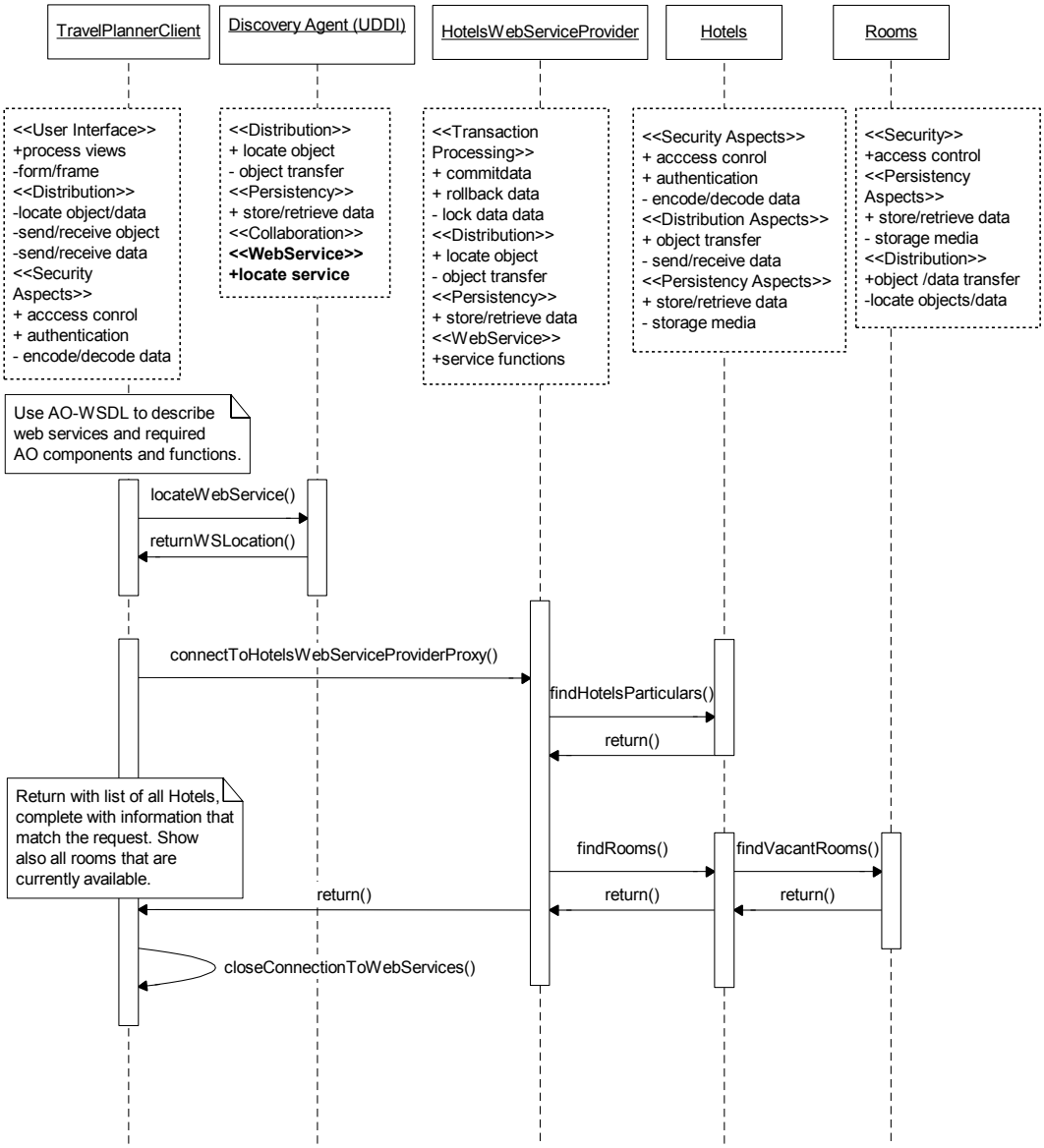


Fig 5.12: Hotels Web Services Sequence Diagram

The sequence diagram of the flights web services to search for available seats is depicted in figure 5.13 below. The aspects involved are also clearly defined just below the objects in the sequence diagram. The collaborative TravelPlanner is the client, i.e. it implements the web service requestor to consume the web services provided by the Flights web service. As such, on its own it has no functionality to

carry out the work of searching for flights and making reservations for the seats available. It uses the aspectual functions exposed by the Flights web service to do this.

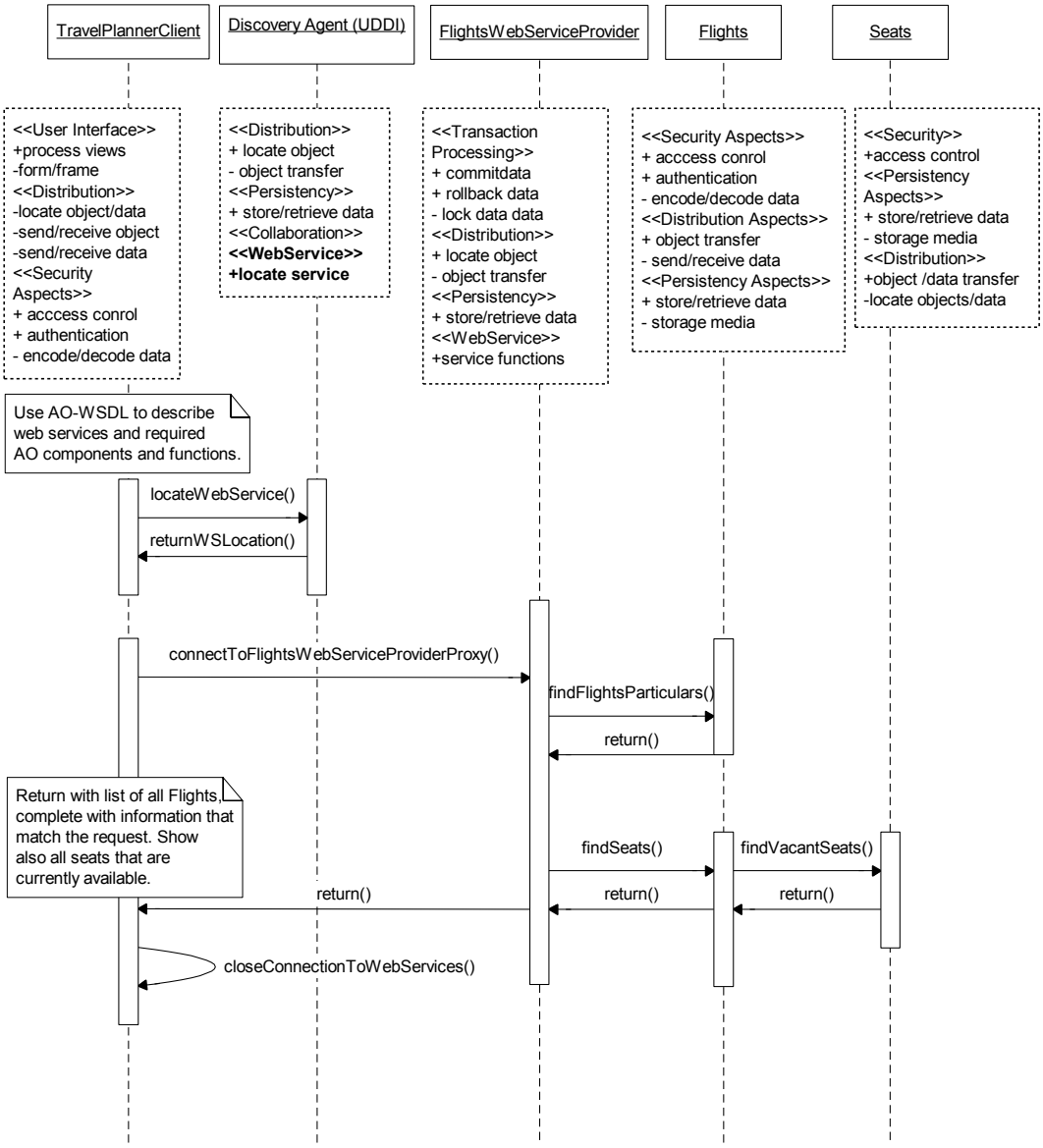


Figure 5.13: Flights Web Services Sequence Diagram

5.3 AO System Architecture for Travel Planner

Figure 5.14 below shows the Aspect-Oriented System Architecture for the collaborative Travel Planner based on the distributed architecture of web services systems. It shows the web service providers and the web service requestors. Additional web service providers, e.g. a car rentals or trains web service, can be included and similarly invoked by the web service requestors. An interlinking object called the AOConnector uses Inversion-of-Control mechanisms (Fowler 04) and is used to connect the requester to the rest of the subsystems in AOWS. This has the advantage of making the clients more lightweight and easier to construct. Both the AOConnector and its Inversion-of-Control mechanism are discussed in-depth in Chapter 7. Also shown in the figure are the databases that store persistent data according to the type of service. The Aspect-oriented Universal Description, Discovery and Integration (AOUDDI) registry provides the functionalities for registering the web service providers. It also supports subsequent discovery and integration of these aspect-oriented web service providers by web service requestors.

The aspect-oriented web service providers use an extended version of the Web Services Description Language (WSDL) (Christensen et al, 01) called the Aspect-oriented Web Services Description Language (AOWSDL) that contains elements enriched with aspects information. These aspects' information better characterises and categorises the web services exposed API's in the AOWSDL document. AOWSDL and AOUDDI are covered in more detail in the next chapter. The Simple Object Access Protocol (SOAP) is the protocol used to facilitate communication between the various parts in the distributed system.

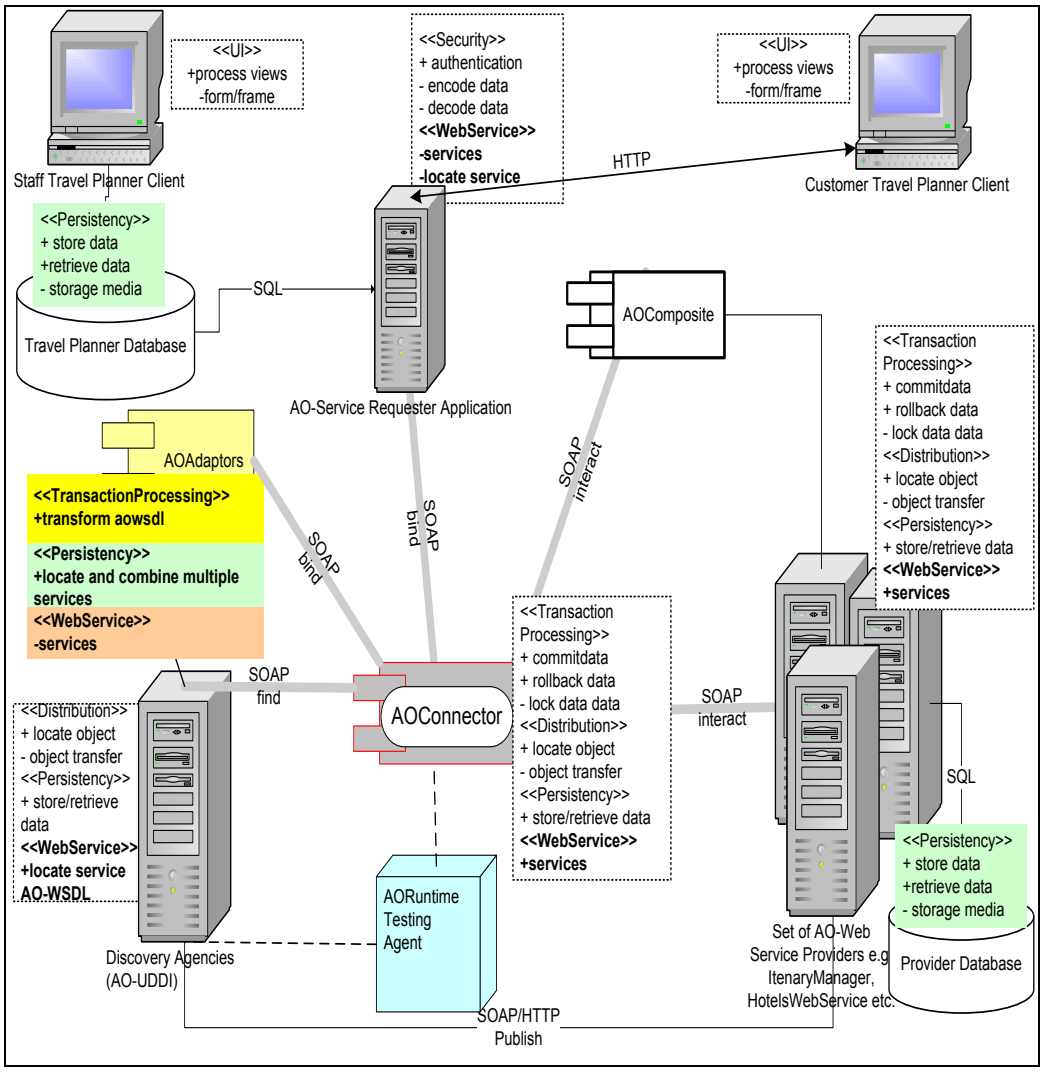


Figure 5.14: Aspect-Oriented System Architecture for Travel Planner

Microsoft's Visio was used as the software modelling tool to carry out analysis and design of the collaborative Travel Planner which included use case diagrams, class diagrams, deployment diagrams, sequence diagrams, component designs and the system architecture diagram. These diagrams capture and portray a software system's functionalities and business logic very well and as such give users a very good understanding about modelling software systems. However this tool does not have the comprehensive inbuilt facilities to show aspectual information in a more visual

manner. We went around this by a variety of techniques, including the use of more textual descriptions in the class diagrams and inserting additional notes.

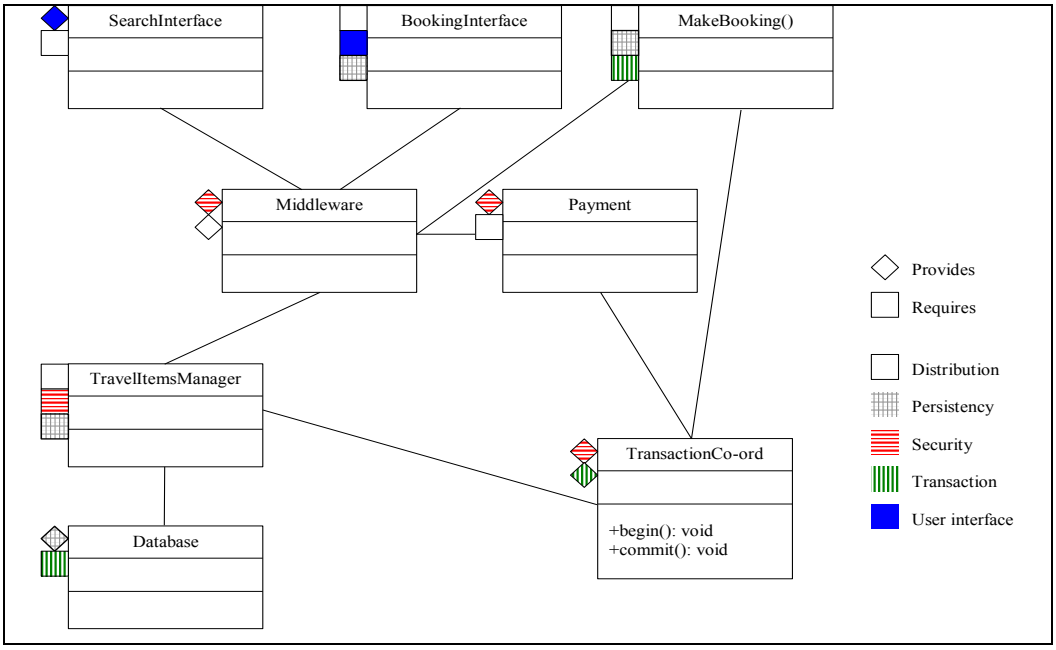


Figure 5.15: An example of more visually enhanced diagram showing interrelationships between different components.

One of our future plans is to develop a comprehensive modelling tool for AOCE to model aspect-oriented components and aspectual information. Besides textual representations, this tool will also have more visual representation features like the use of different icons, colours and patterns to represent the various components and aspects. This is beyond the scope of the present thesis. Figure 5.15 above illustrates a simple example of how this can be achieved to depict the interrelationships between different components where each component is depicted as a traditional UML class. The different types of patterns in the boxes indicate the different aspects and also whether the aspects provide or require crosscutting information from another component. A square box indicates that the aspect requires the crosscutting

information while a diamond shaped one provides it for another component. The tool we have in mind will of course produce additional, more elaborate and comprehensive diagrams and designs than the example shown here.

5.4 Summary

Aspect-Oriented Analysis was carried out to define what the Travel Planner system and its components were required to do, including what types of aspects exist in the respective components of the web services system. When dealing with components, this is best achieved by doing an aspect-oriented analysis of the interfaces of the components. The descriptions of the interfaces will simplify and streamline the efforts of the software designer and make it more understandable when doing the aspect-oriented design for the software components.

Microsoft's Visio, a UML software modelling tool was used to carry out the analysis and design of the collaborative Travel Planner which included use case diagrams, class diagrams, deployment diagrams, sequence diagrams, component designs and the system architecture diagram. These diagrams capture and portray the software system's functionalities and business logic very well and as such give developers a very good understanding about modelling software systems using AOCE techniques.

6 Describing and Locating Aspect-Oriented Web Services

Web service providers use web services discovery agencies, for instance the Universal Description, Discovery and Integration (UDDI) (Newcomer 02, UDDI_website 05) registry to publish their service documents containing details about their location, services and related technical details to enable discovery and integration. The UDDI registry is a business and service registry which is also an open industry initiative to enable businesses utilising web services technology to describe, discover and integrate with each other (Ran 03, Adams and Boeyen 02). Web service providers can make use of the Web Services Description Language (WSDL) (Cerami 02, Christensen et al 01), which is a very popular service document supported by W3C, written in the XML format to describe their services.

A more complete and comprehensive description language that contains references to the aspectual elements and their details is required to describe web services built using AOCE techniques. This is due to the fact that aspect-oriented web services are richer, more characterised, highly modularised, aspectised and componentised, and as such, an extra set of XML grammar was necessary to be formulated to describe them together with all the aspects and their details. We aim to better describe, characterise and categorise web services components within discovery documents so that clients can more easily discover the service that best meets their needs. In this chapter we describe the WSDL, its limitations and how to overcome them using an extended and aspect-enriched form of WSDL called Aspect-Oriented Web Services Description Language (AOWSDL). An Aspect-Oriented UDDI (AOUDDI) that was developed

specially to describe, locate and integrate clients with the aspect-oriented web services is also described here.

6.1 Web Services Description Language

Web service providers need a way to describe their services using a format that is both comprehensive and understandable to the service requestors. It should provide all the details necessary to interact with the service, including message formats, transport protocols and location (Christensen et al 01). The nature of this web services description is such that it hides the implementation details of the service so that it can be used independently of the programming language, hardware or software platform on which it is implemented.

Web Services Description Language (WSDL) is the currently used XML format for describing web services. It has six major elements (Cerami 02 and Newcomer, 02). These are the definitions, types, message, portType, binding and service elements. Figure 6.1 below shows the hierarchy and a brief description of the main elements within the WSDL document.

(1.) **<definitions>** The root element of a WSDL document.

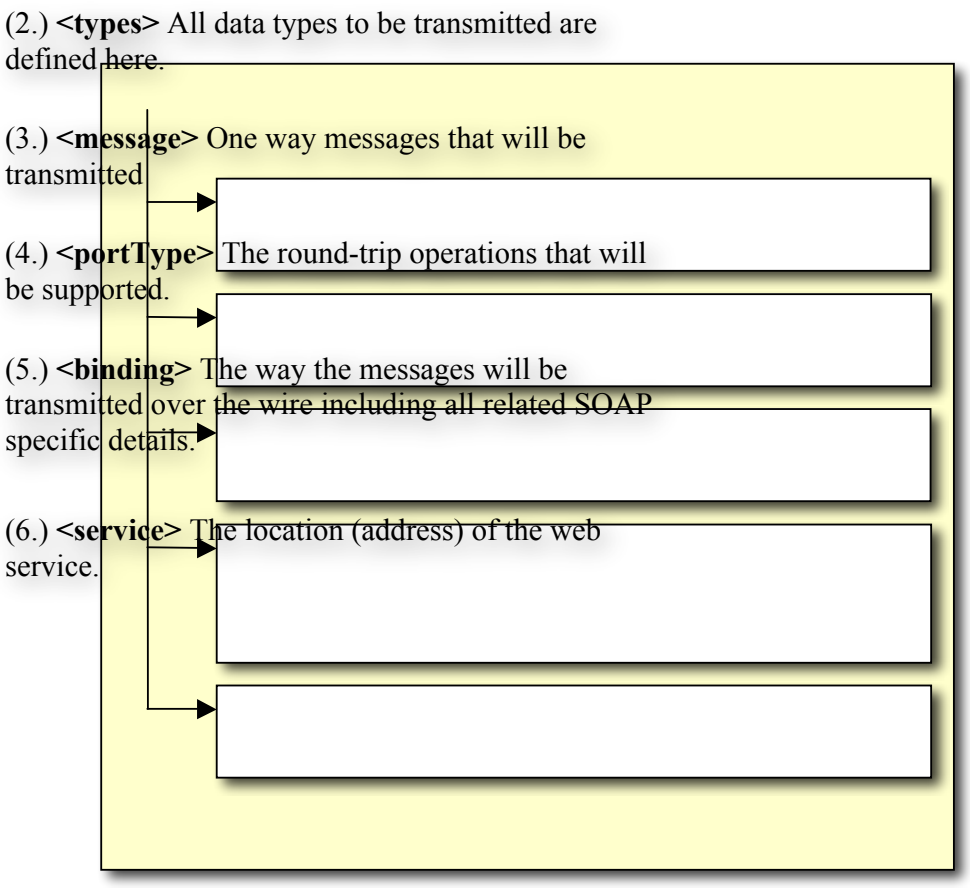


Figure 6.1: WSDL document showing the hierarchy of its main elements (Cerami 02).

Collectively, these elements define the interface information describing all publicly available services; data type information for all message requests and message responses; binding information about transport protocol to be used and lastly, address information for locating the specified service. These six elements and their specific functions are described below.

(1.) "definitions" element

This is the root element of all WSDL documents. It contains the web service's name, declares multiple namespaces and contains service elements.

(2.) "types" element

The types element describes all data used, including any complex types of data, between the client and the server. WSDL is not tied to a specific typing system. But if a service only uses XML Schema built-in simple types, e.g. integers, doubles and strings, then the types element may be left out. This is because WSDL uses the W3C's XML Schema as its default choice.

(3.) "message" element

This element describes a one-way message. It actually defines the name of the message and may contain any number of message "part" elements. These "part" elements can refer to message return values or message parameters. As such message elements refer to single message requests or single message responses.

(4.) "portType" element

The portType element describes a complete one-way or round-trip operation. This element combines multiple message elements to form their complete operations. Here a request and a response message can be combined to form a single request-cum-response operation. A portType element can define any number of such operations. There are four basic patterns of operation (Cerami 02) supported by WSDL, i.e. one-way operations, request-response operations, solicit-response operations and notification operations. We used all these four patterns in our AOWS-based Travel Planner prototype that we developed and they are illustrated and explained in the following paragraphs, with examples extracted from our prototype.

I. One-way operations:

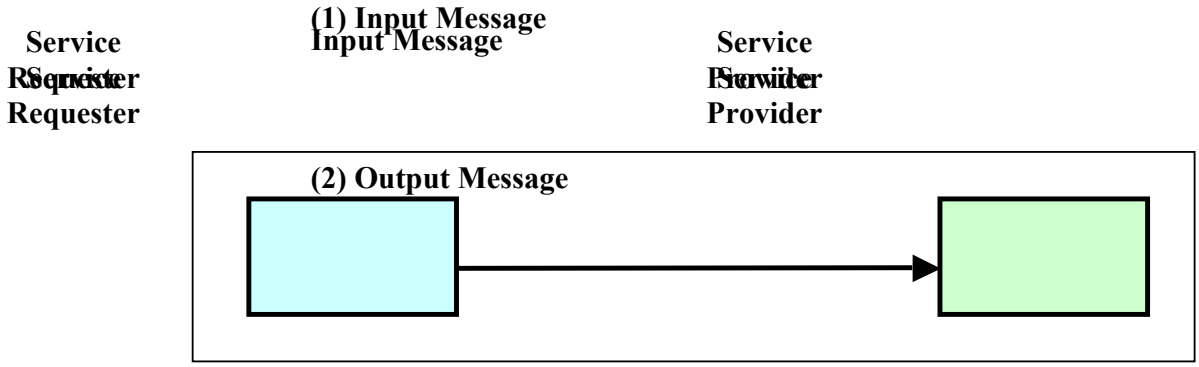


Figure 6.2: Illustration of a one-way operation

Here the service receives a message and the operation as such has a single input element, as shown in Figure 6.2. In our prototype, this type of operation is used by the Travel Planner client to send a one-way SOAP message to the AOUDDI containing the list of the service providers that it is currently consuming. The UDDI keeps a record of this in its database so that, for example, if one of these providers are down or if a new service provider with similar functionalities becomes available, the AOUDDI can notify the client immediately to take appropriate action.

II. Request-response operations:

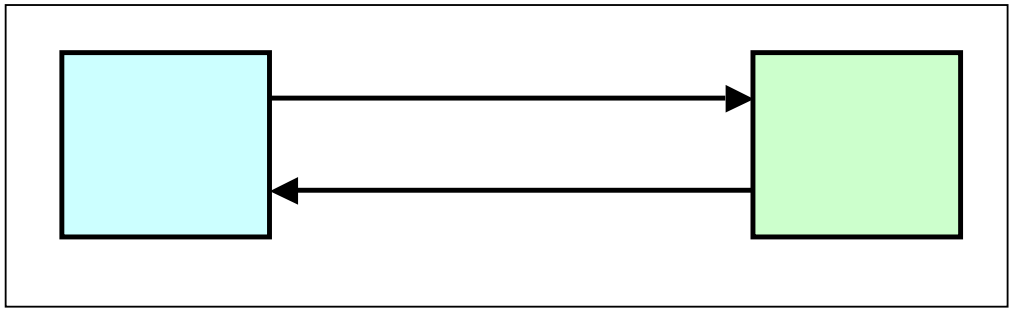


Figure 6.3: Illustration of a request-response operation

The service receives a message and sends a response, as shown in Figure 6.3. The operation has one input element, followed by one output element. For example, the Travel Planner client requests about the details of the rooms in a particular hotel by sending a message to an appropriate service provider and the provider replies back with the relevant details.

III. Solicit-response operations:

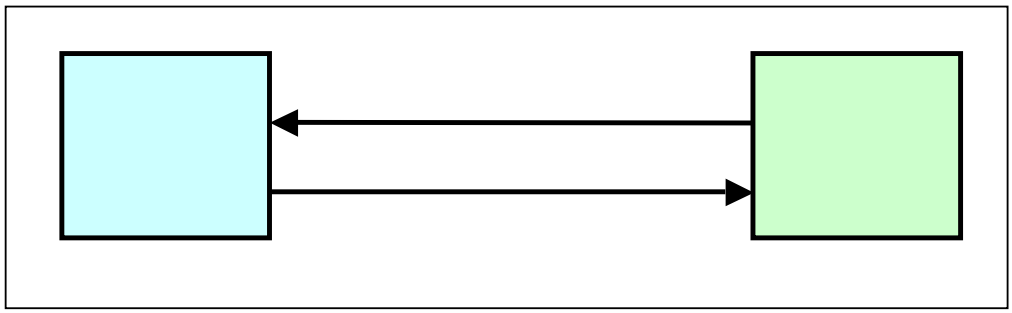


Figure 6.4: Illustration of a solicit-response operation

As shown in Figure 6.4, the service sends a message and receives a response. The operation has one output element, followed by one input element. An example from our AOWS prototype is when the service provider publishes its AOWSDL by sending it to the AOUDDI registry. The AOUDDI stores the AOWSDL, registers the provider and sends the provider an acknowledgement, together with a unique Publish Identity Number, stating that it has successfully registered with that particular AOUDDI. This form of solicit-response operation is also used in call-back type functions by providers to request for more information from clients, i.e. when the information supplied is insufficient to be processed.

IV. Notification operations:

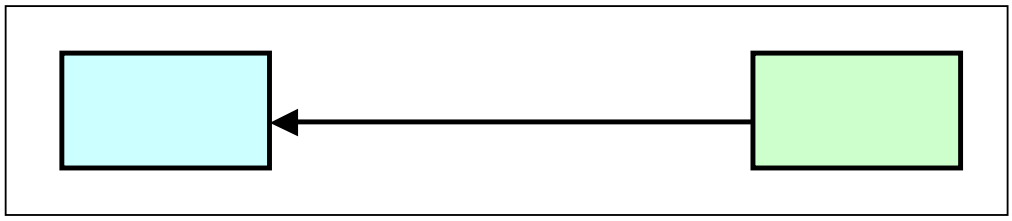


Figure 6.5: Illustration of a notification operation.

As shown in Figure 6.5, the service provider sends an output message, as such the operation has a single output element. The direction is the reverse of the one-way operation explained in item (I) above. In our collaborative Travel Planner, an instance of this type of operation is fired when a new service is registered in the

AOUDDI and becomes available. Based on the information stored in its database, the AOUDDI will notify all the clients that are consuming similar services that a new service has become available by sending out this type of output operation. The respective clients can then check out the new service by carrying tests etc. and consume it if it is found to be more suitable or useful.

Besides these four elements, an optional fault element can also be specified within the operations to encapsulate errors.

(5.) "binding" element

A binding element defines how the service will be implemented on the wire. The Simple Object Access Protocol (SOAP) and Hyper Text Transfer Protocol (HTTP) that describe how to achieve this implementation goes here.

(6.) "service" element

This element defines the address for invoking the services described. It normally includes the Uniform Resource Locator (URL) for invoking SOAP services.

Besides these six major elements, WSDL also defines 2 kinds of utility elements viz. the "documentation" and "import" elements. The documentation element provides human- readable documentation about anything pertaining to the web service. It is very flexible and can be included within any other WSDL element. The import element is used to import other WSDL documents or XML Schemas into the parent WSDL document. This allows for more modularity and clarity.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
       targetNamespace="http://localhost/WSDLWebService/wsdl/HelloService.wsdl"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:tns="http://localhost/WSDLWebService/wsdl/HelloService.wsdl"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">

       <message name="GreetHelloRequest">
           <part name="firstName" type="xsd:string"/>
       </message>
       <message name="GreetHelloResponse">
           <part name="greeting" type="xsd:string"/>
       </message>

       <portType name="Hello_PortType">
           <operation name="greetHello">
             <input message="tns:GreetHelloRequest"/>
             <output message="tns:GreetHelloResponse"/>
           </operation>
       </portType>

       <binding name="Hello_Binding" type="tns:Hello_PortType">
           <soap:binding style="rpc"
             transport="http://schemas.xmlsoap.org/soap/http"/>
           <operation name="greetHello">
             <soap:operation soapAction="greetHello"/>
             <input>
               <soap:body
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="urn:examples:helloservice"
                 use="encoded"/>
             </input>
             <output>
               <soap:body
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="urn:examples:helloservice"
                 use="encoded"/>
             </output>
           </operation>
       </binding>

       <service name="Hello_Service">
           <documentation>A WSDL file for HelloService used for greetings</documentation>
           <port binding="tns:Hello_Binding" name="Hello_Port">
             <soap:address
               location="http://localhost:8080/soap/servlet/rpcrouter"/>
           </port>
       </service>
</definitions>
```

Figure 6.6: Sample of WSDL showing its elements, adapted from (Cerami 02)

We give an example to illustrate the above concepts and elements in a WSDL file. Figure 6.6 above shows an example of a WSDL file that provides a single publicly available function, called “greetHello”. The function inputs a single string parameter, and returns a single string greeting with the word “Hello ” concatenated with the input string. For example, if you pass the string parameter “Santokh”, the service returns the

greeting, "Hello Santokh" This is a Request-Response type of operations, i.e. the service receives a message and sends a response and these are encapsulated in the input and output elements respectively of the binding element as shown.

6.2 An Aspect-Oriented Web Services Description Language

As the Web Services Description Language (WSDL) (Christensen et al 01) is an XML grammar for describing web services, aspect-oriented elements were added to the WSDL documents to make it richer in content, effective and more useful so that it can better describe the advertised aspect-oriented web services. This transformed the WSDL into AOWSDL and resulted in better characterization and categorisation of the aspect-oriented services advertised in the document. The aspectual elements and their details will allow web services clients to dynamically locate web service providers because their descriptions are better and more thorough than in the plain WSDL documents. Also, by using the AOWSDL, clients can discover and consume clearly defined whole aspect-oriented components, each component exposing of a number of related operations necessary to complete whole transactions instead of just exposing isolated individual operations scattered thorough out the service documents as is portrayed in the current WSDL documents.

<definitions> The root element of a WSDL document

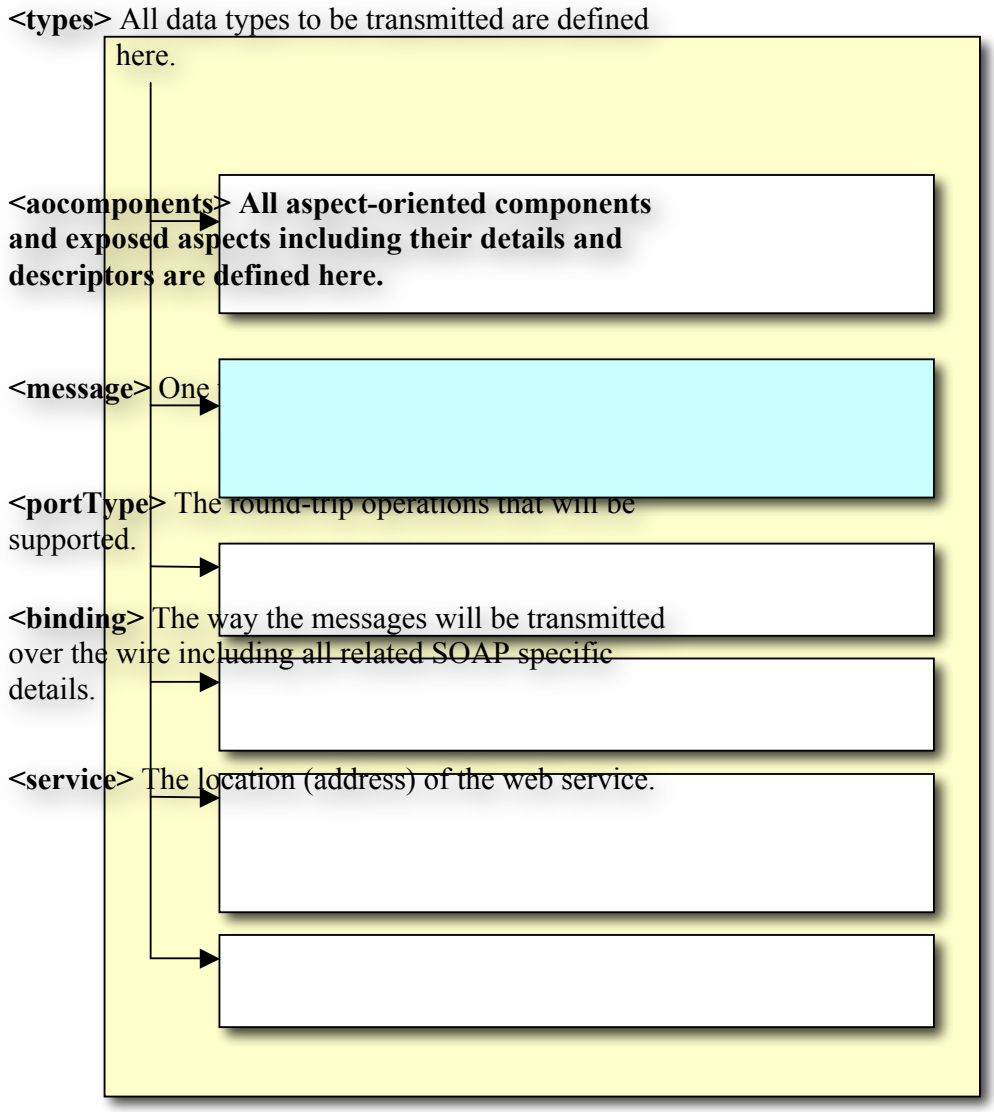


Figure 6.7: AOWSDL document showing the hierarchy of its elements.

A number of aspectual extensions, all neatly bundled into a major “aocomponents” element, were added to the standard WSDL document as shown in figure 6.7. The inclusion of these aspectual elements transformed the WSDL into Aspect-Oriented Wed Service Description Language, or AOWSDL for short. The purpose of this is to enable the description and capture of the rich and highly characterised aspectual features of web services in a systematic manner. Figure 6.7 further gives a

summarised description and overview of the whole AOWSDL document showing the hierarchy of the six major elements of WSDL together with the “aocomponents” major element for describing aspectual features. A standardized method for extending WSDL with aspects nested in components was used. AOWSDL also allows for more dynamic and automatic searches for any given aspect, aspect details and properties of the services advertised because our AOWSDL specifications for web service components follow consistent, formal and clearly defined semantics and syntax. The complete AOWSDL schema and a sample of the AOWSDL can be referred to in the appendix. We describe the main parts of the AOWSDL schema with reference to its application in an example of an AOWSDL document from our Travel Planner web services system.

The initial section of the AOWSDL schema and a sample of its implementation in the AOWSDL document is shown in Figure 6.8. All the aspect-oriented elements and descriptors are enclosed within a main “AOComponents” element. Complete documentation for human consumption about the web service’s aspects and components is placed within the “AODocumentation” element. This documentation also includes high level instructions to software developers about the web service and how to access and consume it. Another element, the “WSDescription”, that is shown circled in both the AOWSDL and its schema, gives crisp instructions that are machine understandable and is used for automatic discovery and integration of the web service. Web service requesters first access this “WSDescription” element and match its description with their requirements.

6. 8(a) 6.8 (b)

Figure 6.8 (a) the initial section of the AOWSDL schema; and (b) implemented part of this section in the AOWSDL from the travel planner.

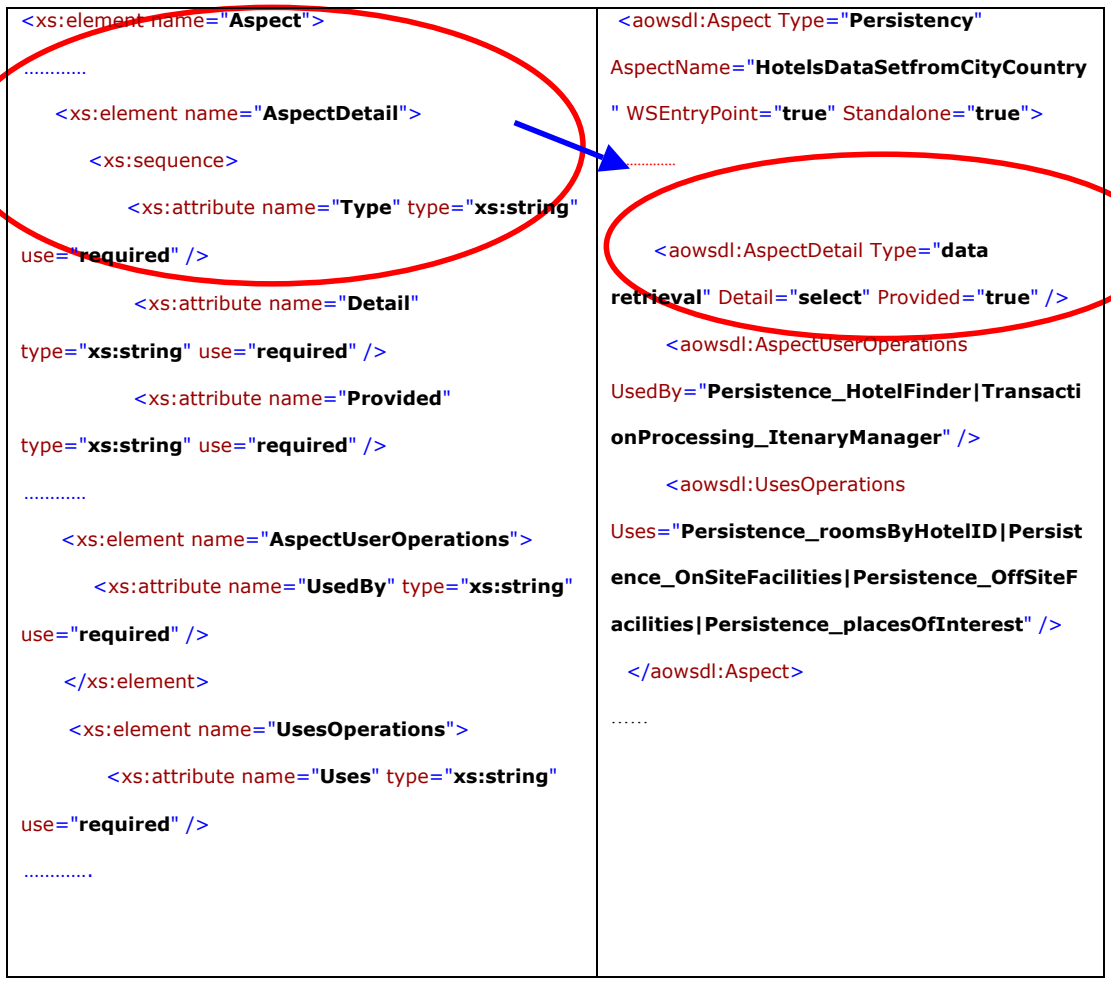
Each and every component of the web service provider is nested within the "AOComponents" element as shown in Figure 6.9 below. These components contain all the aspects that are exposed to the clients. The clients can make further XML queries to verify whether or not their detailed needs match those provided by the components and their aspects. These descriptions are also highlighted in Figure 6.8, where the clear and concise language used allows automatic querying to be possible.

Figure 6.9 (a) Components with aspects nested within them from the AOWSDL schema; and (b) corresponding elements in AOWSDL.

Each component exposes one or more aspects. Each aspect element contains details about all its cross-cutting features. It is specified as a functional or non-functional type of aspect. It also has an aspect type associated with it, e.g. the aspect type could be Persistency, Distribution, Transaction, Security etc. If the aspect can be used without resorting to the use of another aspect first, i.e. there is no precondition that another aspect need to be used before it can be consumed, then its "WSEntryPoint"

attribute is set to “true” in the AOWSDL. All the aspect descriptors shown are used to facilitate automation.

As shown in Figure 6.10 below, each aspect has one or more aspect details associated with it. If this aspect detail is provided, its “Provided” element is set to “true”, if it is required from others, it is set to “false”. This enables clients to understand the aspects in more detail and query whether it serves their needs or not. AOWSDL as such supports better description, characterisation and categorisation of web services than the plain WSDL without aspectual support.



6.10 (a) 6.10 (b)

Figure 6.10. (a) Aspect details in the AOWSDL schema; and (b) corresponding elements from the AOWSDL document in the collaborative travel planner.

The Aspect-Oriented Web Services Description Language (AOWSDL) as such represents a contract between an aspect-oriented service provider and a service requestor. AOWSDL is also platform and language independent and is used to describe aspect oriented web services. Using AOWSDL a web services client can more easily discover and dynamically locate a web service and invoke any of its

publicly available functions. AOWSDL also acts as a platform to help automatically integrate the services provided with the requesting client.

6.3 Locating Web Services

Web service providers need a registry to publish information about their location, the services they provide and other technical details including on how to consume them. Web service requestors on the other hand need a way of knowing what web services are available, where they are located and how to integrate with them. Web services discovery agencies are specially built to fulfil this role, which is akin to "discovering, matching and marrying" appropriate web service providers with requestors so that they coexist in harmony. The Universal Description, Discovery and Integration (UDDI) tool is one such registry (UDDI website 05). It is an industry wide initiative to standardize how web service providers can be discovered, described and integrated with web service requestors.

UDDI defines a SOAP-based API for querying centralized web service directories and makes it possible for developers to discover the technical details of a web service as well as other service-oriented information, classifications and contact details through its WSDL file. It is comprised of three parts: the White Pages, the Yellow Pages and the Green Pages. The White Pages describe the company offering the Service, providing the name, address and contact details. The Yellow Pages classify the company by standard groupings and the Green Pages describe the interface to the Web Service so that someone can write an application to use the Web Service (UDDI website 05).

6.4 Locating Web Services using the Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI) registry

The normal UDDI does not support the use of rich aspectual information in AOWSDL that is useful to better understand, dynamically discover and consume aspect-oriented web services. Publishing AOWSDL documents to UDDI is like "presenting encyclopaedias to illiterates", i.e. the normal WSDL parsers will just ignore the rich aspectual elements as if they were non-existent. As such an Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI) registry was required so that more accurate and useful discovery and integration can be achieved through queries pertaining to the aspects and their details.

6.4.1 Overview of AOUDDI

AOUDDI is a prototype service registry that is used to describe, discover and integrate aspect-oriented web services. Aspect-oriented web service providers use the AOUDDI to publish information about their aspect-oriented components, aspects and aspectual details besides publishing the AOWSDL information about their location, the services they provide and other technical details including on how to consume them. Aspect-oriented web service requestors make aspectual queries the AOUDDI to search for the web services that they require, determine their location and ultimately

integrate with them. They achieve this by resorting to the use of the rich aspectual information available in the AOWSDL documents of the web service providers.

An example of this type AOUDDI in use is shown in Figure 6.11 below. The various aspect-oriented web service providers shown, i.e. the Flights web service, Hotels web service, Car Rentals web service and Payment web service, all publish information on the AOUDDI about their exposed aspectual APIs, the services they provide, their location and other technical details including on how to consume them. The information in the AOWSDL is extracted and stored in a web services repository.

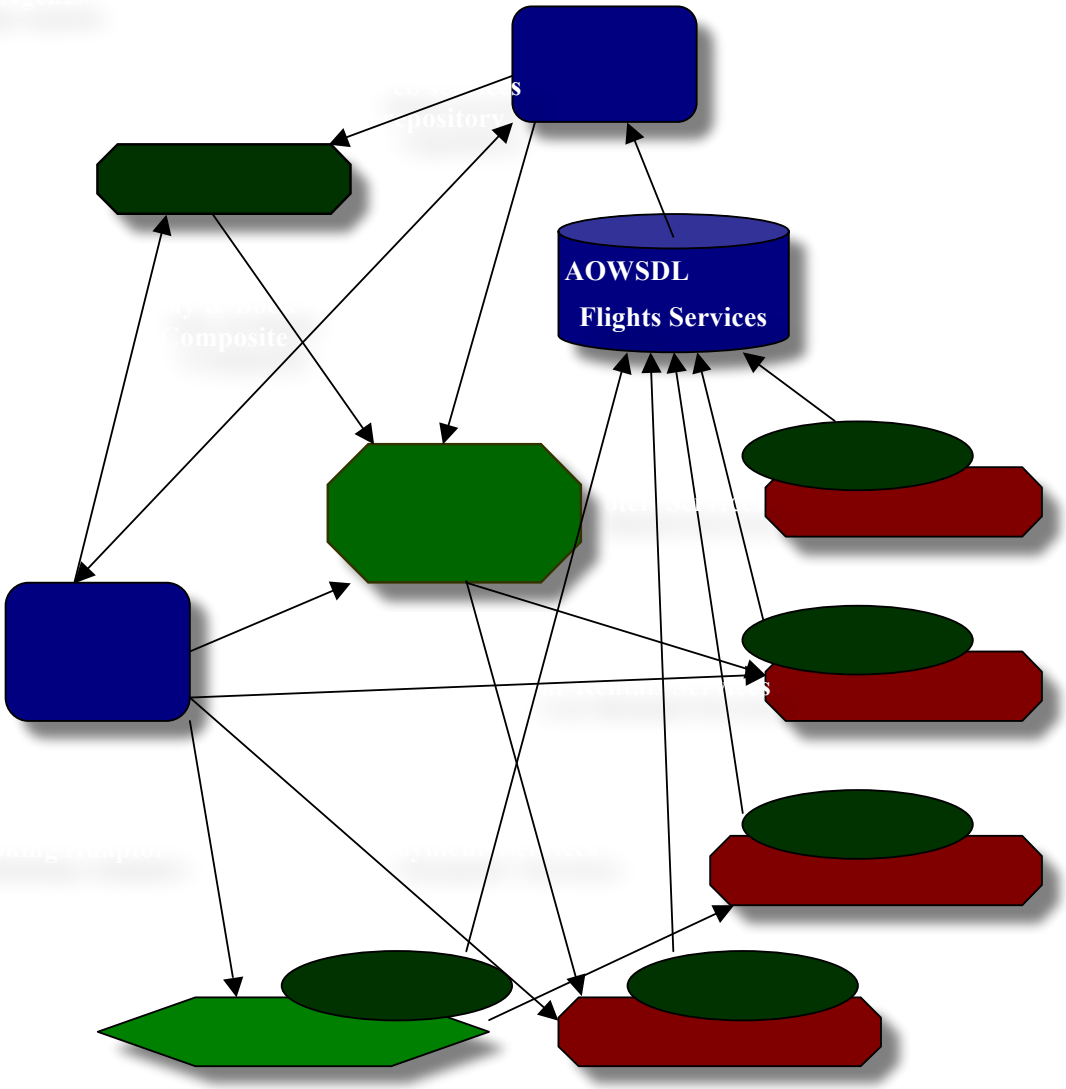


Figure 6.11: AOUDDI's use in the aspect-oriented web services system.

The web service requestor here is the Travel Planner Client as shown on the left in Figure 6.11. It also shows the AOUDDI making use of aspectual information in Testing Agents to further verify and validate the AOUDDI. Aspect information is also used by the AOUDDI to locate adaptors that can be used, for instance, to consume web services using different protocols or create composite mechanisms to consume

multiple web services. The AOUDDI's requirements engineering, analysis and design is explained in the following subsections. A subsection providing more detailed explanations on how to use the AOUDDI is also given.

6.4.2 AOUDDI Requirements Engineering

An Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI) registry prototype was designed and implemented. Figure 6.12 below shows the use case diagram of the AOUDDI.

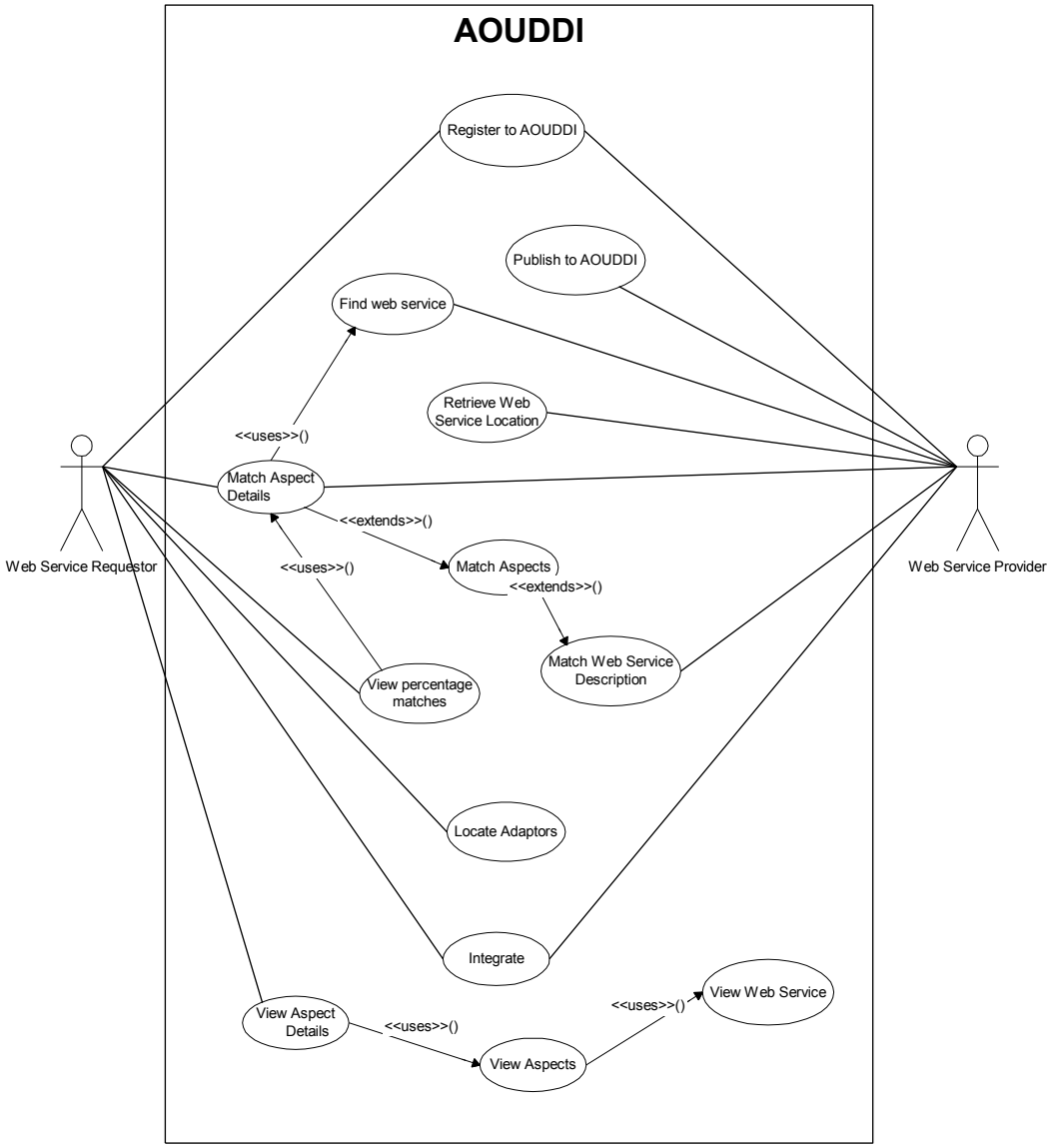


Figure 6.12: Use case diagram of the AOUDDI

As shown through it specifications in the use cases, web service providers are allowed to publish all information regarding the services they provide onto the AOUDDI. This information includes all aspectual information together with their details. Web service requestors are able to query the AOUDDI to search for web services that they require. They are can also make additional and more detailed queries about the aspects. The AOUDDI is able to match the requests in the queries with the details provided by the web service providers in their AOWSDL file. If the web service requestors find a web service that satisfies their requirements they can integrate with the web service providers and subsequently consume it.

6.4.3 AOUDDI Analysis and Design

The aspect-oriented analysis of interfaces in the AOUDDI is shown below in Figure 6.13. The types of aspects involved are abbreviated and clearly shown within double angled brackets preceding the function name. A key to all the abbreviations used for the aspects is included in the diagram. A positive or negative sign preceding the aspect type within the brackets indicate whether the aspect is provided or required. For instance <<+Prs>> means that it is a persistency type of aspect and it is provided by the component. Any sign outside the angled brackets was automatically inserted by the Visio modelling tool. A positive sign is inserted before all functions by the modelling tool to indicate that the visibility of the function is public.

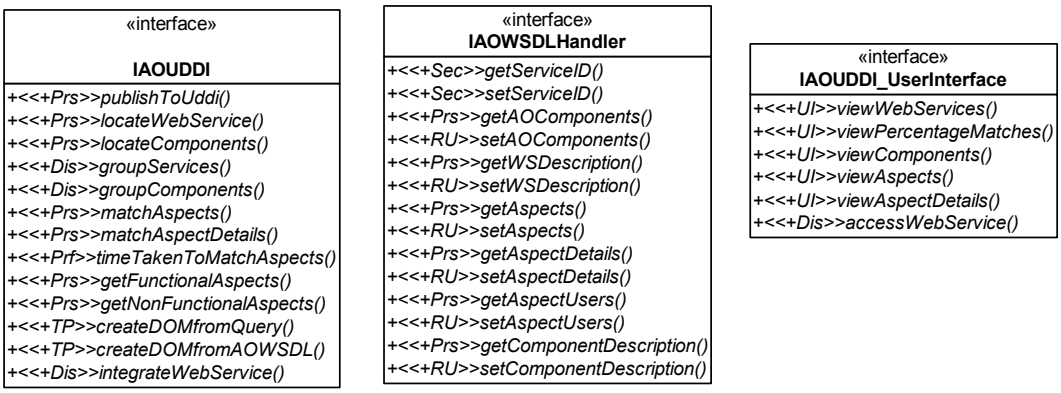


Figure 6.13: AO-UDDI Aspect-Oriented Analysis of Interfaces

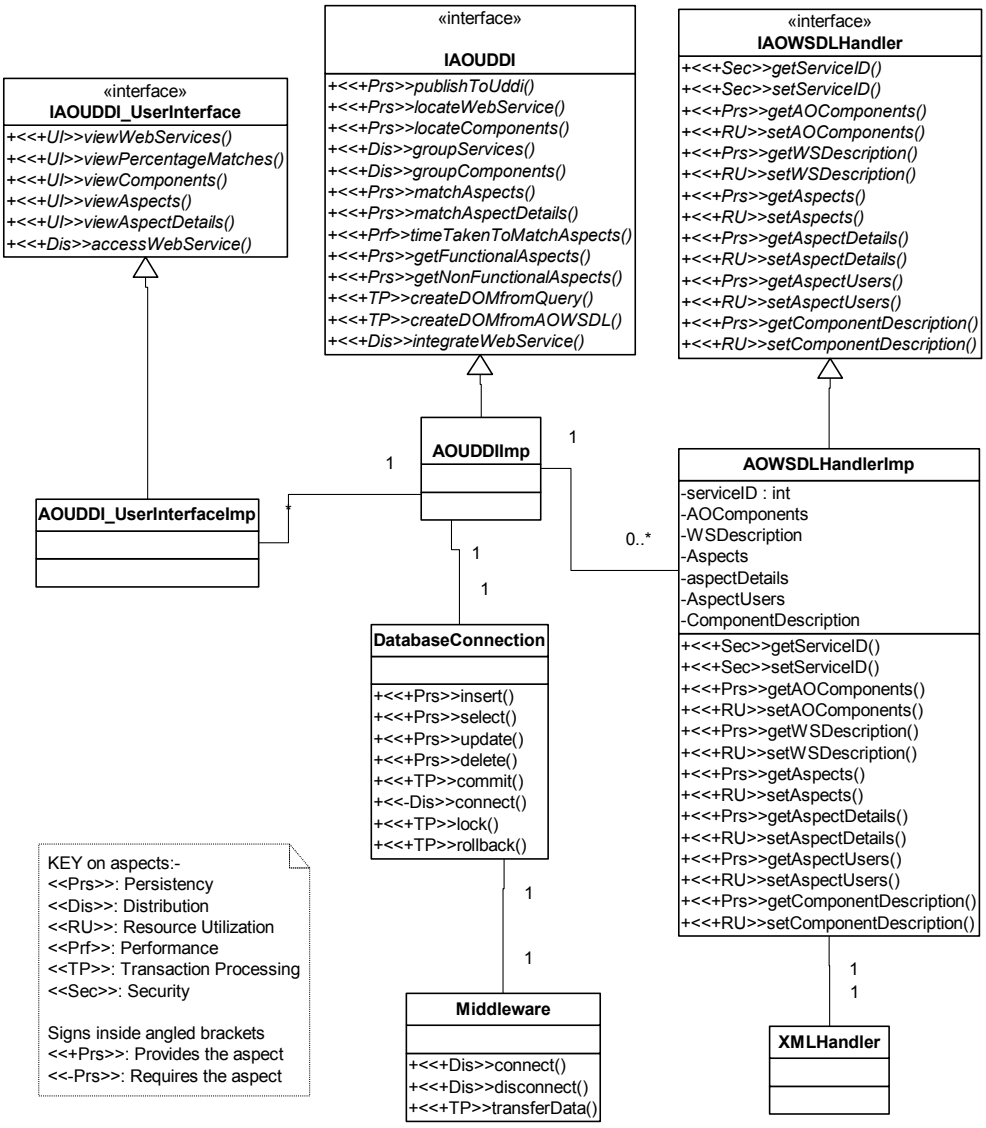


Figure 6.14: Aspect-Oriented Design of AOUDDI

The Aspect-Oriented Design of AOUDDI registry is shown in Figure 6.14. The IAOUDDI interface is implemented by the AOUDDIImp class. This object allows aspect-oriented web service providers to publish their services and related technical information onto the registry. The IAOWSDLHandler which is implemented by the AOWSDLHandlerImp is like a helper class to the AOUDDI in matters relating to the contents of the AOWSDL document. It subsequently allows web service requestors to discover and locate these web services by making queries to the AOUDDI. An

IAOUDDI _UserInterface was also implemented so that users can manually query the AOUDDI to view the web services provided including important information relating to aspects and components.

The AOUDDI also allows web service requestors to locate aspect-oriented components by making XML queries to it based on the description of required components and their aspects. The location and information of the web service provider containing the required components is returned to the requestor. This information is particularly useful when locating adaptors to perform specific tasks. The AOUDDI further enables two or more web services to be grouped together into a composite web service so that a particular set of tasks can be performed. For instance the Booking component of the hotels web service and a Payment web service provider can be combined so that bookings to a room in a hotel and a mode of payment for it can also be arranged.

The AOUDDI provides for more effective and efficient discovery and description of aspect-oriented web services by further allowing queries to be based not only on descriptions of web services and their components but also on aspects and aspect details. These are specific modular units within the components themselves, each exposing useful services that can be consumed by clients requiring them. The description of aspects and their details together with the web services and their components descriptions enable dynamic discovery of web service to be done more efficiently and with more precision. Once the required web service is discovered and

located, the AOUDDI allows the web service requestors to integrate with the web service providers and subsequently consume it.

6.4.4 Using the AOUDDI

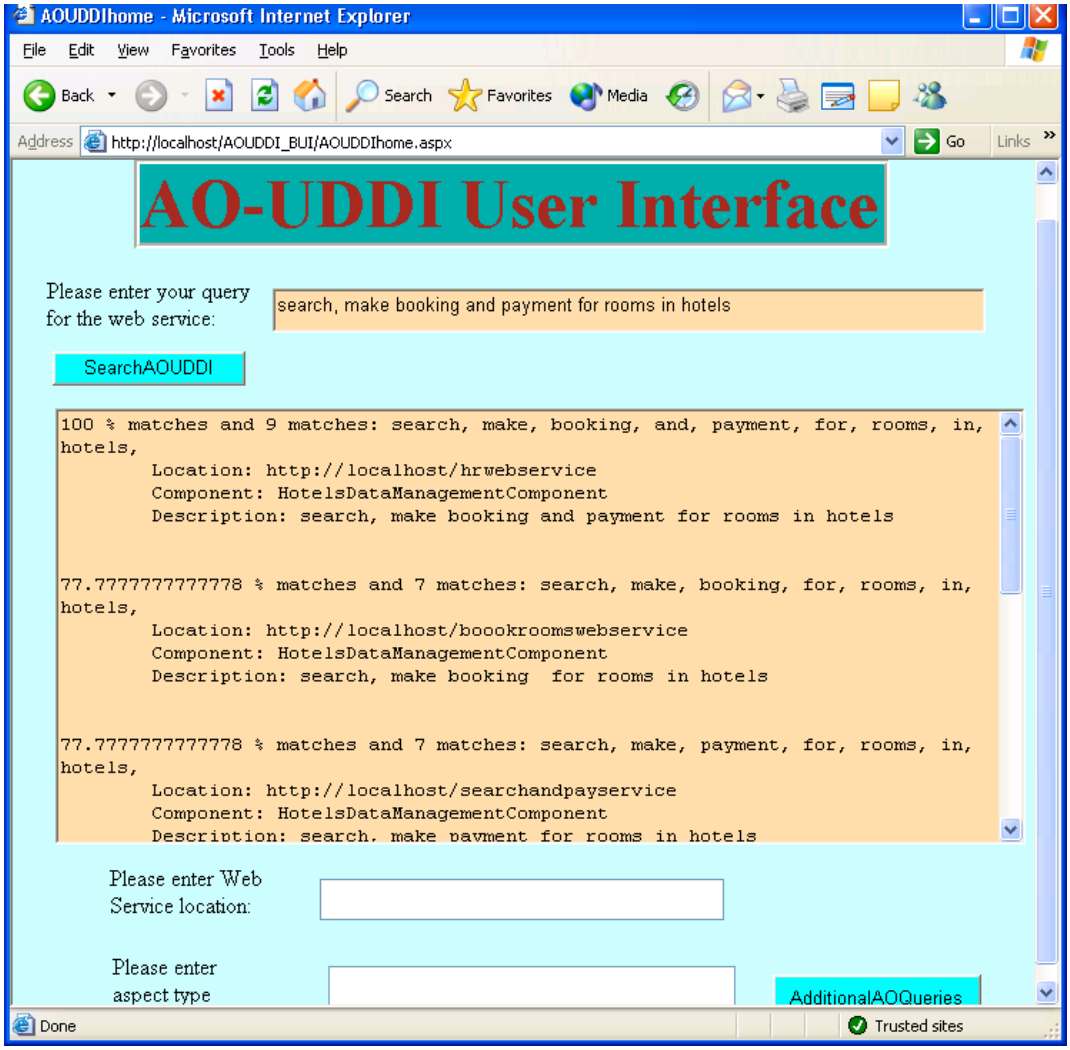


Figure 6.15: AOUDDI User Interface for manual use.

The AOUDDI User Interface for performing the tasks of discovering appropriate aspect-oriented web services manually and integrating with them is shown in Figure 6.15 above. Users first enter the type of service that they require into the first Text Box shown which is labelled “Please enter your query for the web service” and click the Search AOUDDI button. The AOUDDI will return a list of all the web service providers that match the initial request together with a percentage showing the degree of match between the request and the services advertised by the web service providers. For example, in figure 6.15 above the request was for a web service that could provide “search, make booking and payment for rooms in hotels”. The AOUDDI user interface will then query the AOUDDI which will return a list of web service providers, including their locations and descriptions. It also returns the percentage and number of matches between the query and the web service description as shown in the figure.

More specific queries based on aspects can be subsequently made through the AOUDDI user interface. The AOUDDI responses to these queries by matching the request containing the aspects with the aspects in its required components, again giving a percentage match. The AOUDDI user interface allows the user to make further detailed queries based on aspect details as shown in figure 6.16 below. As shown here, the “Component Name” and “Aspect Name” entries are optional as these may only be known to regular AOCE users of that particular web service. The response to this includes the number of matches and details about the components and the aspects.

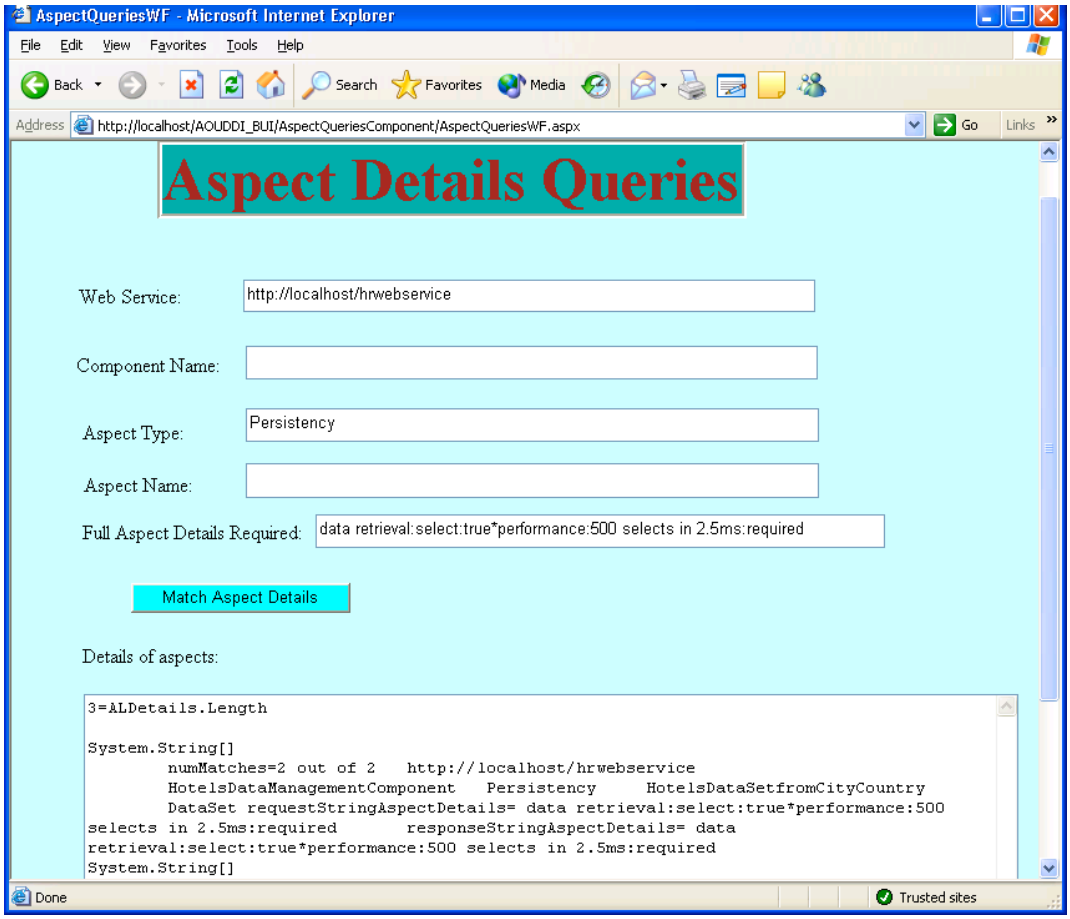


Figure 6.16: AOUDDI User Interface for doing more detailed checks on aspects

If the user is satisfied with the searches, he can stop at any stage and make a decision whether to integrate with and consume the web service or not. All the above queries can also be done automatically by web service requesters querying the AOUDDI to search for web service providers. The searches follow exactly the same steps as above but not through using the AOUDDI user interface. They query the AOUDDI directly.

By using the AOUDDI registry, aspect characterisations can be used both to inform about category choices and also act as a post-filter to refine and rank located candidate services. Aspect characterisations support composition of web service components by retrieving multiple components that are used to satisfy the query.

They also support adaptor location and integration when required by a web service client. We have also used them to perform run-time validation of discovered web service components using testing agents which synthesise test requests to the web services to check their actual run-time behaviour.

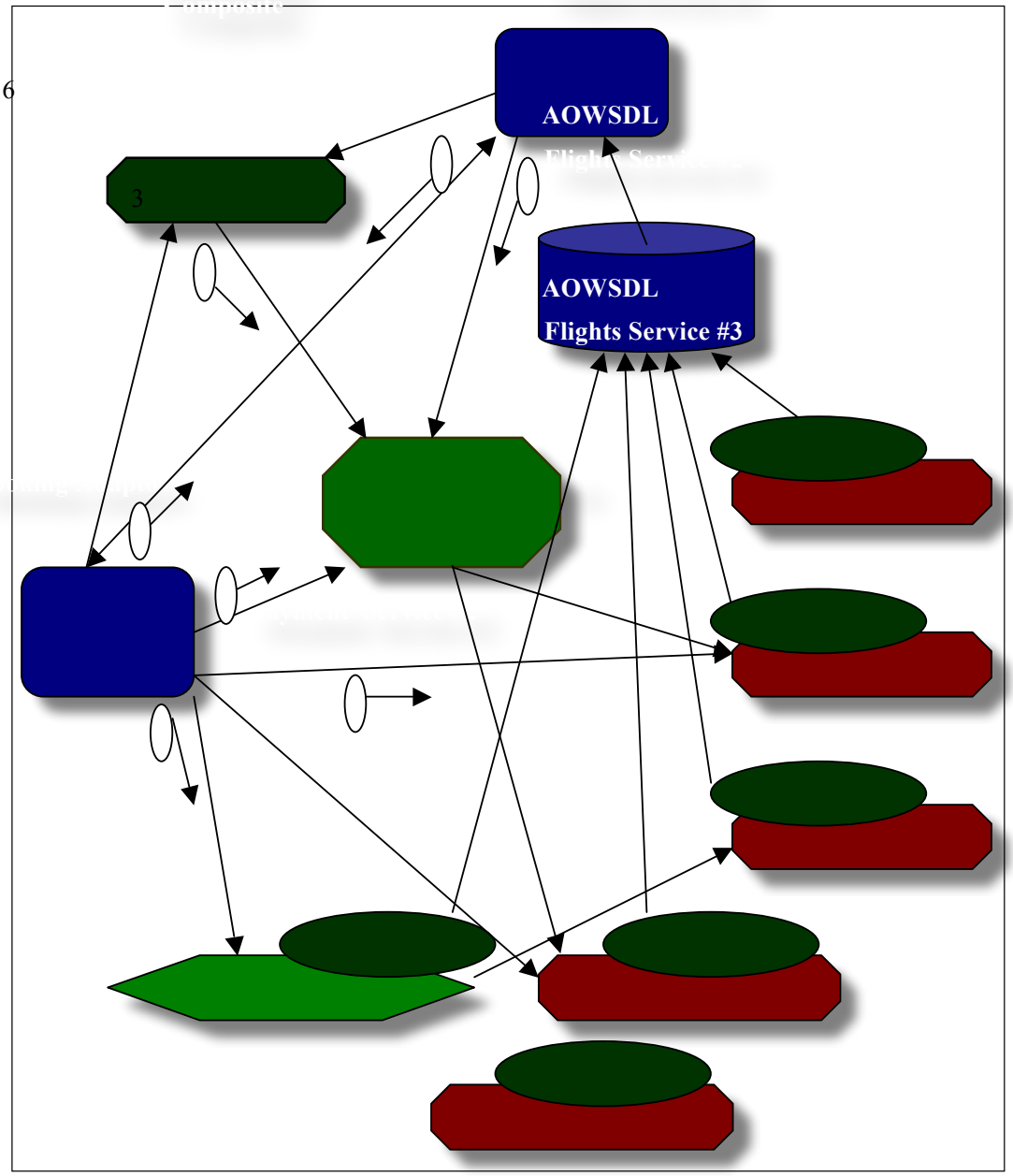


Figure 6.17: An example of an aspect-enhanced UDDI query mechanism for the travel planner.

Figure 6.17 illustrates these features with an example from our .NET travel planner application. The travel planner client issues a query to an AOUDDI registry, asking for a flight booking service (1). This query may specify any flight booking service be returned, or may constrain the desired services by, for example, saying the service must use an optimistic booking protocol, must return a book request query reply in less than 5 seconds, must use a specified security protocol, and so on. The AOUDDI registry locates registered web services matching the category query for a flight booking service, filters and ranks these by further aspect-based constraints, and returns the list to the client application (2). The user selects a service or the client chooses a "best match", and the service is invoked as required (3). Sometimes a desired service provides an incompatible SOAP protocol to that required by the client. The AOUDDI registry can search for an appropriate adaptor mapping one protocol to the other. Alternately, a component can make use of adaptors "standardised" for different aspects e.g. a generic data storage, event broadcasting, transaction co-ordination, itinerary item booking etc service (4).

A more complex query to the AOUDDI registry by the client might ask for a combined flight booking and payment service. Such as combined service doesn't exist, but the AOUDDI registry provides a composite service of an appropriate flight booking service and payment service (5). Accessing the composite functionality is via a simple synthesised web service that sequences interaction with the existing services (6). Testing and Verification Agents (7) were used to verify the accuracy of the responses to the queries.

6.5 Summary

The Web Services Description Language (WSDL) is used by web service providers as the XML format to describe their services including all the exposed methods, method signatures, namespaces and the handling URL. Web service providers utilise web services discovery agencies, like the Universal Description, Discovery and Integration (UDDI) registry to publish details about their location, services and related technical details. The UDDI registry is a business and service registry that enables businesses utilising web services technology to describe, discover and integrate with each other.

The current WSDL does not contain XML grammar that is necessary to describe the rich, highly characterised, categorised, aspectised and componentised web services that were built using AOCE techniques. A more complete and comprehensive description language that includes references to the components, aspectual elements and their details is required to describe web services built using AOCE techniques.

As such an extended and aspect-enriched form of WSDL called Aspect-Oriented Web Services Description Language (AOWSDL) was formulated. By using the AOWSDL, clients can also discover and consume clearly defined whole aspect-oriented components, each component exposing of a number of related operations necessary to complete whole transactions instead of just exposing isolated individual operations scattered thorough out the service documents as is portrayed in the current WSDL documents. An Aspect-Oriented UDDI (AOUDDI) was also developed specially to describe, locate and integrate clients with the aspect-oriented web services. The description of aspects and their details together with the web services and their

components description enable dynamic discovery of web service to be done more efficiently and with more precision. Once the required web service is discovered and located, the AOUDDI also allows the web service requestors to integrate with the web service providers and subsequently consume it.

Aspect characterisations can be used both to inform about category choices and also act as a post-filter to refine and rank located candidate services by using the AOUDDI registry. Aspect characterisations support composition of web service components by retrieving multiple components that are used to satisfy the query. They also support adaptor location and integration when required by a web service client. Aspect characterisations are also used to perform run-time validation of discovered web service components using testing agents which synthesise test requests to the web services to check their actual run-time behaviour. The AOUDDI used in conjunction with the AOWSDL as such provides a superior system for the description, discovery and integration of web services.

7 The AOConnector in AOWS

In this chapter we explain a novel, convenient yet vital interlinking object called the Aspect-Oriented Connector, or AOConnector. It is used to inter-connect and allow for the efficient and effective flow of information and instructions between the various AOWS subsystems that we have discussed in the previous chapters. The AOConnector is reusable, extensible and can be replicated and customised for use with different clients. Its purpose is to make the various AOWS subsystems more lightweight, understandable and maintainable. An alternative implementation to the aoconnector approach will be described in the next chapter, i.e. one that incorporates the extensive use of multi-agents based exclusively on AI techniques and agents co-operating and negotiating with each other to dynamically execute tasks that enable autonomous AOWS description, discovery, integration and subsequent consumption of the services.

7.1 AOConnector

Our earlier AOWS prototype, (Singh et al 04), did not incorporate the AOConnector subsystem into that version of the aspect-oriented web services system. This made its subsystems, especially the AORequester, AOUDDI, AOComposite, AOTestingAgents and AOAdaptors particularly bulky due to being composed of numerous components and heavy on code. We decided to extract the dynamic support, processing and linking functionalities into another intermediate subsystem so that clients need not know about nor integrate directly with the other subsystems, and as such can concentrate on core business logic and functionalities, thus making the clients more lightweight, understandable, maintainable and reusable. To do this we

introduced the AOConnector which relies on dynamic reconfiguration and Inversion of Control mechanisms (Fowler 04) into a later AOWS model that was refactored and redesigned from the earlier version to address these issues.

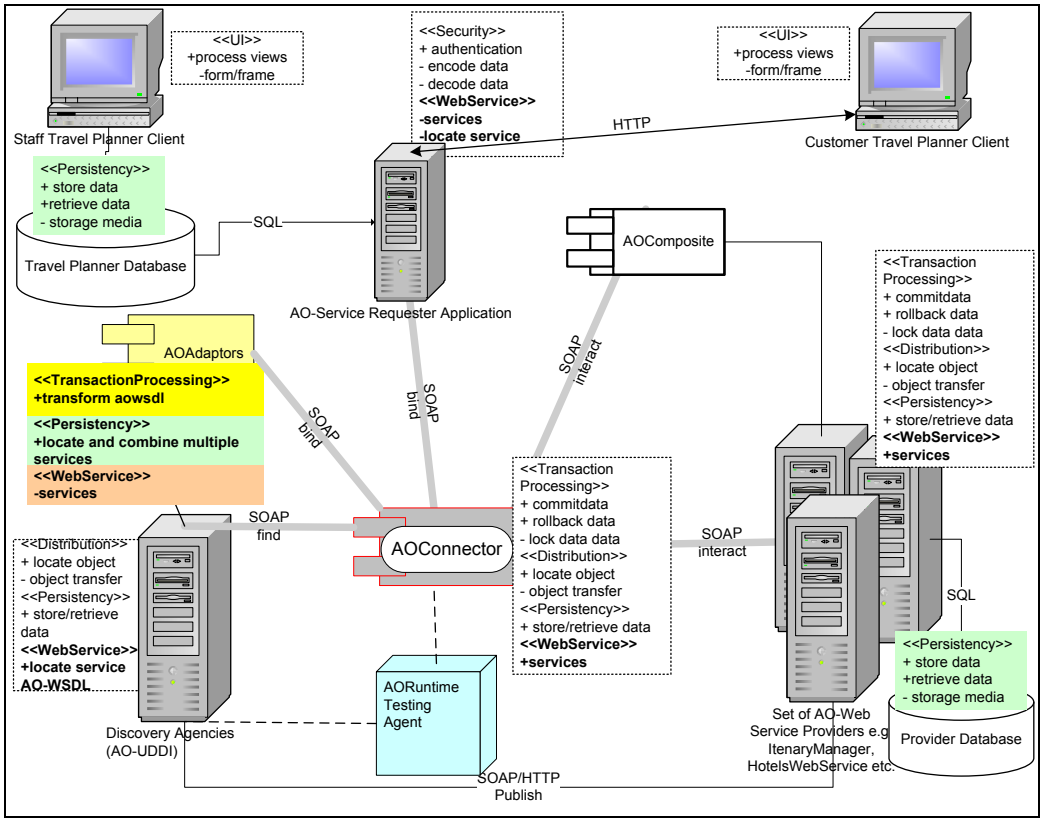


Figure 7.1 The travel planner's AOWS-based architecture with the AOConnector.

To give a clearer understanding of the significance of the AOConnector in our AOWS based systems, we describe its role in our prototype travel planner. The travel planner's AOWS-based architecture and the main aspects of components in its subsystems (the aspects are in the format <<Aspect name>>) are shown in figure 7.1 above. The aspect-details for each aspect are listed below its name. These details have a "+" (detail provided) or "-" (detail required) symbol preceding them. Components making up each subsystem/application expose interfaces that relay information about these aspect-oriented functions. Interfaces are implemented within the component

containing them and are used by other aspect-oriented components assembling them together to build the application in accordance with the AOCE methodology.

7.1.1 AOConnector Requirements Engineering

The AOConnector is required to provide an independent mechanism to facilitate component communication between a multitude of subsystems, i.e. the requester (client), AOUDDI, multiple service providers, testing agents, AOAdaptors and AOComposites. It also uses the mechanism of Inversion of Control so that clients can use it to look-up and integrate with discovered web service providers without the clients having to know the identity of the providers. Clients just need to pass their requests to the AOConnector object and this object will carry out all business processing including integrating and consuming web-services to achieve this. As such its requirements specifications have numerous use-cases. Figure 7.2 below shows the use cases for the AOConnector object in relation to the various subsystems interacting with it.

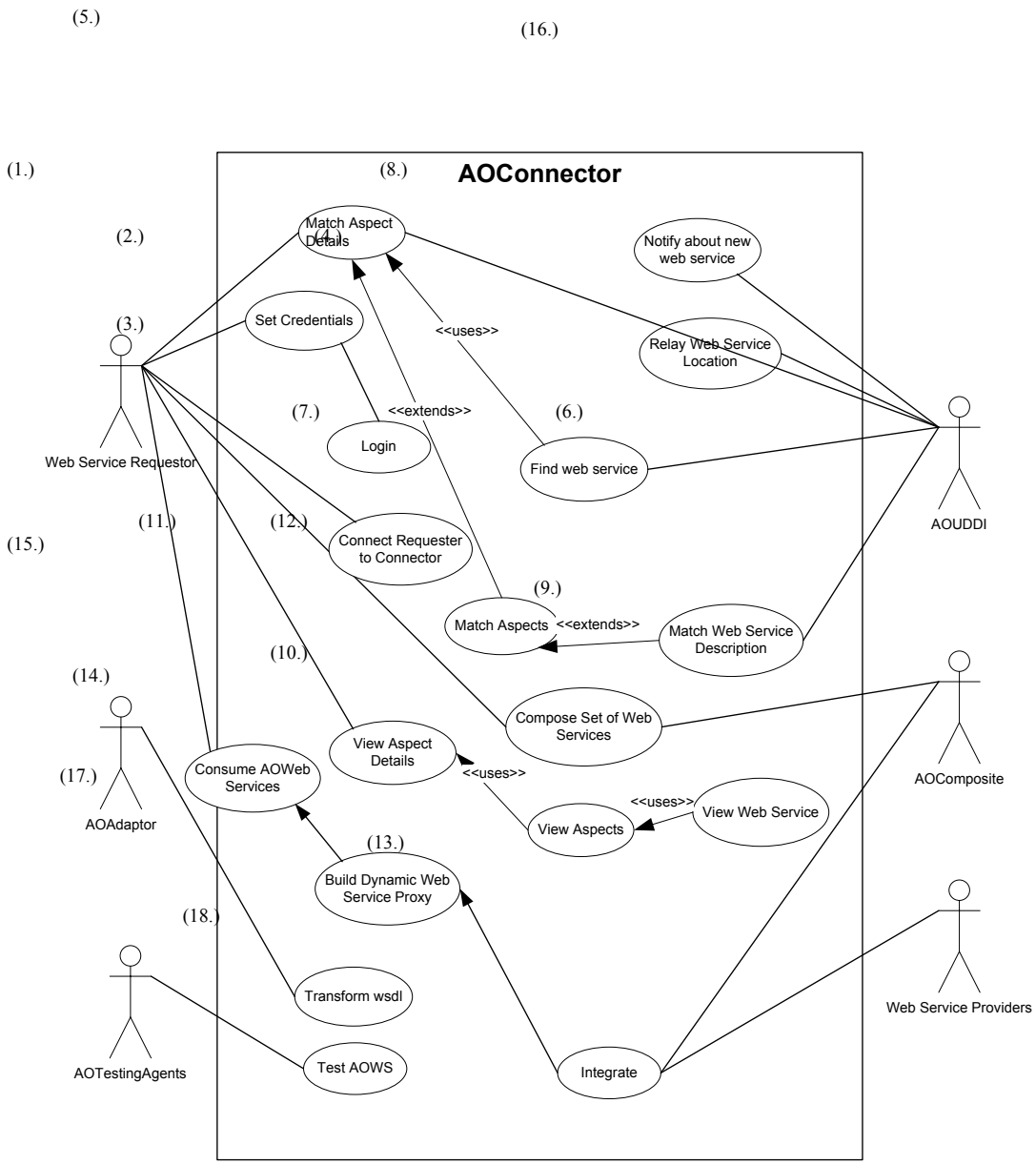


Figure 7.2 Use case diagram of the AOConnector object

In the figure shown above, the use case to set credentials (1) is used to give access to only the authorised service requester to allow it to link to and use the connector object to access and interact with the other subsystems in the AOWS. This also ensures that only one requester uses the AOConnector and permission is denied if the same connector is tried to be used by other requesters simultaneously. The login function (2) together with the setting credentials function further allows authorised access to manually use the AOConnector if we wish to make changes, e.g. if we wish to manually link up to selected service providers without resorting to the AOWS inherent dynamic integration mechanism.

The AOConnector must first be connected (3) to the requester, before it can be used to query the AOUDDI to search (4) for the appropriate service provider that is requested by the client. Multiple queries are allowed to be made to determine that the best matching service provider is selected by making additional aspect enriched queries (5) in XML format using the SOAP over HTTP protocol to the AOUDDI. These queries include those to match the service description (6), aspects (7), aspect details and their respective properties (5) as shown in the use cases above. We can then get the web service description document and the location (8) of the most optimum provider i.e. its AOWSDL document, and parse it through AOWSDL parsers to extract the information contained in it including its location and services that it exposes. The AOConnector then needs to dynamically build an AOWSDL proxy, shown as the "build dynamic web service proxy" (15) use case, which can then be used to call remote procedures over the wire so that it can be executed by the corresponding provider and the results obtained transmitted back to the requester via the AOConnector.

Requesters should be able to view the web service (9), including the aspects (10) and aspect details (11) related to the service. Requesters can also use the connector to construct an AOComposite (12) which is a composite of a variety of web service providers. The client/requerter can then integrate (13) with either the individual service providers or the composite object via the connector. Dynamic web service proxies should also be allowed to be built (14) to support autonomous mechanisms. The services can then be consumed (15) by the clients using the connector object.

The AOUDDI can also notify (16) the requester through the AOConnector about any new web service provider that has just published itself with the AOUDDI. Adaptors

can be used to transform (17) the Service Description Documents that are of a different protocol to the ones used in AOWS. AOTesting Agents can be used to test and validate (18) any aspect of the AOWS system to check whether it conforms to the specification or Quality of Service (QoS) as promised by the subsystem or service.

7.1.2 AOConnector Architecture and Design Diagrams

The AOConnector uses a novel specialisation of the pattern of Inversion of Control (Fowler 04) that we modelled, designed and developed ourselves. Our technique can be used to develop highly decoupled, mobile, lightweight and unit-testable software. The main idea behind IOC is that an object exposes its dependencies via some form of contract, for instance the interface of the AOConnector object in our AOWS serves as its contract. Dependencies can be anything that an object needs to perform its designated function but is not concerned with its implementation, i.e., it may include the object's interface, system resources etc (Mathew 05). In a nested object graph, each object in the call chain exposes its dependencies to the outer caller that uses it, which in turn exposes those dependencies including any of its own to its caller and so on, until all dependencies manifest itself at the top. The top-object then assembles the dependency graph before activating the objects. The top-object is generally an entry point into the software system, e.g. in our Travel Planner client, its the main method that serves as the entry point. In AOWS, a nested situation as described above arises when we use an AOComposite, which exposes its dependencies to the AOConnector to communicate with it, and the AOConnector in turn exposes its dependencies to communicate with the client. Figure 7.3 below shows the inter-relationships between the client (requester) and the other subsystems (shown enclosed within a red circle) of AOWS via the AOConnector. In this system, the AOConnector object exposes its public functionalities via its interface and these are its dependencies.

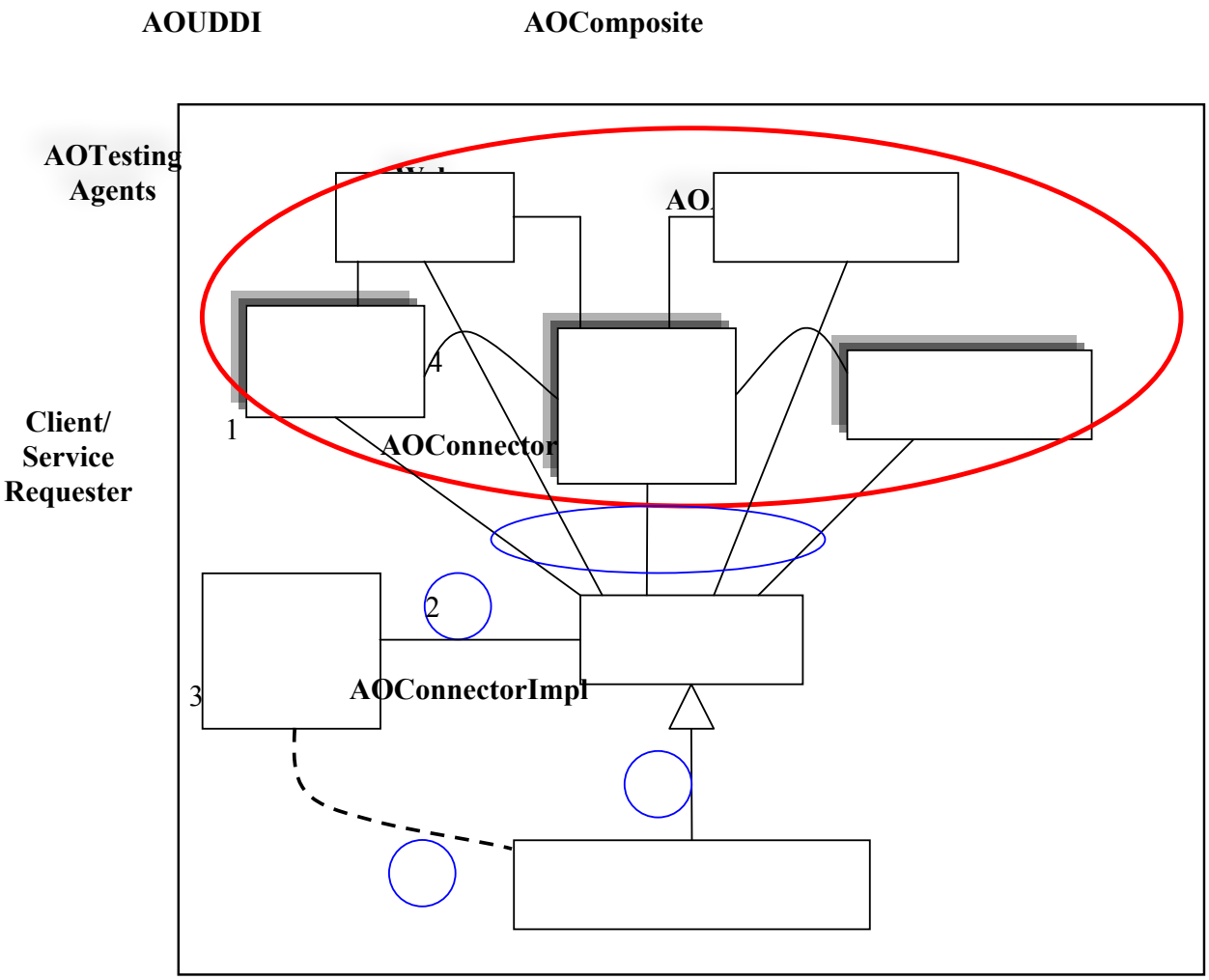


Figure 7.3: The AOConnector uses the Inversion of Control mechanisms

The client (service requester), as shown through item numbered (1) in figure 7.3, links to the AOConnector and utilises it to make remote service calls to the appropriate service providers. Through this pattern, the client does not concern itself with which service is called by the connector to execute its tasks. The client uses a method of the AOConnector called “TransactionProcessing_ExecuteMethod” and let the connector decide which service the connector has integrated with to use to serve the clients request best. The connector is implemented by the AOConnectorImpl concrete class that implements this method of its interface. The full signature of this method is given in Figure 7.4 below:

```
public object TransactionProcessing_ExecuteMethod(string MethodName,
object[] Param)
```

Figure 7.4: The TransactionProcessing_ExecuteMethod method's signature

It takes two parameters, the name of the method of the remote web service to be invoked which is a string object called "MethodName" and an array object of the parameters to be used as input parameters by the service. The AOConnector's interface is implemented in the AOConnectorImpl (2.) which is its derived class (sub-class) and it can be accessed by the client through its constructor etc. as shown by item numbered (3) in the figure. The connector object then looks up (4.) the appropriate service provider that in turn executes the method (by dependency injection of the input parameters into the service). The provider returns an appropriate response to the connector that further processes it if necessary and then relays it to the requesting client.

```
AOConnector myAOConnector = new AOConnectorImpl();
String[] myParams = {Stockholm, Sweden};
Dataset myDataset =
myAOConnector.TransactionProcessing_ExecuteMethod(searchForHotels,
myParams);
```

Figure 7.5: Example of using the TransactionProcessing_ExecuteMethod

Figure 7.5 above shows an example of a client calling the TransactionProcessing_ExecuteMethod() method on the AOConnector. The client uses it to look for a dataset of all the hotels that can be found in Stockholm, Sweden. The AOConnector will search for this method from the list of services it has

integrated with and if found, request the particular service to execute the "*searchForHotels*" and relay the results to the client.

If the particular method does not exists within the currently consumed web services, the connector will query the AOUDDI for a service that exposes such a method and, integrate with the service provider, consume the service to execute the method and return the results. If such a method does not exists even within all the services registered with the AOUDDI, then the AOConnector will transmit a message back to the requester that such a method does not exists and will suggest similar methods, for instance similar methods with less or more parameters, if such methods are available. As such the use of Inversion of Control in the AOConnector object allows for separation of business logic calls to be handled by the connector thus making the clients more light-weight, modular and easy to maintain and refactor. Clients do not need to overload themselves with the server side logic and remoting connections and transactions, all these are handled by the connector object.

The aspect-oriented analysis of the component interfaces in the AOConnector sub-system is shown below in Figure 7.6 below. The types of aspects involved are abbreviated and clearly shown within double angled brackets preceding the function name. A key to all the abbreviations used for the aspects is included in the diagram. A positive or negative sign preceding the aspect type within the brackets indicate whether the aspect is provided or required. For instance <<+Prs>> means that it is a persistency type of aspect and it is provided by the component. Any sign outside the angled brackets was automatically inserted by the Visio modelling tool. A positive sign is inserted before all functions by the modelling tool to indicate that the visibility

of the function is public. Since all these functions are to be consumed and used by other components/subsystems interacting with the AOConnector, all the signs preceding them are positive indicating that they are all public operations and further, they are consumable even remotely over the web. This gives the advantage of having any of the subsystems, including the connector, residing in any machine and allows for remote access to any of the subsystems by any of the other subsystems.

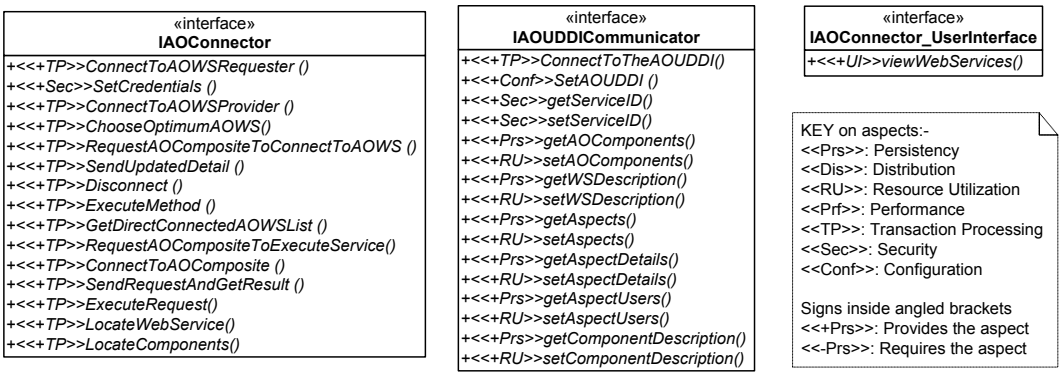


Figure 7.6: Binding Interfaces of the AOConnector subsystem that are exposed to the other subsystems in AOWS

Figure 7.7 below further shows the various components and supporting classes that make up the AOConnector subsystem. The connector is highly componentised based on clearly defined division of functionalities so that no overlapping or duplication of work is done in any of the interacting objects. The 3 interfaces that are exposed for use by the other subsystems are the same as described above in Figure 7.3. All the other interfaces of the components are to be used within the AOConnector only and are hidden and not exposed outside the connector. These non-externally exposed components, shown in the figure below, are the AOWSDLRefStorageFacility Component, ArtificialIntelligence Component, Connection Component, DynamicDiscovery Component, NotificationHandler Component, RequestHandler

Component, XMLDecoder Component and DynamicWSProxy Component. Their use and functions are explained in more detail below.

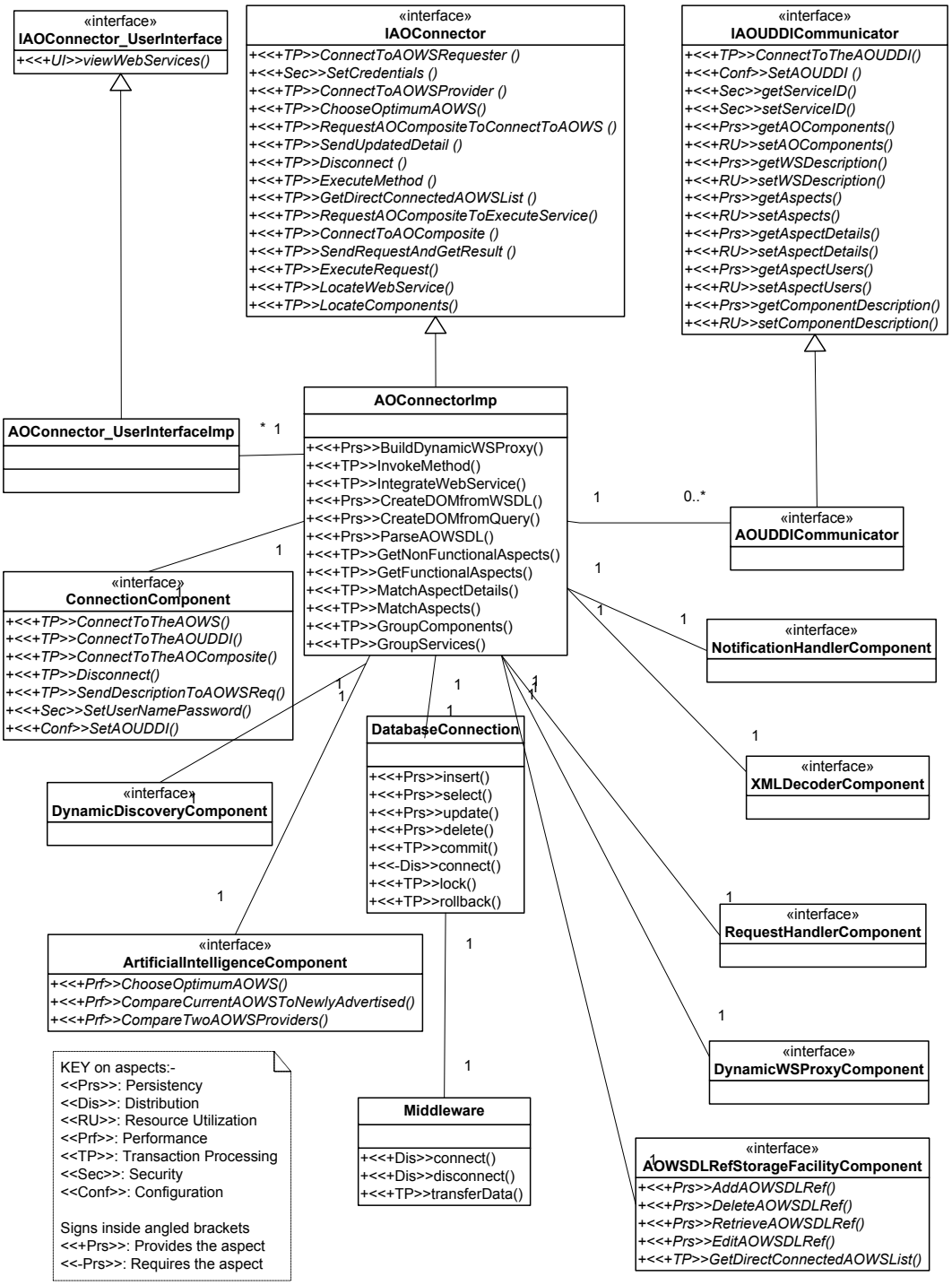


Figure 7.7: The various components and supporting classes within the AOConnector subsystem

The AOWSDLRefStorageFacility component stores the AOWSDL web references locally for more efficient and quicker access. The ArtificialIntelligence component contains a number of artificial algorithms that are used to carry out searching tasks through the parsed AOWSDL document, including considering the aspects, their details and properties. The Connection component is used to make connections between the connector object and the other sub-systems within the AOWS. The DynamicDiscovery component allows for dynamic discovery of web service providers to be achieved and is used together with the AOUDDI to verify that the services discovered are what the requester requires. All new notifications about newly registered services are dealt by the NotificationHandler component. The RequestHandler component handles all the requests made by the requester, and either processes it or passes it on to the relevant subsystem to process the request. All XML decoding, whether from the requests, responses or AOWSDL, is done through the XMLDecoder component. The DynamicWSProxy component builds the AOWSDL proxy dynamically based on the AOWSDL of the provider so that integration and subsequent consumption of the provider can be achieved.

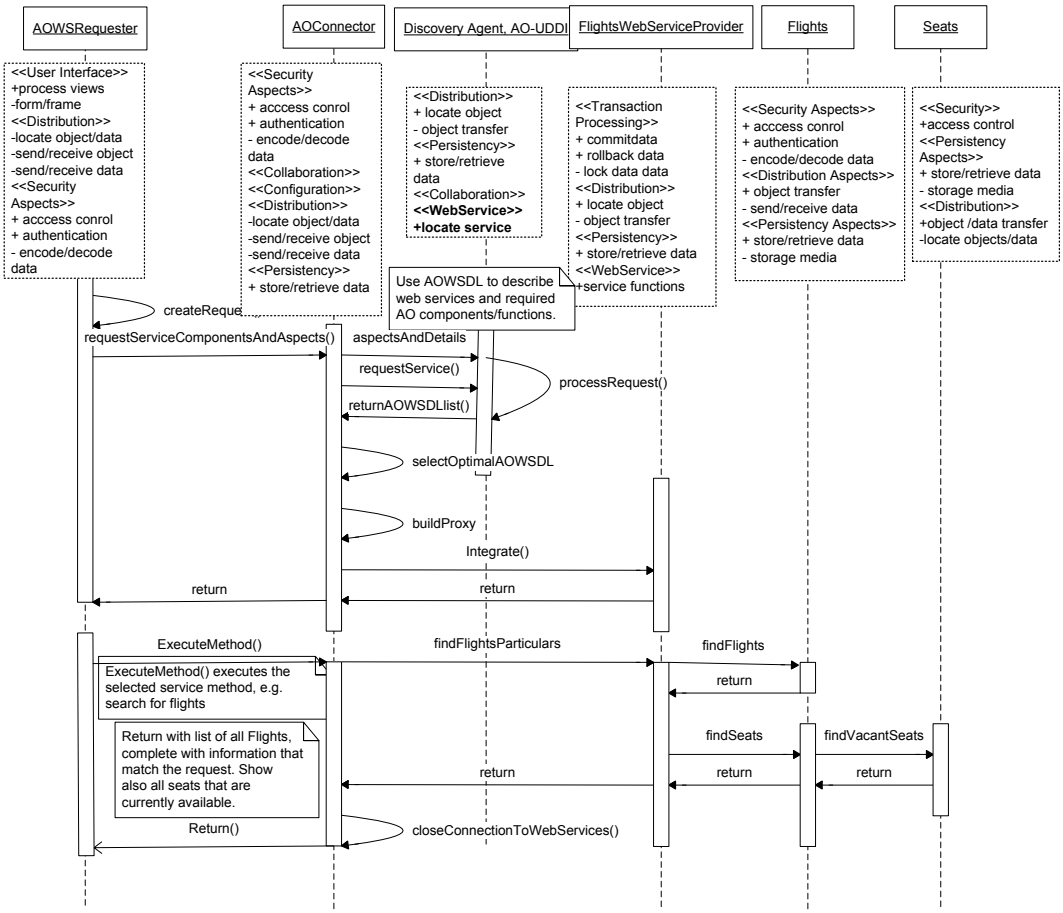


Figure 7.8: Sequence diagram for dynamic discovery, integration and consumption using AOConnector

Figure 7.8 above shows the sequence diagram for the dynamic discovery, integration and subsequent consumption of a web service provider located by using the AOConnector. The web service provider we are searching for in this example is a flights web service provider that can be used to find the availability of vacant seats to particular destinations of choice. First a request object containing the full details of the required services, aspects and aspect details/properties is constructed by the requester. This is a String based query object that is transmitted to the connector subsystem as a compact and composite request in an XML format. The connector

parses and splits up this compact request and passes it on to the AOUDDI as a series of queries made up of separate requests for service description(s), aspects and its details/properties. The AOUDDI process these queries and return the AOConnector a list of service providers available that best match the requests. These services are ranked according to percentage matches based on the request and services available. The connector then has a choice to dynamically select the best service based in either the service descriptions, aspect matches or a combination of both.

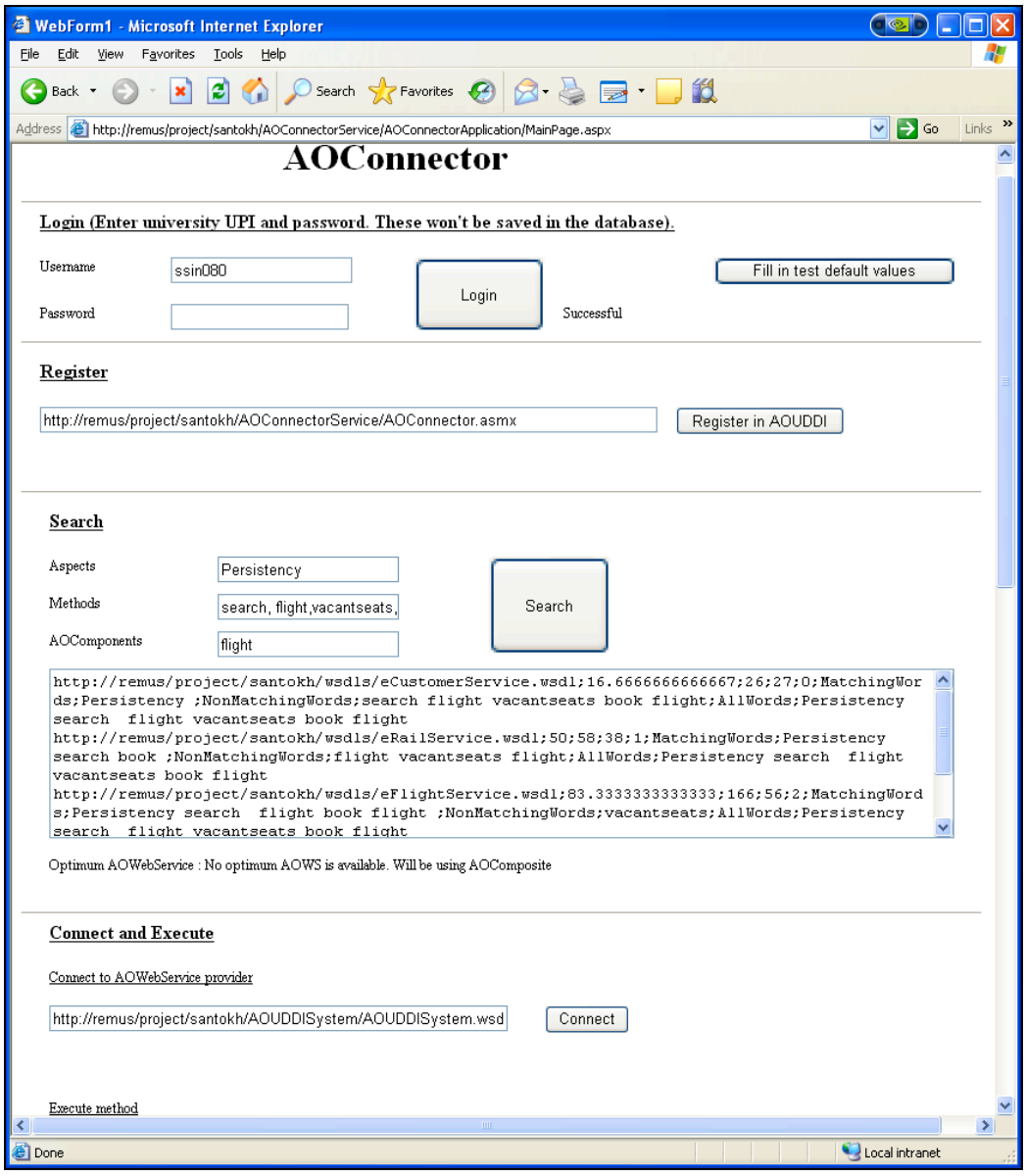


Figure 7.9: Screen-shot of the user interface of the AOConnector object

Selection of these services can also be done manually by users using the AOConnector user interface to view the search results and choosing the most suitable service from the list. Figure 7.9 above shows a screen-shot of the user interface of the AOConnector that allows us to carry out all the dynamic tasks step by step with some manual control as to choice of web services to be integrated with. If a 100% match is not found, then the service provider is inadequate and the user is suggested through the user interface to utilise an AOComposite object (which returns a composite of web service providers to completely satisfy all the user's requirements through the use of a cluster of multiple providers to provide the full range of services) instead of doing a direct connection with the highest ranking provider. But if the user still wants to use provider with the highest match, he is still able do so by simply choosing to integrate with it and subsequently consume it.

Figure 7.10 below shows the remainder of the user interface that allows us to consume the services in the provider by resorting to the execute method shown. If a single web service provider cannot provide all the functions requested for by the client then the interface allows us to construct an AOComposite and consume it by executing methods through it. We can check for notification from the AOUDDI (the URL of the AOUDDI queried is specified in the textbox preceding the Notify button) of any new provider by clicking on the Notify button. The particular AOUDDI's notification will appear in the textbox below the location of the button. Once we have finished using the service or want to replace the service, we can terminate the service by disconnecting from it.

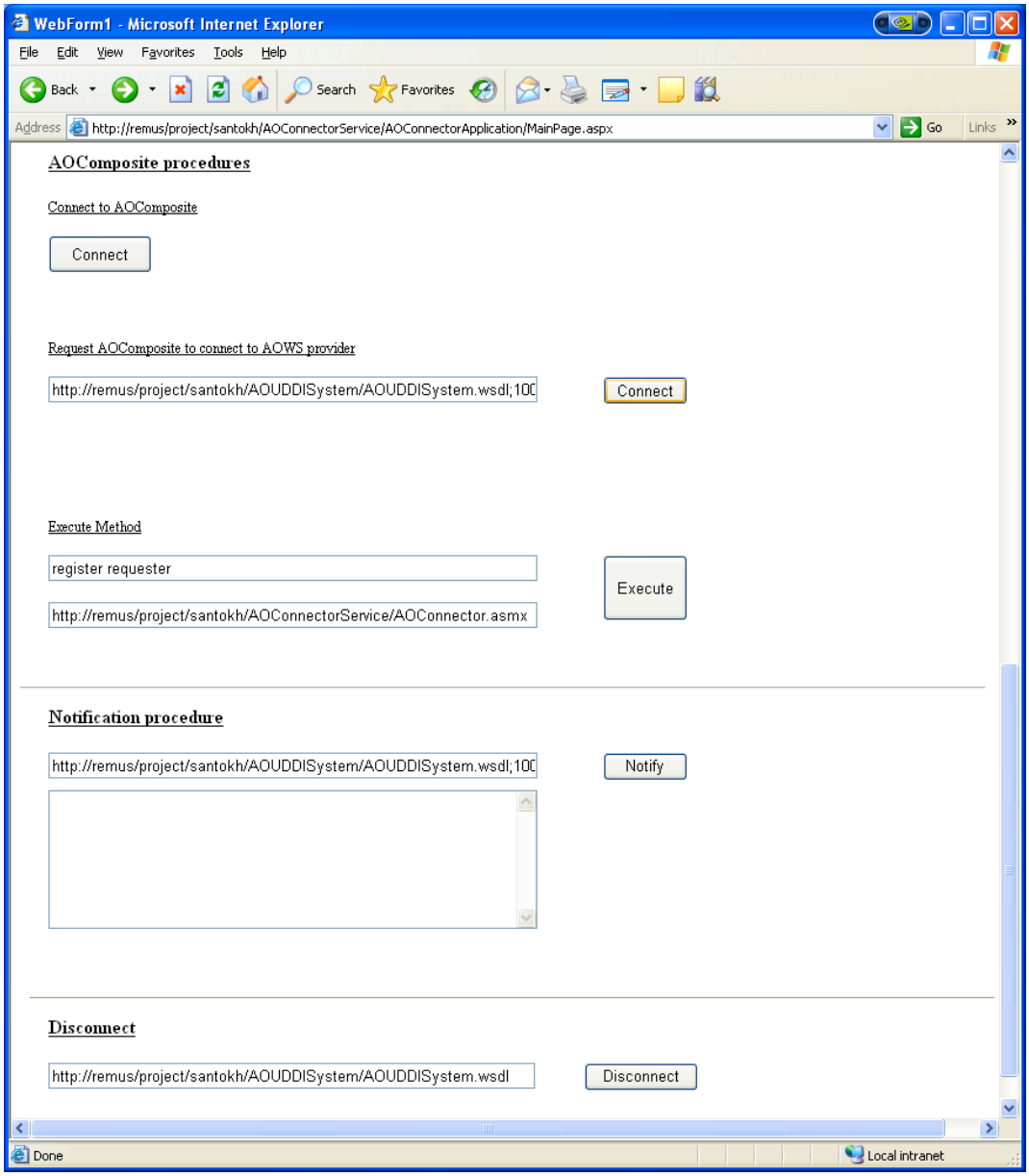


Figure 7.10: Screen-shot of the remainder of the user interface of the AOConnector.

Once the service provider is selected, the AOConnector dynamically builds and compiles a web service proxy and saves it in the binaries folder and keeps a local copy of it in memory. The client can then consume the web service by calling the TransactionProcessing_ExecuteMethod() operation of the connector and passing in the method name and parameters of the function it needs to consume. As shown in the second series of sequences originating from the requester in the sequence diagram in Figure 7.8 above, the requester makes a request to find flights and seats that are

available by inserting the values of the parameters e.g. departure and destination dates, times and venues, seat type, number of seats required, etc. The AOConnector will relay this information to the Flight web service which will then process the request and provide the appropriate response containing the results of its searches. The AOConnector will relay this information to the requester and close the web service connection.

The AOConnector is also used to keep track that if a service needs to be replaced with a better provider, then the to be discarded web service is first checked to see that it has finished executing all the requests and has returned its results before it is disconnected. It does this by keeping track of all requests made to each and every service and whether or not a response has been received. If no response has been received, the AOConnector knows that the web service is still processing the results and will wait for the web service to finish. Once this is done, the connector will disconnect the "old" web service and destroy its proxy thus breaking all contact and web references to it. The connector will then dynamically build a new proxy based on the information contained in the AOWSDL document of the new provider. Any call for subsequent requests will then be directed to and processed by this new web service provider.

7.1.3 Implementation of AOWS using the AOConnector

We then implemented the .NET aspect-oriented web services system based on the AOWS model incorporating AOConnectors that is composed of components that:

- are self-describing not only in terms of their interfaces but include aspect characterizations for richer run-time understanding and configuration;
- at run-time are able to dynamically locate web service components providing required services specified by their aspect characterizations queried through AOConnectors;
- may make use of “standardised” aspect-based adaptors to interact with discovered web services in a de-coupled manner without hard-coding type or behavioural information about the component.

Our novel method made the client more lightweight and easier to implement through the use of the AOConnector object. Our client need not have any knowledge of any of the web services that are being consumed to carry out its operations as all this information is stored and handled by the connector. The client can also call an operation of the AOConnector named “TransactionProcessing_ExecuteRequest()”, with the web services method’s description in a String format and its input fields in the form a String array object as the operation’s parameters. As an example, the method’s description from the client could even be "flight get schedule for cities" instead of the method’s actual name, the String Array object may have “Auckland” and “Wellington” as the departure and destination parameters with additional date and time constraints. In this example a request will be made to the connector to invoke the flights service (from the keyword “flight” in the input String parameter) that it has linked to and return the list of flight schedules based on the input parameters supplied.

The connector will first select the appropriate web service provider based on the existing information already supplied to it by the AOUDDI and dynamically use its proxy to invoke the service. The service will then execute the method and provide an appropriate response to the connector which will then relay it to the client. In this way the client is made very light weight and simple as they need not be aware of the details of the web services that are integrated and consumed by the AOConnector. In fact we mainly need to concentrate on the user interface aspects of the client, as all information about the web services is stored in the AOConnector during its initial configuration when it was linked with the client. If any service becomes unavailable, the connector dynamically requests the AOUDDI and gets a new updated list and selects a service from this new list. Programmers can also manually select from this list by using the connector's user interface (Figure 7.10) that was described earlier.

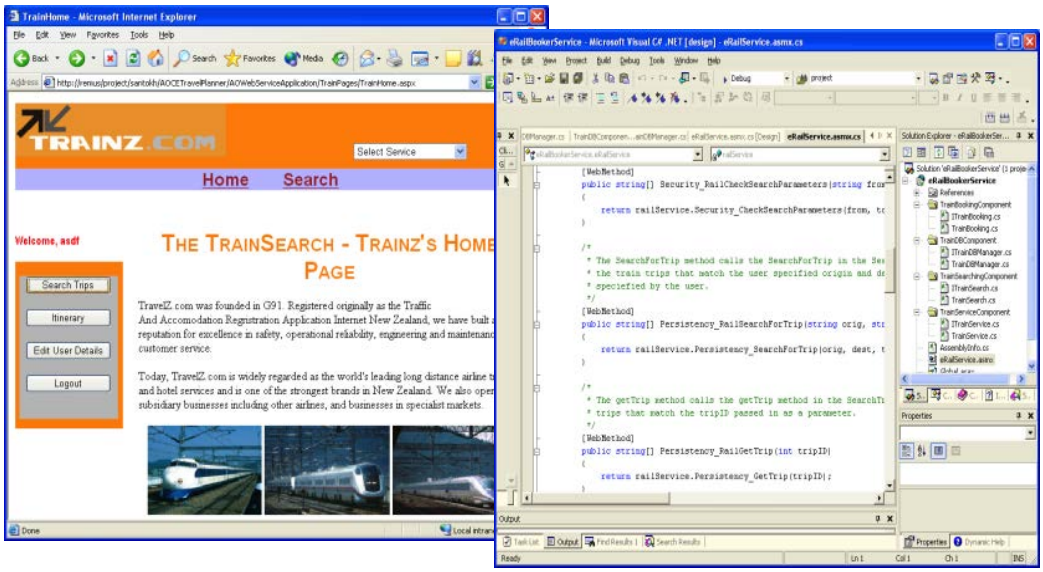


Figure 7.11 (a) The Travel Planner GUI and (b) example C# code snippet implementing aspects.

Figure 7.8 shows a section of the GUI of this Travel Planner application built using AOWS techniques. It shows the web form of an application used to search and subsequently book and make payments for trains to particular destinations. On the right is a sample C# code snippet. It depicts implementation of aspects identified in the program. The aspects were captured in the systemic components and are identified and described to make the components better characterized and categorized. The Solution Explorer to the right of the program listing portrays the aspect-oriented components of the software. These can be expanded in the Visual Studio IDE to show their interfaces and implementing classes within the components.

7.2 Discussions about the AOConnector

The AOConnector is of essence in our architecture as it allows for the separation and retention in the client of the core client functionalities of the requester from other ancillary and collaborative operations like making dynamic linkages and connections, relaying requests to the correct AOWeb services, obtaining responses, processing information and making decisions e.g. choosing the best web service provider. The connector also acts as a buffer-cum-conduit for the flow of information between the requester and the outside world. Clients can thus be engineered as lightweight systems as they need less code and fewer components and can concentrate on their main functional activities. This makes them easier to design and implement. Also, being lightweight, clients are more understandable and easier to refactor, thus making the system as a whole more maintainable and scalable.

The downside of using an AOConnector is that it constitutes another subsystem to be designed and implemented. Furthermore clients have to rely heavily on the AOConnectors for communication meaning more transactions are required to complete each request/response operation. These are easily outweighed by the numerous advantages described above. Furthermore developers just need to link a client to the AOConnector object. Also the AOConnector can be reused or replicated for use with other clients and AOWS subsystems thus making any future AOWS development easier and more streamlined.

Based on the AOWS architecture, we developed our travel planner prototype system using Visual C# web service components and .NET. The web service clients were

implemented without hard-coding any remote service information but instead use our extended AOConnector, AOUDDI mechanism and AOWSDL documents to locate components satisfying required services. Web services were implemented so that they are dynamically located by the AOConnector and integrated with clients. Web service components can be run-time tested by dynamic validation agents to ensure that they meet their aspect characterized performance and other constraints in actual deployment. Several adaptors were implemented to allow a web service client to interact with discovered web services without direct knowledge of their SOAP protocols and behaviour, instead using standardised, aspect-categorized adaptor messages for indirect interaction.

Our novel AOWS architecture enabled us to achieve a higher level of characterization and modularization in our travel planner system compared to other conventional approaches (Vitharana et al 03, Colyer et al 04). The use of the AOCE methodology to build our web services-based travel planner system resulted in increased understanding of the interrelationships between the various subsystems and components concerned. By identifying and capturing cross-cutting concerns using AOCE for the travel planner services we found that the development process was considerably simplified. The aspect-enhanced designs and implementations were found to be more easily understood, making this AOWS-based system more maintainable and scalable. Others working on software development related with aspects and components (Katara et al 03 and Colyer et al 04) have found similar results.

7.3 Summary

The AOConnector was found to be a very useful and novel subsystem because it allowed clients to be constructed as smaller and more manageable units and allowed for the separation and retention in the client of the core client business functionalities of the requester while the connector dealt with all the discovery, integration and consumption issues of the web services. The connector used an Inversion of Control mechanism to control and process the flow of information between the requester and the outside world. This caused the clients to be more lightweight as they needed less code and fewer components and can concentrate on their main functional activities. These made it easier to design and implement the clients and the clients are also more understandable and easier to refactor, thus making the AOWS system as a whole more maintainable and scalable.

8 Multi-Agents in AOWS

An alternative implementation to the AOConnector approach is described in this chapter. This is an architecture that incorporates extensive use of multi-agents (Sycara 98) based on AI techniques and intelligent agents co-operating and negotiating with each other to dynamically execute tasks that enable autonomous AOWS description, discovery, integration and subsequent consumption of the services. We call this novel software architecture Intelligent Aspect-Oriented Web Services or IAOWS to differentiate it from the one described in the previous chapter that does not use intelligent agents to negotiate and coordinate tasks within and between the various subsystems involved. We first explain what multi-agents are and give an overview of their use in IAOWS. We then describe our IAOWS architecture and an initial implementation using .NET web services technology to engineer and deploy the Multi-Agents and capture the rich cross-cutting aspects together with their behavior and interaction within our highly distributed system. We also give examples depicting the actual use and deployment of multi-agents in our Travel Planner prototype based on IAOWS.

8.1 Multi-Agents

The objective of using Multi-Agents is that they allow for more autonomous description, discovery, integration and interactions based on the comprehensive use of AI techniques. The agents themselves are composed of highly modular aspect-oriented components comprising independent units with each agent having clearly defined functions/tasks to individually or collectively negotiate and execute within the context of the other co-operating intelligent agents.

8.1.1 IAOWS Overview

IAOWS uses the concept of multi-agents and aspects, in this case aspects that impact on different parts of not only the web services, but also the agents. The characteristics of Multi-Agents (Sycara 98) within a system are that (i.) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (ii.) there is no system global control; (iii.) data are decentralized; and (iv.) computation is asynchronous. Figure 8.1 below shows an example from our earlier prototype travel planner system that we refactored and restructured based on the IAOWS concept of incorporating multi-agents into our previous system. As shown, by using discovery agents the client/requester looks-up various services from a registry (1). We deployed Discovery Agents to coordinate with the client's Requesting Agents to search the repository of the AOUDDI, and return results best matching the web service descriptions requested for, including descriptions for their components, aspects, aspect details and provided/required aspectual features

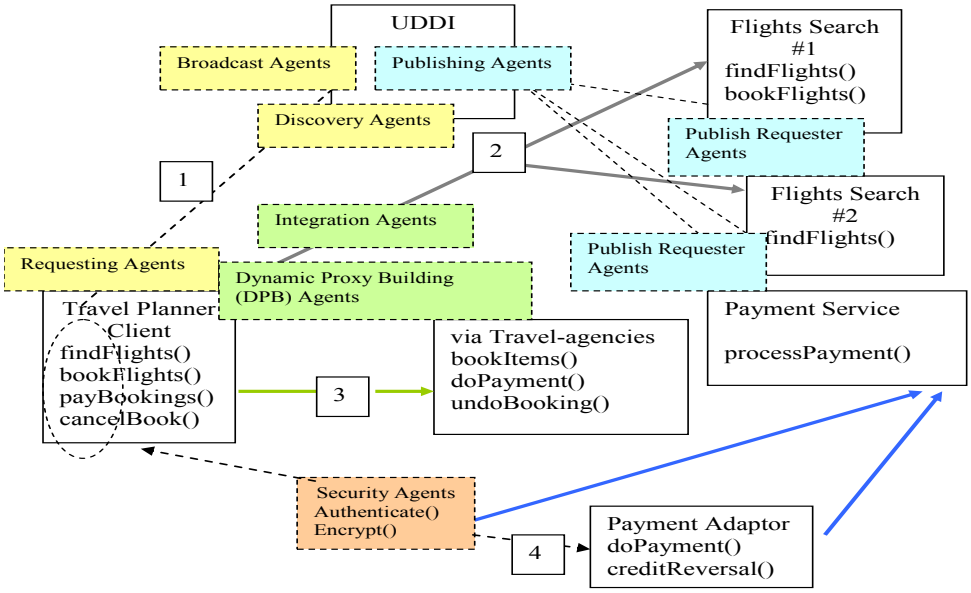


Figure 8.1: Example of web-service based travel planner utilizing multi-agents.

As shown in this example, flight searches for clients are performed by dynamically integrating with various discovered flight service providers (2) using the integration and proxy building agents. Bookings can be made directly or through agents (3), and if required payment subsequently made through a web-service based allowable mode of payment (4), this series of transactions can be verified by security agents. The Travel Agencies (3) are used as a back-up manual measure for those who do not have the time to search, book etc., and are more comfortable paying others to do these activities for them. Security issues handled by security agents may include a need for user authentication and data encryption/decryption. In specifying client needs and web services providing them, we need to specify these security requirements, and the relevant Multi-Agents will interact, coordinate and negotiate with each other to produce an optimal solution. Aspectual constraints and their required/provided properties are used in testing and validating any discovered service.

To support better dynamic discovery, integration and subsequent consumption of services in web-service based systems, we designed and developed Intelligent Aspect-oriented Web Services (IAOWS) using Multi-Agents. This research further extends our AOCE work in which we developed extensions to the object component model to support component design, de-coupled implementation and run-time discovery and integration using component aspects (Dong et al 03, Coyler and Clement 04). Component aspects are cross-cutting concerns impacting on components, including persistency management, distribution, security, transaction processing and resource use. Components provide capabilities to others or require services from them across these different system aspects. Aspect details capture functional and non-functional properties and allow design-time reasoning and run-time component description and adaptation.

Key aspects that multi-agents use when discovering web services to interact with include security model, transaction management, performance measures for operations, and fault and exception-handling approaches. As such, when building web services we may describe data persistency approach, database transactional behaviour for operations, resource utilization, communications infrastructure, monitoring and logging, etc. During discovery and integration, we may need to locate adaptors, transaction managers, and security managers, and compose (or orchestrate) services. We aim to better support this range of activities when designing, implementing and deploying web services using IAOWS. We have developed a model of IAOWS-based systems, together with a variety of multi-agents, and proof-of-concept implementation of the model with .NET web services.

8.1.2 Multi-Agents Requirements Engineering

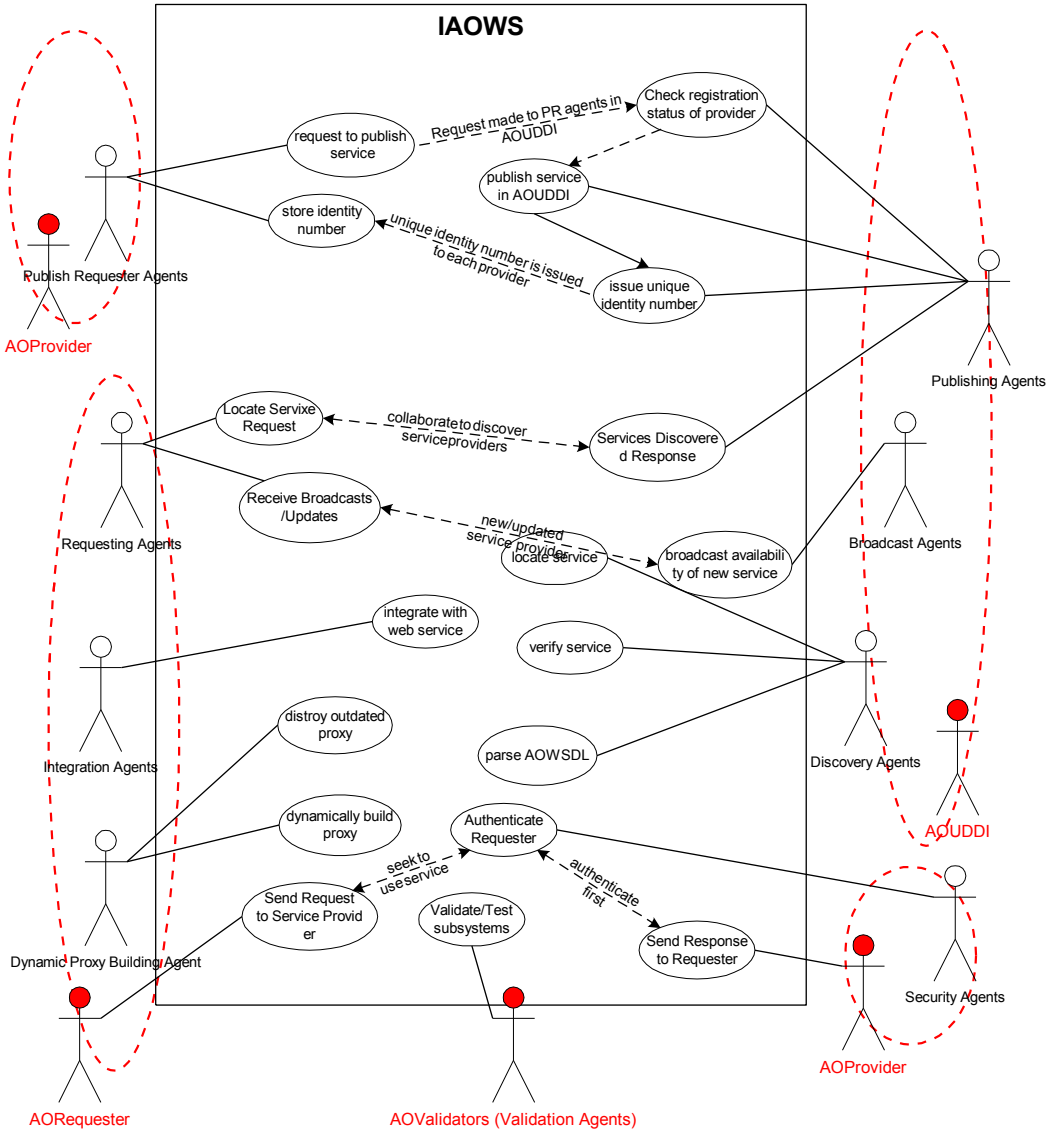


Figure 8.2: Use cases of the agents in IAOWS

Figure 8.2 above shows the use cases of the multi-agents involved in the IAOWS. The filled stick-figures in red are the subsystems of IAOWS and the remaining stick-figures within the broken-lined boundary are their respective aspect-oriented multi-agents. All communication between these distributed and collaborating agents if located on different machines is done in XML format using the SOAP over HTTP

protocol. Listed below are the requirements specification describing the use cases of the multi-agents shown in the above figure:-

(1) Aspect-oriented web service providers use Publish Requester Agents to publish their services to the AOUDDI. Publish Requester Agents request the Publishing Agents in the AOUDDI registry to publish its service by transmitting and depositing its AOWSDL document with the registry.

(2) The Publishing Agents in the AOUDDI coordinate with the Publish Requester Agents by first doing a status check of the provider, i.e. check whether the provider has already registered previously or not.

(3) If the document does not exist in the registry, i.e. if the provider is not registered, the Publishing Agents will issue the Publish Requester Agents with a unique identity number called an ID_Publishing number.

(4) If the AOWSDL is an updated version or a new provider has just registered, the Publishing Agents will call the Broadcast Agents in the AOUDDI to broadcast to all requesters via their requesting agents registered with the AOUDDI that a new version of the AOWSDL document from a particular service provider is available.

(5) The requesting agents will verify with their integration agents whether the particular web service is currently being consumed. If the reply is positive, the integration agents will give a complete list of services used and their required aspectual features to the requesting agents.

(6) The requesting agents will then communicate with the discovery agents in the AOUDDI to verify whether the required services are still available from the provider. It does multiple XML queries on the discovery agents

which include requesting for particular aspects, the details and provided/required properties. The discovery agents resort to case based reasoning to answer the queries of the requestors.

(7) The discovery agents also have efficient parsers that parse the whole AOWSDL document and pull out all the information within the document and store it in storage devices for detailed look-up purposes.

(8) If an already consumed but updated provider can still provide the services that the requester needs, then the proxy in the requester needs to be updated dynamically. The requesting agents will call the integrating agents to update the reference.

(9) To integrate with a better service provider, the integrating agents will instruct the dynamic proxy building agent in the requester to dynamically destroy the existing unwanted proxy of the web service if the provider is not currently being called to carry out remote processing.

(10) The dynamic proxy building agent then dynamically recreates a new proxy class based on the updated AOWSDL file.

(11) The integrating agents are notified that the dynamic proxy is created and these agents in turn pass on the command to the requester so that the proxy can now be used by clients.

(12) The dynamic proxy building agent can also use reflection to dynamically discover and call all methods, with their parameters and properties on this proxy class.

(13) Requesters can consume the web service by making requests (service calls) to any of the service providers that they have integrated with.

(14) Service providers can process the calls made by the clients and transmit the output back to the service requesters.

(15) Validation agents in the Aspect-oriented Validators can be used to test or validate that the IAOWS system or any of its subsystems are performing as per specification.

8.1.3 Multi-Agents and IAOWS Architecture

In this subsection we further describe the various types of intelligent agents that we used, including describing their tasks and functions, to enable the dynamic discovery and integration of web services to be realized, including consuming the services in IAOWS. We had to be very careful to assign tasks to the correct agents that are most appropriate to handle them and to ensure that there were no overlapping tasks (Shen et al 05). We also had to ensure that the agents were communicating and coordinating with their appropriate counter-parts/subsystems. These agents were run using their own separate threads asynchronously so that they did not hold up processing time and can compete with each other for resources on a first-come-first-served basis. The separate threads also allow for all independent processes to continue execution while waiting for the responses for the respective threads. Figure 8.3 below shows the architecture of the IAOWS used to develop our collaborative Travel Planner system. As depicted here, IAOWS uses the concept of multi-agents and aspects, in this case aspects that impact on different parts of web services that can be captured and utilized by the agents.

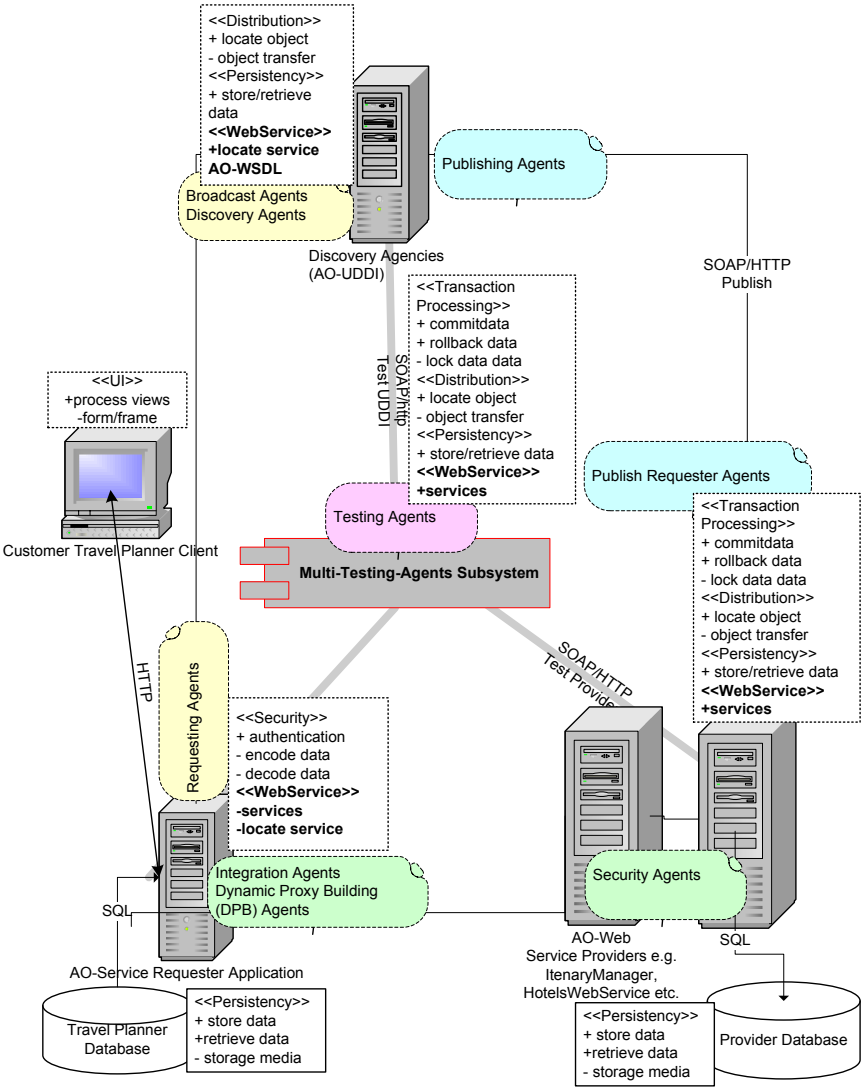


Figure 8.3: The architecture of Intelligent Aspect-oriented Web Services

In this figure, Aspect-oriented web service providers, for example the Hotels web service, Itinerary Manager web service, Flights web service, Payment web service, etc., all use their own Publish Requester Agents (PR Agents) to publish their services to the AOUDDI. PR Agents do this by transmitting and depositing the unique AOWSDL document of their respective providers to the registry. Publishing agents in the AOUDDI coordinate with the PR agents in the providers and if the publishing is successful, issue the PR agents with a unique identity number called an ID_Publishing

number. Publishing agents in the AOUDDI will first check to see whether the document already exists in their repository, and if it exists, check whether it is a duplicate copy or an updated version. No new ID_Publishing number is issued in either of these cases. But if it was a new service the publishing agent will automatically generate and issue a new ID which it will also store in its database together with the AOWSDL document. On the other hand, if the publishing agent checks and discovers that there is already an exact copy registered and deposited in its repository, then the redundant AOWSDL document that was submitted is discarded and no further action needs to be taken by the publishing agents. All actions taken and processing done by the publishing agents is also stored in a cache so that this cache can be indexed first and appropriate action taken almost instantaneously. This makes the agents more efficient than agents in systems without caches and can also safeguard against multiple attacks by unscrupulous providers making repeated publications to overload the AOUDDI.

If it is an updated AOWSDL version, the publishing agents will call the broadcast agents in the AOUDDI to broadcast to all requesters via the requesting agents in the Travel Planner client registered with the AOUDDI that a new version of the AOWSDL document from a particular service provider is available. The requesting agents will verify with their integration agents whether the particular web service is currently being consumed. If the reply is positive, the integration agents will give a complete list of services used and their required aspectual features to the requesting agents. The requesting agents will then communicate with the discovery agents in the AOUDDI to verify whether the required services are still available from the provider. It does multiple XML queries on the discovery agents which include requesting for particular aspects, the details and provided/required properties. The discovery agents

resort to case based reasoning to answer the queries of the requestors. The discovery agents also have efficient parsers that parse the whole AOWSDL document and pull out all the information within the document and store it in Hash Tables for detailed look-up purposes. All communication between the distributed and collaborating agents on different machines is done in XML format using the SOAP over HTTP protocol.

If an already consumed but updated provider can still provide the services that the requester needs, then the remote web reference (proxy) (Gannod and Bhatia 04) in the Travel Planner client needs to be updated dynamically. The requesting agents will call the integrating agents to update the reference. The integrating agents will instruct the dynamic proxy building (DPB) agent in the requester to dynamically destroy the existing proxy of the web service if the provider is not currently being called to carry out remote processing. The DPB agent then dynamically recreates a new proxy class based on the updated AOWSDL file, adding all the relevant assemblies and functions to it in C#. It then compiles the AOWSDL proxy class into a dynamic link library (.dll) file, saves it in the BIN (.NET's binaries) folder and adds a reference to it for the requester. This completes the task of dynamically updating the proxy class. The DPB agent can then use reflection to dynamically discover and call all methods, with their parameters and properties on this proxy class. The integrating agents in the Travel Planner application are notified that the dynamic proxy is created and these agents in turn pass on the command to the requester so that the proxy can now be used by clients.

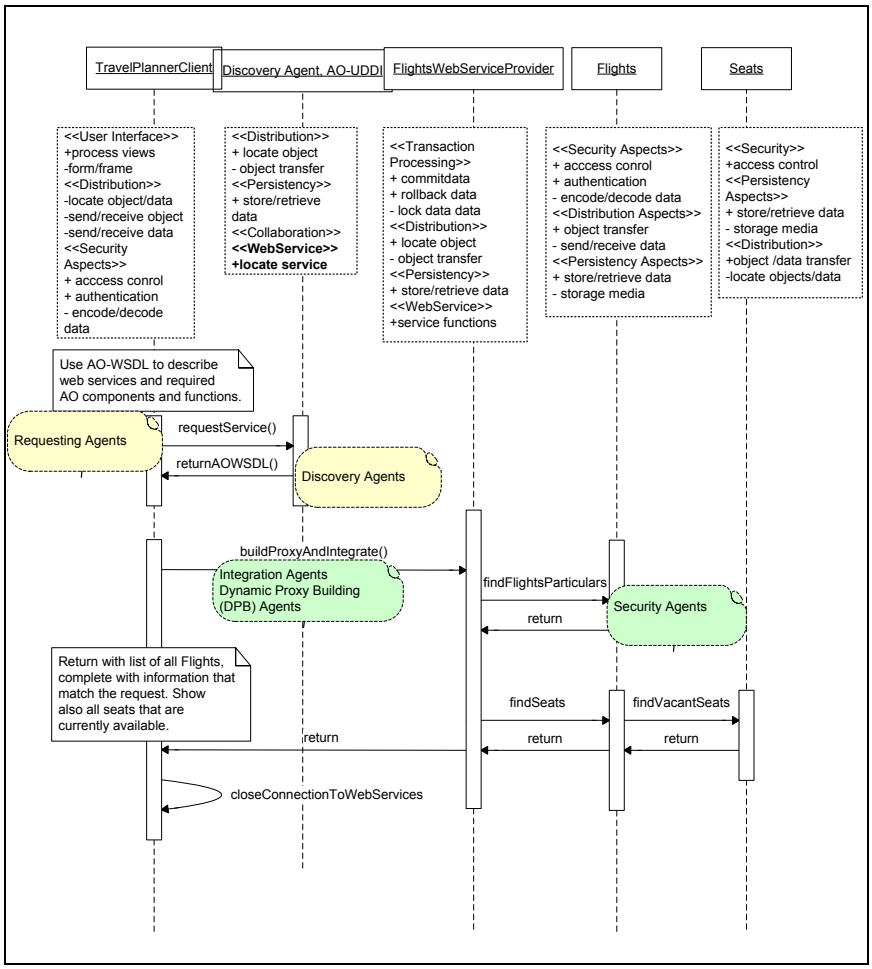


Figure 8.4: Sequence diagram showing dynamic discovery, integration and consumption of a flights web service using multi-agents

Figure 8.4 above shows an example of an AO-Sequence Diagram for the collaborative Travel Planner with aspects and Multi-Agents, it describes the sequence of events for searching, integrating and consuming a Flights web service. The requesting agents here request for a flights service for making reservations on flights. Discovery agents do an AI search and return a best matched service based on the request. An interplay of AI agents with the help of aspect-enriched queries and responses together with coordinated effort allows this to be achieved. The Multi-agents shown here follow all the procedures and transactions explained earlier in this section. Shown here also are security agents that only allow service bindings and interactions with clients that are

authorized to use the services. This is achieved by inserting secret encoded keys (Adams and Boeyen 02) for access to the web service by the client in its XML requests that are deciphered by the security agents. Access is only allowed if the correct code is used by the client, and this kind of transactions are for instance used to authorize staff to edit databases entries etc. that normal customers do not have permission to do.

8.1.4 Implementation of IAOWS

We designed and developed a prototype collaborative Travel Planner based on the IAOWS model of deploying Multi-Agents in a remotely connected server that can dynamically discover and integrate with relevant aspect-oriented web service providers so that users can use it to plan and make bookings for various itinerary items for their travel or holidays. Figure 8.5(a.) below shows a section of the GUI of this web-based Travel Planner developed using AOCE. It shows the web form (a file type having its extension as .aspx) of the application used to search and subsequently book and make payments for flights to particular destinations. We also implemented a trimmed down version of the GUI containing all its functionalities in a smart device application shown in Figure 8.5(b.).

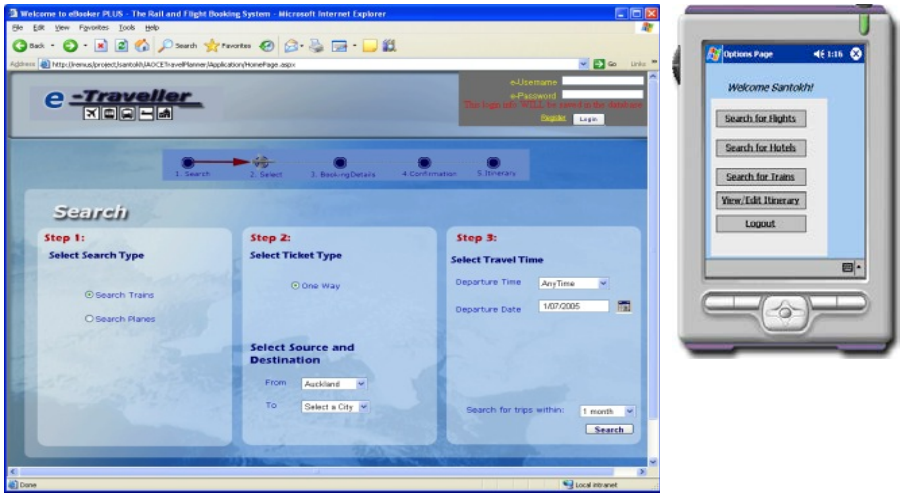


Figure 8.5: Travel planner applications (a.) web based in PC (b) smart device application that interacts with the web services.

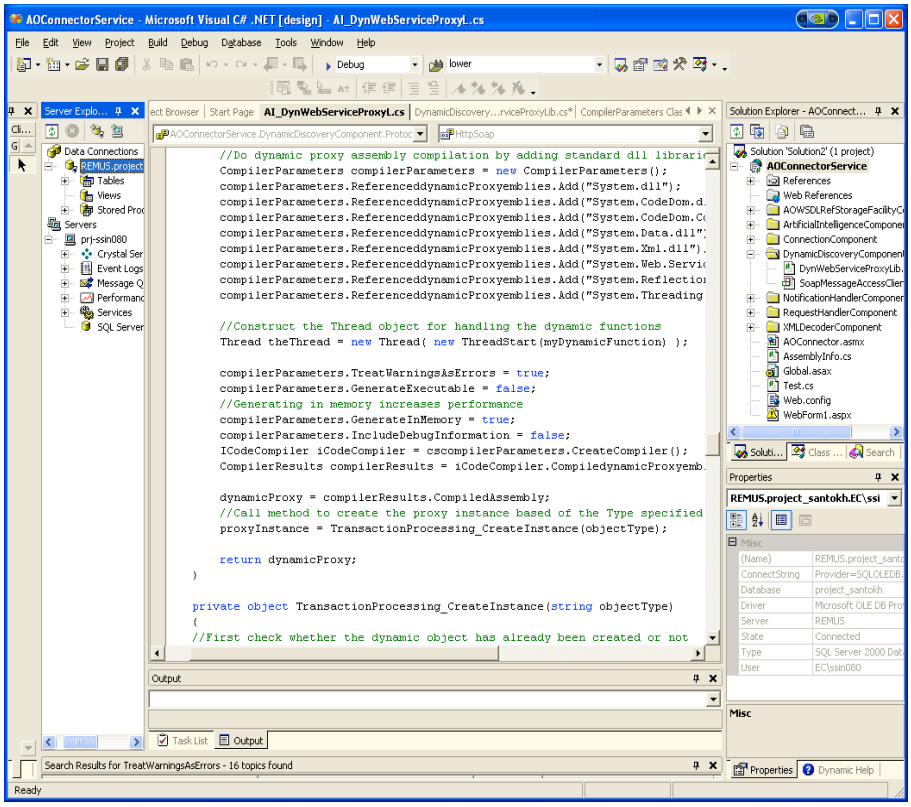


Figure 8.6: C# Code of the dynamic proxy building (DPB) agent in the requesters.

Figure 8.6 shows a sample of the code written in C# of the dynamic proxy building (DPB) agent in the requesters. This agent is used to dynamically create or update a

web service proxy based on an AOWSDL file found to be suitable through the coordinated effort of the requesting agents in the clients and discovery agents in the AOUDDI. Integrating agents instruct the DPB agent in the requester to dynamically destroy the 'old' proxy (if any) first before it creates a new proxy class based on the discovered service document. As can be seen, all the relevant assemblies and functions are added to the proxy first. The AOWSDL proxy class is then generated, compiled and referenced by the DPB agent so that it can be used by the requester. The integrating agents dynamically create instances of this proxy to enable remote execution of operations on it using the SOAP/HTTP as the transport protocol.

All agents are placed within well defined aspect-oriented components so that they can be easily reused and to achieve better modularity. This high level of modularity allowed us to rapidly and accurately refactor, update and test our AI algorithms, especially algorithms for conducting searches and case based reasoning purposes. Also aspect-oriented components address cross cutting concerns (Kiczales et al 97, Lieberherr 99) and are better characterized and categorized. Implementation was done entirely in C# using Microsoft's Visual Studio and the .NET Framework so that we could concentrate on the core issues involving the architecture, design and deployment of Multi-Agents within IAOWS without the strain of learning and debugging in a multitude of languages.

8.2 Discussions about the Multi-Agents and IAOWS

We had to carefully plan and limit the number and different kinds of agents in our IAOWS system to an optimal and controllable number (Shen et al 05) so that the series of tasks executed by each type of agent is clearly defined and not overlapped

with other types of agents. We had to define the tasks precisely so that when it needs to be executed we know exactly which agent to call into action. Since we already had experience designing and developing the prototype of AOWS (i.e. without the multi-agents), this extension to incorporate the multi-agents was not too difficult as we already had the technical knowledge and expertise of how the distributed system works. We reused the code and aspect-oriented components from the earlier prototype, and refactored where necessary, and this made our development based on AOCE techniques more efficient and effective.

The AOConnector object had to be dispensed with when using multi-agents because the connector became too bulky and inefficient due to the number of agents that had to be increased dramatically in the AOWS system to support the transactions and operations in the AOConnector if it were to be used in conjunction with the multi-agents. It also had become increasingly difficult to keep track of the numerous agents in the AOConnector collaborating and negotiating tasks with the agents in the other subsystems. This lead to some agents in the AOConnector duplicating tasks already done in other subsystems and gave rise to redundant agents, decreased efficiencies and a drop in understandability and performance.

We used AI algorithms e.g. CBR in discovery agents mining the AOUDDI repository, and A* search algorithms to choose the best web service that meets the search criteria composed of aspect-enhanced multiple queries. As the number or registered web services in the repository of the registry grew, searches became slower because of the huge amount of information that had to be processed based on the criteria the agents had to match and satisfy. We tried using other algorithms like Best-First Search, Greedy Search and Means-Ends Analysis, but we abandoned them as the results obtained through these techniques were not as good as the A* Search which merges

two heuristic functions into one superior function and this can satisfactorily process both aspectual and non-aspectual queries.

We also discovered that the deployment of agents increased the modularity in our software systems, but we had to dispense with the AOConnector object to reduce the complexities associated with the increase in modularity and higher coupling due to the introduction of the various types of multi-agents into the AOWS system. All our sub-systems have become more lightweight as we have extracted a multitude of key operations/components from our requesters, providers and AOUDDI subsystems and placed them in Multi-Agents. In our updated system that we presented in this chapter, we now just need to call these agents to carry out their specific tasks in the software. As such, our IAOWS system and its aspect-oriented components are now easier to reuse and refactor, making the overall system more maintainable and scalable.

In our ongoing and future work we are looking into avenues of extending and deploying these agents in a semantic aspect-oriented web services system that we are currently in the initial stages of designing and formulating. We will also add in other co-ordinating agents here to carry out any additional tasks involved due to the introduction of the new features. We believe that the semantics and aspects will give the agents their full power and capabilities to carry out more comprehensive and accurate dynamic discovery, integration and consumption of web services within our IAOWS framework.

8.3 Summary

This extension to our research where we had designed and developed Aspect-Oriented Web Services without the use of Multi-Agents (that was explained and discussed in the earlier chapters) is also a particularly significant phase in the development of web-

service based systems that can support automation in the area of dynamic discovery, integration and subsequent consumption by clients through the use of Multi-Agents and aspectual features. It allows us to come nearer at realizing the dream that web services can indeed cater for more extensive dynamic application to application communication without human intervention. The Multi-Agents deployed here not only addressed the issues that hampered dynamic look-up and integration capabilities of web-service based systems, they also made such systems more modular, maintainable, reusable and scalable as compared to traditional non Aspect Oriented web service based systems.

Chapter 9: Using Formal Methods in Alloy to Model, Analyze and Verify AO Web Services Systems

In this chapter we describe and discuss in-depth the use of a formal modeling language called Alloy (Jackson 02), to model, analyze and verify that the novel abstractions in our aspect-oriented web services system (AOWS) that we have designed and developed are in fact formally feasible and that they are logically and mathematically correct. Alloy is based on first-order logic and can be applied to model, analyze and dynamically validate complex software systems (Dong et al 03). We can specify the software structural properties in this language and further use the Alloy Analyzer (Alloy_homepage 05) tool to automatically analyze the specifications of the model. We will first describe and specify the AOWS structural objects and properties in Alloy and then use it to provide evidence that the aspect-oriented web services system functionalities can be abstracted from the implemented prototype Travel Planner according to its design and implemented specifications. This will provide invaluable insights into the various abstractions and subsystems of the AOWS, including the bindings and interactions between them, which in turn will enable us to clearly identify and understand their significance and impact on the various components in our implemented system, thus giving us greater understanding and insights about our AOWS system from a mathematical modeling point of view.

We first give an overview of Alloy and the concepts involved in using it for formal modeling purposes. We then elaborate and build on all the relationships in our AOWS system for our model. Once the model is constructed, we will further use the Alloy Analyzer tool to analyze and verify the AOWS system.

9.1 Alloy overview

ALLOY is a higher level parallel programming language appropriate for programming massively parallel computing systems. It is based on a combination of ideas taken from functional, object oriented and first-order logic programming languages (Language_list 05). It is similar to Z or OCL, the Object Language of UML, but Alloy is designed more for automated model analysis and verification compared to the other two languages. Alloy is specifically targeted at the creation of micro-models of complex software systems because they can then be automatically checked for correctness (Jackson MITLab 02). The usefulness of Alloy is evident in the fact that it does not require the mechanism to describe the effect of a behavior. This conveniently allows us to divide our aspect-oriented web services model, which is really one very large model, into several smaller and simpler models, which will be much easier to analyze and verify.

Alloy is a based on first-order logic that can be used for structural modeling by expressing complex structural constraints and behaviors. Alloy treats relations as first class citizens and uses relational composition as an operator to combine various structured entities. The essential constructs (Alloy_tutorial 05) of Alloy used for AOWS modeling, analysis and validation purposes are as follows:

- A signature (sig) is a paragraph that introduces a basic type, a collection of relations (called fields), and a set of constraints on their values that can be defined in our AOWS. A signature may inherit fields and constraints from another signature.

• A function (fun) evaluates the first order expressions into a value. It is a parameterized function that can be used in other expressions.

• A predicate (pred) captures behavior constraints in our AOWS and evaluates them into true or false. It is a parameterized formula that can be further applied in other constraints.

• A fact (fact) imposes global constraints on the relations and objects. A fact is a formula that takes no arguments and need not to be invoked explicitly. It acts as axioms in the model, which is always true.

• An assertion (assert) specifies an intended property in the AOWS system. It is a formula whose correctness needs to be checked, assuming the facts in the model.

Given below is a summary of some of the more important and commonly used Alloy Semantics (Alloy_tutorial 05) that are also used for modeling the AOWS in this thesis. The following symbols written within single quotes show the sequence of characters that are recognized as semantic tokens:

• '=>' represents the implication operator;

• '>=' and '=<' represent integer comparison operators;

• '->' represents the product arrow;

• '<:' and ':>' represent restriction operators;

• the double colon'::' is used for receiver syntax;

• '++' represents the relational override operator;

• '&&' represents conjunction and '||' disjunction;

• '—', '//' or '/* and */' all represent comment markings.

The negated operators (such as !=) are not treated as single tokens, so they may be written with white-spaces between the negation and comparison operators in our models. Note also that in Alloy the set notation operations of "||"and "&&"has the usual meanings of "or" and "and" respectively.

We next discuss the Precedence and Associativity rules used in Alloy. The precedence order for logical operators, tightest first, is as follows:

• negation operators: '!' and 'not';

• conjunction: '&&' and 'and';

• implication: '=>', '<=>', 'implies' and 'iff';

• disjunction: '||' and 'or'.

The precedence order for expression operators, tightest first, is:

• unary operators: '~', '^' and '*';

• restriction operators: '<:' and ':>';

• dot join: '.' ;

• square brackets means join: '[]';

• arrow product: '->';

• intersection: '&';

• override: '++';

• union and difference: '+' and '-'.

Note that in particular dot binds more tightly than square brackets, so a.b[c] is parsed as (a.b)[c], i.e. with the dot evaluated first.

All binary operators are associative, except for: the logical implication operator, and the expression operators dot, intersection and difference. Implication associates to the

right, and the expression operators associate to the left. So, for example, p => q => r is parsed as p => (q => r), and a.b.c is parsed as (a.b).c.

Given below are the notations and meanings of the operations used and samples of examples extracted from our AOWS models that use these operations.

- '=' means "equal". It doesn't assign a value.

e.g. aowsRequester.request = myRequest

This means both the left hand set and the right are equal. It doesn't assign the value of myRequest to aowsRequester.request in the example above, as would have happened in programming languages like Java, C, C++, C#, Visual Basic etc.

- '+' means "union".

e.g. myAOConnector'.aowsdl = myAOConnector.aowsdl + myAOConnector.newlyAdvertisedAOWSDL

This means that myAOConnector'.aowsdl should be equal to myAOConnector.aowsdl unioned with myAOConnector.newlyAdvertisedAOWSDL.

- '-' means exclude"

e.g. myAOComposite'.aowsdl = myAOComposite.aowsdl - myAOComposite.oldAOWSDL

This means that myAOComposite'.aowsdl should be equal to myAOComposite without the myAOComposite.oldAOWSDL entity.

- 'in' means that the value is "within" the set.

e.g. myAOComposite.oldAOWSDL in myAOComposite.aowsdl

This means that myAOComposite.oldAOWSDL should be within the set myAOComposite.aowsdl, i.e. within the set of all AOWSDL documents in the composite object before we can remove the unwanted AOWSDL (oldAOWSDL) from the composite.

The above constructs and semantics are used extensively to model ours AOWS system. After modeling we pass the models through the Alloy Analyzer which is a tool for analyzing models written in Alloy. Given a finite scope for a specification, the Alloy Analyzer translates it into a propositional formula and uses SAT solving technology to generate instances that can satisfy the facts and properties expressed in the specification. In other words, given a formula and a scope, a bound on the number of atoms in the universe, it determines whether there exists a model of the formula (i.e., an assignment of values to the sets and relations that makes the formula true) that uses no more atoms than the scope permits The Alloy Analyzer provides two kinds of risk analysis for our AOWS system. The first risk is that the constraints given are too weak. Flaws of this sort are found by the Alloy Analyzer by the checking assertions, in which a consequence of the specification is tested by attempting to generate a counterexample. The second risk is that the constraints given are too strong; in the worst case, the constraints contradict one another and all possible states are ruled out. Flaws of this sort are found in simulation in which the consistency of a fact or function is demonstrated by generating a snapshot showing its invocation.

Alloy and its Analyzer have been used primarily to explore abstract software designs. Its use in analyzing code for conformance to specifications and as an automatic test

case generator is being investigated in ongoing research projects (Alloy_homepage 05).

The key features of Alloy are that it allows for the expression and modeling of complex structures to be defined and constructed with just a few powerful operators. It can be applied to both static and dynamic structures and is highly declarative, i.e., it has a full logic, including conjunctions and negations, and describes a system as a collection of constraints. It is analyzable, i.e. it caters for both simulation and checking. The Alloy tool is fully automatic, with no user intervention required and can generate concrete samples and counterexamples. But just like any testing, it is sound but not complete. Alloy's analysis can execute a model forwards or backwards, even without test cases, and has no ad hoc restrictions on logic. As such, the Alloy and its Analyzer is a very useful tool and suits us very well in modeling, analyzing and verifying Aspect-Oriented Web Services systems which have some very complex and sophisticated structures, both static and dynamic.

9.2 The AOWS Relationships to consider in Alloy

In Alloy everything is depicted as meaningful relations between the various entities existing within the particular system that is being modeled. Figure 9.1 below shows the ten important relationships that exist between the various entities of the Aspect-Oriented Web Services system. Each relationship, shown in the figure as numbered and enclosed within a circle, is explained in-depth below.

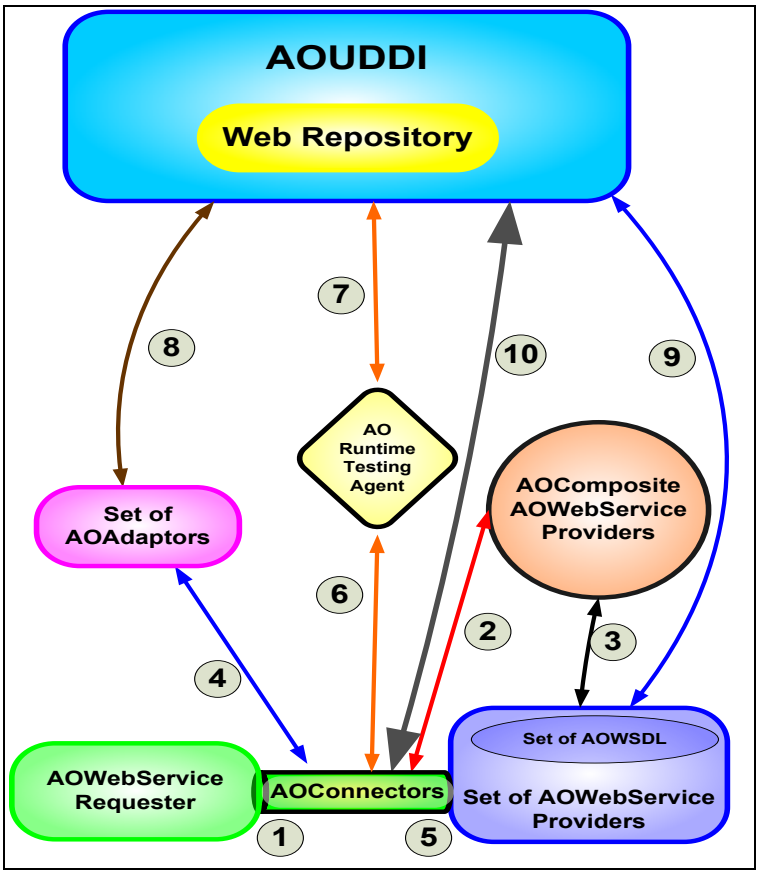


Figure 9.1: The AOWS architecture showing its relationships.

The numbers depicted in Figure 9.1 above represent the following relationships:

1.) The *first* relationship represents the interaction between the AOWebServiceRequester and an AOConnector. Here the requester uses the AOConnector object to communicate with a variety of other useful sub-systems, i.e. the AOUDDI, the set of AOWebServiceProviders, the Composite of AOWebServiceProviders, the Runtime Testing Agent and the set of AOAdaptors. The requester would be able to locate appropriate web service providers according to the required components or aspects specified in its SOAP request by utilizing the AOConnector object to conduct the appropriate searches in the AOUDDI registry shown above.

2.) The *second* relationship represents the specific communication between the AOConnector and the Composite set of AOWebServiceProviders. If the AOWebServiceRequester requires a variety of services that cannot be satisfied by a single provider but can be satisfied collectively by consuming a number of service providers, the composite object will select the set of appropriate providers from the available web service providers as required by the requester.

3.) The *third* relationship flows from the *second* relationship and depicts the communication between the AOComposite and set of AOWebServiceProviders. For example, a particular user may wish to carry out the complex task of searching for and booking a flight or train to a holiday resort (hotel) and further arranging to make payment for it. This will mean that the requester will need to consume multiple web service providers including, for instance, the AOHotelWebServiceProvider, AOFlightWebServiceProvider or AOTrainWebServiceProvider, AOPaymentWebServiceProvider and AOItenaryWebServiceProvider from the

AOWS system. All the relevant service providers will be selected and bundled together by the AOComposite object and supplied to the AOConnector object (as depicted in the *second* relationship) to be integrated with the AOWebServiceRequester (as per the *first* relationship). The relationship between the Composite AOWebServiceProviders and the AOConnectors involves also the transfer of a details (e.g. the set of AOWSDL file locations) from the AOConnector to the Composite AOWebServiceProvider. According to the details provided the AOConnector can, through the use of the composite object, call the appropriate aspects or components contained in the set of providers to be passed on and used by the AOWebServiceRequester. This kind of indirect connection that involves the connector using another intermediate object (the AOComposite) to access and consume service providers is called an "Indirect Connection" in our modeling and analysis of AOWS in Alloy.

4.) The *fourth* relationship is between the AOConnector and the set of AOAdapters. An AOConnector object will make use of a specific adapter when some particular service needs to be adapted for use by the AOWebServiceRequester. AOAdaptors are used to modify and translate aowsdl files written in different protocols or formats so that standard aowsdl parsers can interpret and use them.

5.) The *fifth* relationship shown in figure 9.1 represents the interaction between the AOConnector and individual AOWebServiceProviders. The connector object make service calls to the appropriate aspects or components contained in the of provider concerned and process and/or pass the response on to the requester. The

connector connects directly with the service providers and this connection is called a "Direct Connection" here.

6.) The *sixth* relationship is between the Runtime Testing Agent and the AOConnector. The Runtime Testing Agent continuously monitors the whole system for any type of bugs or errors. These are recorded and could include, for example, caught exceptions or logged events like the AOConnector not being notified when a new AOWebServiceProvider gets registered in the AOUDDI.

7.) The *seventh* relationship encompasses the Runtime Testing Agent monitoring the AOUDDI System in order to make sure that the AOUDDI is functioning as per specifications. For instance, after a new AOWebServiceProvider gets registered with the AOUDDI, the Runtime Testing Agent will check whether or not the AOWSDL of the AOWebServiceProvider gets stored in the Web Repository of the AOUDDI System.

8.) The *eighth* relationship is between the AOUDDI and set of AOAdapters. The AOUDDI will pass the XML document containing the information of any new AOWSDL that has been registered in the AOUDDI to the AOAdapter. The AOAdapter will then check whether the new AOWSDL is in proper format (for instance, the relative position of the aspectual elements might have been mixed-up), if it is not it will convert the AOWSDL to the proper format and pass the converted document back to the AOUDDI..

9.) The *ninth* relationship illustrated in Figure 9.1 above involves the *registration (and subsequent advertisement to service requesters)* of the AOWebServiceProviders to the AOUDDI system. This happens by registering and publishing the provider in the AOUDDI and it includes storing the provider's AOWSDL file in the AOUDDI's repository. If registration was successful, the AOUDDI will send the service provider a unique ID for ease of keeping track of provider and to facilitate any future updates/communications with the provider.

10.) The *tenth* relationship is between the AOUDDI and the AOConnector. This relationship includes a *notification* to the AOConnector that a new AOWebServiceProvider has registered with the AOUDDI registry. The notification involves the sending of the AOWSDL location and other relevant information about the new AOWebServiceProvider to the appropriate AOConnector(s). When the AOUDDI tool registers any new service provider it will relay the information about the provider to the relevant connectors. The *tenth* relationship also involves a request from the connector to the AOUDDI for the discovery of service providers and the response from the AOUDDI to the connector. This request originates from the AOWebServiceRequester and is passed on to the AOUDDI through the connector object. The AOUDDI will respond by providing the appropriate AOWebServiceProvider(s) details based on the request.

9.3 Modeling the AO Web Services Systems Specification using Alloy

Using the constructs and semantics in Alloy we formally modeled AOWS based on the inter-relationships between its subsystems described above. Samples of the

signatures used to construct the Alloy model are described here. The full list and description of all the signatures used for the AOWS model is too large to include in this sub-section completely and can be referred to in the Appendix. Figure 9.2 below shows the signature of the AOConnector, central to our architecture, which acts as a conduit-cum-buffer for the flow of information, instructions, requests and responses between the various subsystems that make up AOWS. Each client connects to only one connector and vice versa. For both dynamic and static functional systemic purposes, the connector needs to store and know about any updated, useful and current information about all the relevant and available aspect-oriented web service providers through their respective AOWSDLs, the composite and any web service that is consumed by the client, as shown in the fields of the connector.

```
sig AOConnector{
      aocomposite : lone AOComposite,
      directlyConnectedAOWS : set AOWSDL,
      newlyAdvertisedAOWSDL : lone AOWSDL,
      chosenAOWSDL : lone AOWSDL,
      oldAOWSDL : lone directlyConnectedAOWS
}

sig AOWebServiceRequester{
      aoconnector : AOConnector,
      newlyAdvertisedAOWSDL : lone AOWSDL
}

sig AOWebServiceProvider{
      aowsdl : set AOWSDL
}

sig AOWSDL{
      aoComponents : AOComponents
}

sig AOComponents{
      name : String,
      aoComponent : set AOComponent,
      aoDocumentation : AODocumentation,
      aoWSDescription : AOWSDescription
}

sig AOComponent{
      name : String,
      aoComponentDescription : AOComponentDescription,
      functionalAspect : set FunctionalAspect,
      nonFunctionalAspect : set NonFunctionalAspect
}

sig FunctionalAspect {
      type : String,
      aspectName : String,
      aoWSEntryPoint : Boolean,
      standalone : Boolean,
      aspectDetail : FunctionalAspectDetail,
      userOperation : String,
      returnType : String,
      parameter : Parameter
}
```

Figure 9.2. AOConnector, AOWebServiceRequester, AOWebServiceProvider, AOWSDL and related aspectual signatures used to model AOWS.

Figure 9.2 also defines the signatures for the AOWS requesters (AOWebServiceRequester) and providers (AOWebServiceProvider). Each Requester connects to all other AOWS subsystems through its unique connector and makes requests and obtain responses through it. The most important feature of a provider for

any client is the AOWSDL document that exposes its services. We model each AOWSDL as a set of AOComponents that are a collection of aspect-oriented components. Each AOComponents set contains a unique name for its provider, a set of AOComponent(s), an AODocumentation, and AOWSDescription. The AODocumentation is human readable and contains crisp and summarized information about the advertised services including aspect-oriented components that are exposed but resident in the service provider. AOWSDescription, is a machine readable element. It is for robots to decipher and as such contains less descriptive language, and is used for dynamic discovery and integration. In addition, AOWSDL also contains elements describing completely the service's definitions, types, messages, operations, port types and bindings, including those for importing further service description documents.

Each AOComponent (which is contained in the AOComponents set) contains sets of FunctionalAspects and NonFunctionalAspects. FunctionalAspects are a collection of aspects related to specific aspectual functions, for instance persistency aspects having aspect details catering for search, update, delete and insert operations involving persistent data storage, editing and retrieval, whereas the set of NonFunctionalAspects are not specific to core business functionality, e.g. memory management and performance aspects. Each component also has a name and an AOComponentDescription. The name is used as an identity string object within that particular AOWSDL and AOComponentDescription describes what the particular component does.

```
fact { no aowsProvider1,
      aowsProvider2 : AOWebServiceProvider |
      aowsProvider1.aowsdl = aowsProvider2.aowsdl }

fact { all myAOWSDL : AOWSDL |
      (one aowsProvider : AOWebServiceProvider |
            myAOWSDL in aowsProvider.aowsdl) }

pred DirectConnectionToNewAOWS ( myAOConnector' : AOConnector,
myAOConnector : AOConnector ) {
      --precondition
      myAOConnector.newlyAdvertisedAOWSDL !in myAOConnector.aowsdl
      -- update the aoconnector
      myAOConnector'.aowsdl = myAOConnector.aowsdl +
myAOConnector.newlyAdvertisedAOWSDL
}
```

Figure 9.3. Facts and predicates, relating providers, requesters and the AOConnector

The signatures for all the elements of the AOWS system collectively specify all the subsystems and other objects of AOWS in our Alloy models. Figure 9.3 shows a sample from the many facts and predicates that define the structure and behavior of the AOWS models. The facts shown here define the structure of aspect oriented web service providers and state that an AOWSDL is unique to a particular web service provider (i.e. no two AOWSDLs can be exactly the same as they have at least a different URL) and that each AOWSDL must originate from a service provider (i.e. a AOWebServiceProvider) so that requesters can discover, integrate and consume the services. The predicate captures the behavior of an AOCconnector as it dynamically integrates with a new service provider that is found to match the requirements of the requester and makes a direct connection with the provider (i.e. not via an AOComposite object). It also updates the records in the connector to show that the new provider has integrated with the client by adding the AOWSDL of the new provider to the record of the collection of AOWSDLs maintained in the connector.

```
sig SearchForHotel {
      type : Persistency,
      aspectName : String,
      aoWSEntryPoint : Boolean,
      standalone : Boolean,
      aspectDetail : SearchForHotelDetail,
      userOperation : String,
      returnType : String,
      parameter : SearchForHotelParameter}

sig SearchForHotelRoom {
      type : Persistency,
      aspectName : String,
      aoWSEntryPoint : Boolean,
      standalone : Boolean,
      aspectDetail : SearchForHotelRoomDetail,
      userOperation : String,
      returnType : String,
      parameter : SearchForHotelRoomParameter}

sig SearchForHotelDetail {
      type : SearchForHotelDataRetrieval,
      detail : SelectHotel,
      provided : Boolean}

sig SearchForHotelRoomDetail {
      type : SearchForHotelRoomDataRetrieval,
      detail : SelectHotelRoom,
      provided : Boolean}

fact { all searchHotel : SearchForHotel | (one searchHotelDetail :
 SearchForHotelDetail        |        searchHotelDetail        in
searchHotel.aspectDetail) }

fact   {   all   searchHotelRoom   :   SearchForHotelRoom   |   (one
searchHotelRoomDetail :
 SearchForHotelRoomDetail | searchHotelRoomDetail in
 searchHotelRoom.aspectDetail)}
```

Figure 9.4: Alloy code snippet from a formal model of the Travel Planner application.

We also modeled and simulated an aspect-oriented web service based collaborative Travel Planner system based on the concept of our AOWS architecture. This can be used to make comprehensive travel arrangements e.g. searching/booking for flights, hotels, trains etc., and making payments for those services. Figure 9.4 shows a snippet of the Alloy code to formally model this application. It shows the aspects identified together with their respective aspect details, aspect type, its provided/required properties etc. to be used to perform aspectual searches for hotels and rooms in the application by consuming multiple relevant web services. This program serves to

show that besides our AOWS theoretical architecture and framework, even actual AOWS based application can also be modeled, analyzed and verified that they are logically and mathematically feasible using Alloy.

9.4 Generating models and Analyzing AOWS using Alloy

After defining the signatures for the AOWS as described in the previous subsection, the whole system including parts of it can be analyzed and verified through the use of predicates, facts and assertions. These can be processed automatically by using the Alloy Analyzer. The predicates, facts and assertions are used to generate simulations for the inter-relationships between the objects defined in the AOWS system. In the following subsections we illustrate some of the models that were automatically generated using the Alloy tool based on our abstractions and constructs of AOWS.

9.4.1 Alloy model for the relationship between AOWebServiceProvider and AOUDDI

The model illustrated in figure 9.5 below is generated by the Alloy tool and it depicts the *six* main top-level objects, shown enclosed within shaded diamond shapes, of the AOWebServiceProvider that are crucial in its relationship and communication with the AOUDDI. These six top-level objects are the AOWebServiceProvider itself, its unique AOWSDL, AOComponents (which contains the set of AOComponent (s)), AOComponent, NonFunctionalAspects and FunctionalAspects. The terminal elements in this model are shown enclosed within oval shapes and these are the aspects and

their details and behavioral properties like parameters, the operations it uses (UsesOperations) and what other operations use this aspect (AspectUserOperations).

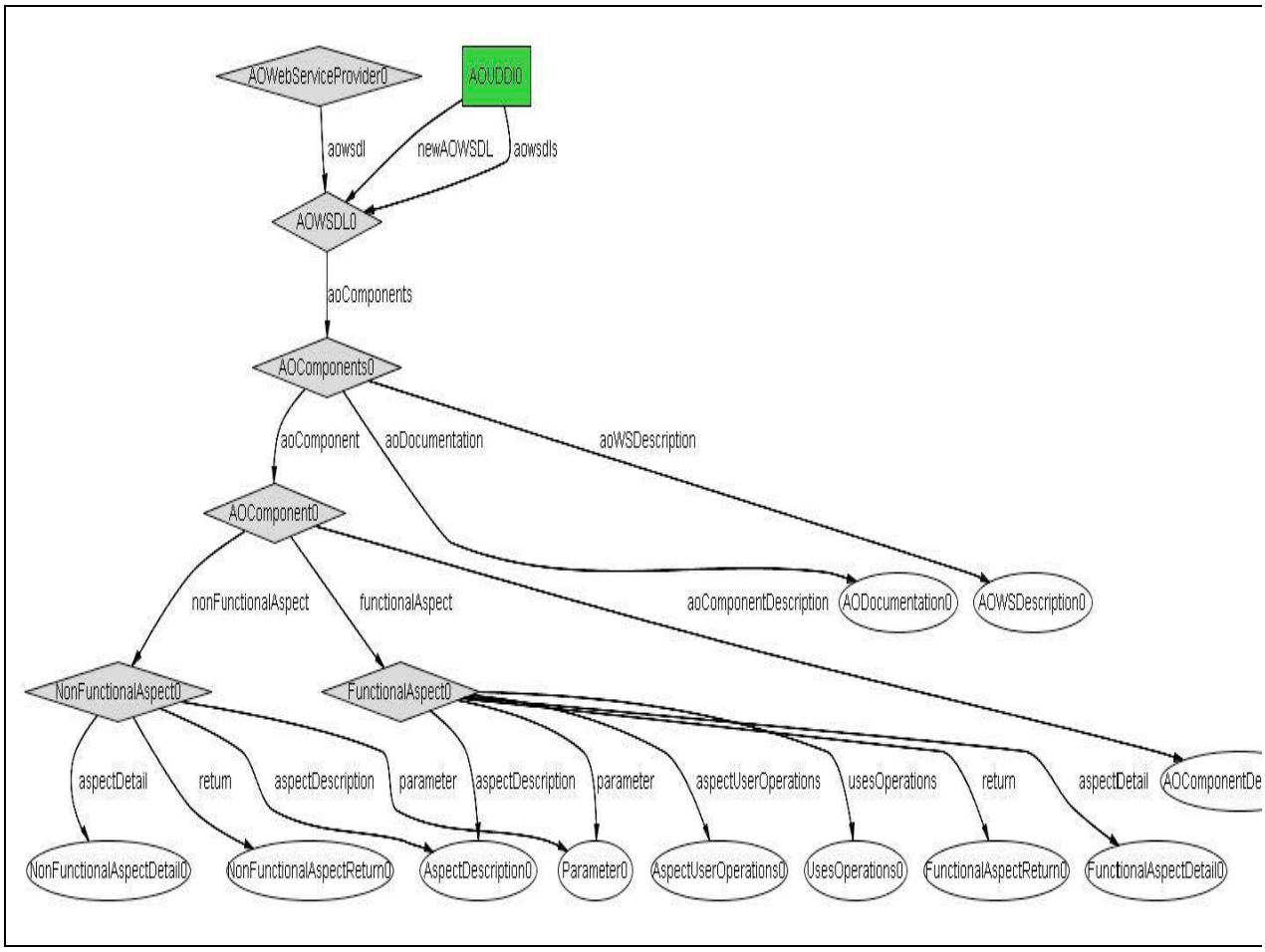


Figure 9.5: Alloy model for the relationship between the AOWebServiceProvider and the AOUDDI

As also shown in figure 9.5 above, each AOWebServiceProvider contains one unique AOWSDL, which is stored in the AOUDDI's repository as part of the set of AOWSDLs. The repository will also be used to store any new AOWSDL as a 'newAOWSDL' if it has just been registered in the AOUDDI but not yet advertised to the requestors. Once it is advertised to the requesters it is not considered to be new

anymore and is stored together with the rest of the collection of AOWSDLs in the repository of the AOUDDI.

Each AOWSDL document as mentioned in Chapter 6 ("Describing and locating AOWS"), besides possessing the 6 major extensible elements of the WSDL, i.e. the definitions, types, message, portType, binding and service elements, it also has a large number of aspectual extensions, all neatly bundled into a main "aocomponents" element. The 'AOComponents' set of the model shown in Figure 9.5 above represents the "aocomponents" element, and it contains the AODocumentation, AOWSDescription and AOComponent(s) set. The AODocumentation and AOWSDescription elements are used to describe the provider. Each AOComponent exposes one or more aspects that describe details about particular cross-cutting features impacting the component e.g. transaction processing, resource utilization, etc. An aspect belongs to either a functional or non-functional type of cross-cutting concern. Each aspect has an aspect type associated with it, e.g. of aspect types include Persistency, Distribution, Performance, Security etc, to categorize the cross-cutting concern. The purpose of this is to enable the description and capture of the rich and highly characterized aspectual features of web services in a systematic and orderly manner. The AOWSDL also allows for more dynamic and automatic searches for any given aspect, aspect details and properties of the services advertised because our AOWSDL specifications for web service components follow consistent, formal and clearly defined semantics and syntax.

9.4.2 Alloy model for the relationship between the AOUDDI and AOWebServiceRequester

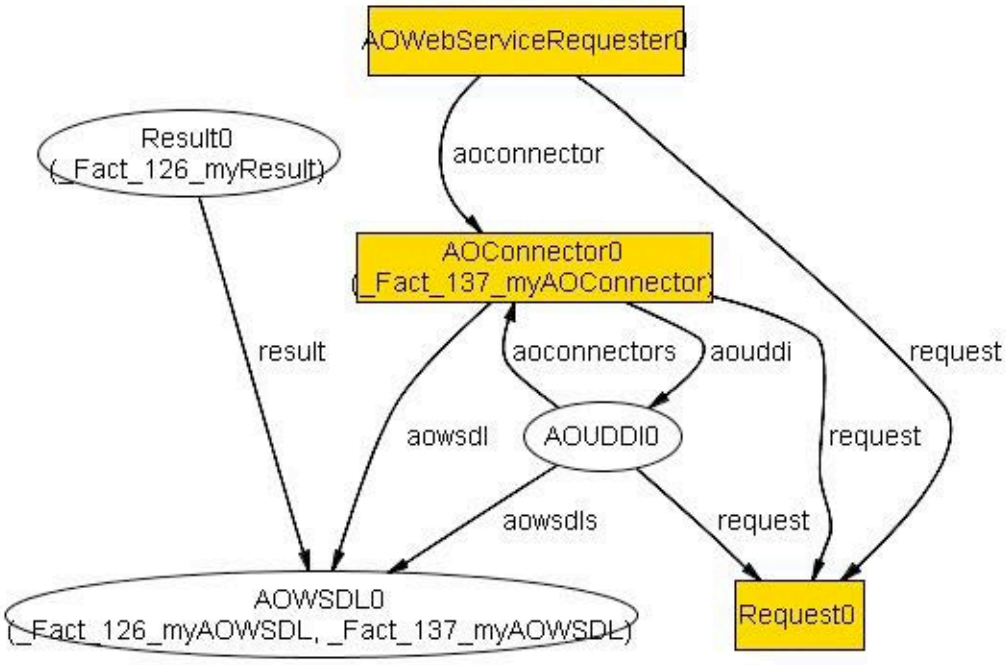


Figure 9.6: Alloy model for the relationship between the AOUDDI and the AspectOrientedWebServiceRequester

The model illustrated in figure 9.6 above shows the four main objects involved in the relationship concerning the communication the between the AOUDDI and the AOWebServiceRequester. The directional lines (arrows) represent “has a” relationships (e.g. the requester “has a” unique AOConnector associated with it and “has” Request objects that can be used to make requests to the AOUDDI through the

AOConnector) based on the fields of the Signatures of the objects concerned that are modeled according to the logical flow of events that will follow from the relationship. The four main objects here are the AOConnector, AOWSDL, AOWebServiceRequester and AOUDDI. The requester makes a request for a particular service to the connector which passes it on to the AOUDDI. This result will contain the AOWSDL of the provider that matches the requirements of the requester. The alloy diagram also shows the facts (numbered above) that were utilized from the program to produce this simulation.

9.4.3 Alloy model for the relationship between the AOWebServiceProvider and the AOWebServiceRequester

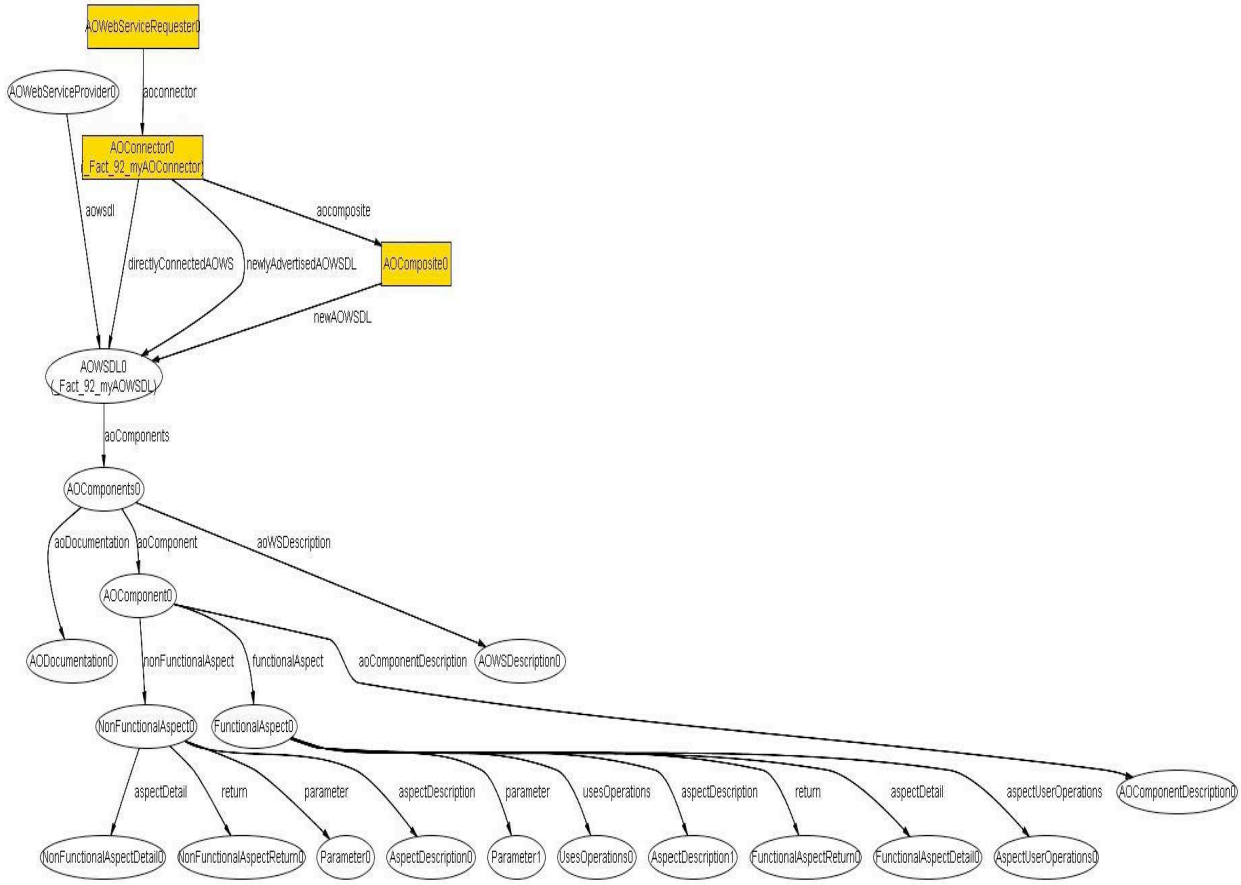


Figure 9.7: Alloy model for the relationship between Aspect Oriented Web Service Provider and Aspect Oriented Web Service Requester

The model illustrated in figure 9.7 above shows the main objects involved in the relationships between the AOWebServiceProvider and the AOWebServiceRequester. The main objects here are the AOWebServiceProvider, AOWebServiceRequester,

AOWSDL, AOComponents (which contains the set of AOComponent(s)), AOComponent, AOConnector, AOComposite, NonFunctionalAspects and FunctionalAspects. The terminal elements in this model are shown enclosed within oval shapes and are from the leaf elements (e.g. aspect description, aspect details, input parameters etc.) from our AOWSDL documents.

All these models have been built based on signatures, predicates and facts that provide constraints that can be checked for correctness by performing runs on the models and looking out for counter-examples. In the next subsection we discuss the application of assertions to our models to verify that they are indeed logically and mathematically correct and feasible.

9.5 Applying Alloy Assertions to Verify the AOWS Model

Six assertions have been applied to the AOWS system to check through simulation whether the system will perform the functions according to its specifications. The assertions have been created according to six different types of scenarios that are representative of all the operations of the system and are elaborated below. If no counter-examples are found by the Alloy Analyzer in its check runs for each and every scenario depicted below based on the signatures, predicates and facts of the program (the full program is listed in the Appendix), then we have successfully modeled a complete Aspect-Oriented Web Service system that is feasible and its construction is correct.

9.5.1 Scenario 1

Here when a new AOWebServiceRequester (client) is first constructed and deployed on the AOWS domain, for it to be able to integrate/consume any web service and become a useful part of the AOWS system, we must first verify whether we can connect it with an AOConnector from our AOWebServices system. All requestors can only access and use the remaining subsystems/objects in AOWS if they are successfully linked to an AOConnector first. We then check whether we can connect that particular AOConnector with the AOUDDI in our model for the purposes of discovery and integration with web services. All these checks are verified based on the use of predicates, functions and facts that are run through the assertion called "*NewConnection*" mentioned below.

The sequence of events that should happen is listed as follows:-

1. The requester will first connect to the AOConnector (as represented by the function RequesterConnectToAOConnector() in the program below) through a series of initializations that involve both the requester and the connector object updating their respective fields.

2. Then link the AOConnector to the AOUDDI and update the AOUDDI about the newly linked connector (AOConnectorConnectToAOUDDI()) so that it can keep track of the activities/requests of the connector and notify it if any new service providers were to register with the AOUDDI.

This sequence of events is simulated in Alloy through the following assertion:

```
assert NewConnection {
   all aowsRequester : AOWebServiceRequester, myAOConnector : AOConnector, myAOUDDI
   : AOUDDI, myAOUDDI' : AOUDDI |
            {
```

```
                    InitializeAOConnector(myAOConnector)
                    InitializeAOWSRequester( aowsRequester )
                    RequesterConnectToAOConnector( aowsRequester, myAOConnector )
                    AOConnectorConnectToAOUDDI ( myAOUDDI, myAOUDDI',
myAOConnector )
            }
} check NewConnection for 2
```

The outcome was successful as no counter examples were found for this assertion.

9.5.2 Scenario 2

This scenario continues from Scenario 1 above, whereby it simulates how a new service provider is dynamically discovered and integrated to the requester by the connector object. Here a new AOWebServiceProvider is introduced into the AOWS domain and must first be simulated to successfully register with the AOUDDI. If it is better than the AOWebServiceProvider currently connected to the requester (i.e. it meets the requirements of the requester by 100%), then it is to be dynamically connected to the AOConnector. The AOConnector will also need to be dynamically disconnected from the earlier inferior AOWebServiceProvider.

The sequence of events that should happen is listed as follows:-

1. First a new AO web service provider registers to the AOUDDI (RegisterNewAOWS()), the AOUDDI will register and publish the provider and update its registry details.

2. Then the AOConnector is notified about the newly registered service provider (NotifyConnectorAboutNewAOWS()).

3. The Requester is notified about the new provider by the connector (NotifyRequesterAboutNewAOWS())

4. A connection is made to the provider directly – i.e. without using AOComposite object (DirectConnectionToNewAOWS())

5. The old web service that needs to be disconnected is selected, this old service is then replaced with the newly discovered one (SelectAOWSToDisconnect())

6. Finally the old web service provider is disconnect from the connector object (DisconnectDirectConnection())

This sequence of events is simulated in Alloy through the following assertion.

```
assert TestDirectConnectionToNewAOWS {
   all aowsProvider : AOWebServiceProvider, myAOUDDI : AOUDDI, myAOConnector :
   AOConnector, aowsRequester : AOWebServiceRequester,
            myAOWSDL : AOWSDL, myAOUDDI' : AOUDDI, myAOConnector' :
AOConnector|
            {
                     RegisterNewAOWS( myAOUDDI, myAOUDDI', aowsProvider )
                     NotifyAOConnectorAboutNewAOWS ( myAOUDDI', myAOConnector )
                     NotifyRequesterAboutNewAOWS ( myAOConnector, aowsRequester )
                     DirectConnectionToNewAOWS ( myAOConnector, myAOConnector' )
                     SelectAOWSToDisconnect ( myAOWSDL, myAOConnector )
                     DisconnectDirectConnection ( myAOConnector, myAOConnector' )

            }
} check TestDirectConnectionToNewAOWS for 2 but 0 Result, 0 Request
```

The outcome was successful as no counter examples were found for this assertion.

9.5.3 Scenario 3

Here a new AOWebServiceProvider registers with the AOUDDI. However, contrasting this from Scenario 2 above, the requester consumes its providers through an AOComposite object because each provider on its own does not meet all the requirements of the requester (i.e. the requirements are not 100% satisfied by any single provider). If this new provider is better than the existing provider providing similar services in the AOComposite, then the AOConnector will request the AOComposite object to integrate with new provider. The AOConnector will also request the AOComposite to disconnect with the inferior and now redundant old AOWebServiceProvider connected to it.

The sequence of events that should happen is listed as follows:-

1. A new AO web service provider registers with the AOUDDI (RegisterNewAOWS).
2. The AOUDDI notifies the AOconnector about the new provider that has become available (NotifyConnectorAboutNewAOWS).
3. Notify users about new provider (NotifyRequesterAboutNewAOWS).
4. Select old web service that needs to be disconnected - old one is replaced with new one (SelectAOWSToDisconnect).
5. Get AOComposite (GetAOComposite).
6. Send AOWSDLs to AOComposite so it connects and disconnect appropriate web service (SendAOWSDLToAOComposite).
7. AOComposite will connect to new web service. (IndirectConnectionToNewAOWS).
8. Finally AOComposite will disconnect from inferior and redundant web

service. (DisconnectIndirectConnection).

This sequence of events is simulated in Alloy through the following assertion.

```
assert TestIndirectConnectionToNewAOWS {
   all aowsProvider : AOWebServiceProvider, myAOUDDI : AOUDDI, myAOConnector :
   AOConnector, aowsRequester : AOWebServiceRequester,
   myAOWSDL : AOWSDL, myAOComposite : AOComposite, myAOUDDI' : AOUDDI,
   myAOComposite' : AOComposite |
   {
                RegisterNewAOWS( myAOUDDI, myAOUDDI', aowsProvider )
                NotifyAOConnectorAboutNewAOWS ( myAOUDDI', myAOConnector )
                NotifyRequesterAboutNewAOWS ( myAOConnector, aowsRequester )
                SelectAOWSToDisconnect ( myAOWSDL , myAOConnector )
                InitializeAOComposite( myAOComposite)
                GetAOComposite ( myAOConnector, myAOComposite )
                SendAOWSDLToAOComposite ( myAOConnector, myAOComposite )
                IndirectConnectionToNewAOWS( myAOComposite, myAOComposite' )
                DisconnectIndirectConnection ( myAOComposite, myAOComposite' )
   }
} check TestIndirectConnectionToNewAOWS for 2 but 0 Request, 0 Result
```

The outcome was successful as no counter examples were found for this assertion.

9.5.4 Scenario 4

In this scenario, a new request is made by the AOWebServiceRequester to the AOUDDI through the AOConnector to search for an AOWebServiceProvider providing particular services. The AOUDDI finds appropriate AOWebServiceProviders that satisfy these criteria either partially or completely. The AOUDDI, transmits the results containing the list of providers together with

their percentage of matches (of services) back to the AOConnector. The connector will select the best provider from the list of providers returned by the AOUDDI, i.e. one that could meet 100% of the set of requirements of the requester. The connector will then be simulated to directly connect to the selected AOWebServiceProvider that meets all its requirements (without the need to construct and consume an intermediate composite object composed of multiple service providers as in the case of indirect connections).

The sequence of events that should happen is summarised as follows:-

1. The requester creates a new request (through the CreateRequest method).

2. The requester sends the request to discover the web service provider to the AOConnector (SendRequestToAOConnector).

3. The AOConnector will then send the request to the AOUDDI (SendRequestToAOUDDI).

4. The AOUDDI will process and execute the request and transmit the results of discovered service providers together with their percentage of matches back to the AO connector (ComputeResultAndTransmit).

5. The AO connector will select best AO web service provider based on a 100% percentage match criteria (SelectBestAOWS).

6. Finally, a direct connection from the AOConnector to the AOWS provider is created (DirectConnectionToRequestedAOWS), i.e. a connection without the aid of an intermediate composite object composed of service providers because a single service provider can satisfy all the requirements requested for by the requester.

This sequence of events described above is simulated in Alloy through the

following assertion.

```
assert TestDirectConnectionToRequestedAOWS {
    all myRequest : Request, aowsRequester : AOWebServiceRequester, myAOConnector :
    AOConnector, myAOUDDI : AOUDDI,
    myResult : Result, myAOWSDL : AOWSDL, myAOUDDI' : AOUDDI, myAOConnector' :
AOConnector |
    {
                CreateRequest ( myRequest, aowsRequester )
                SendRequestToAOConnector ( aowsRequester, myAOConnector )
                SendRequestToAOUDDI ( myAOUDDI, myAOUDDI', myAOConnector )
                ComputeResultAndTransmit ( myResult, myAOUDDI, myAOConnector )
                SelectBestAOWS ( myAOConnector, myAOWSDL )
                DirectConnectionToRequestedAOWS( myAOConnector, myAOConnector'
)
    }
} check TestDirectConnectionToRequestedAOWS for 2
```

The outcome was successful as no counter examples were found for this assertion.

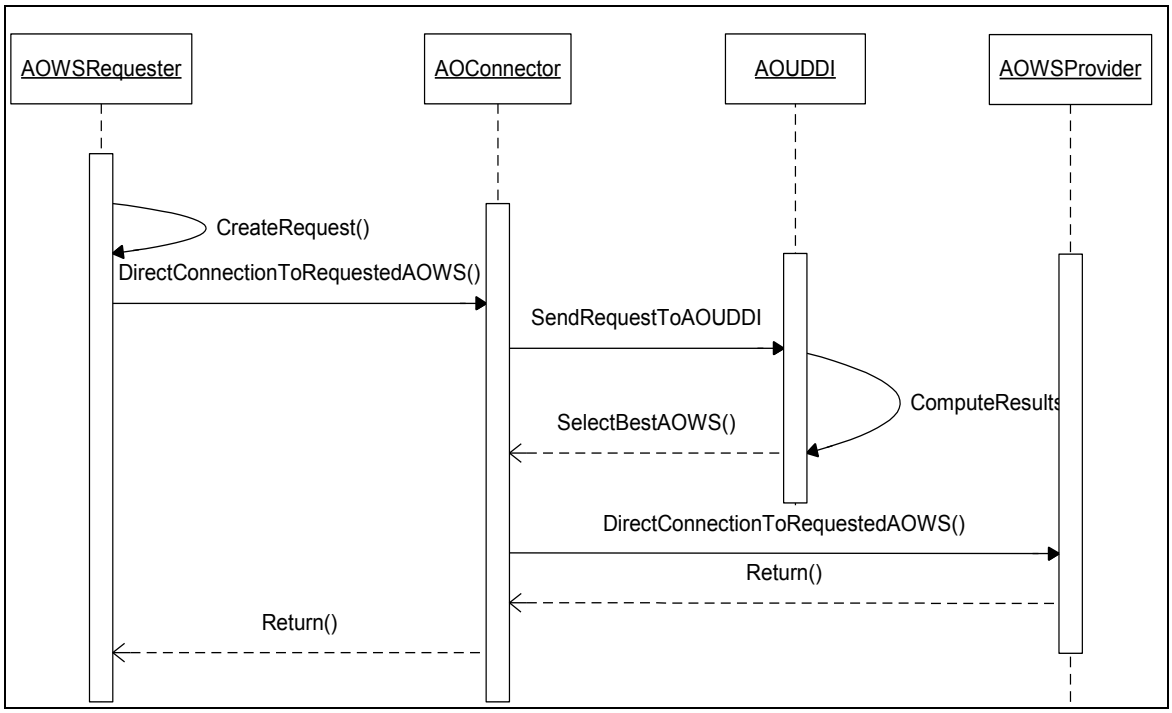


Figure 9.8. Sequence diagram depicting the dynamic service discovery via an AOConnector that was simulated using Alloy assertions.

Figure 8 above shows a sequence diagram of this dynamic service discovery process via an AOConnector simulated formally using Alloy. It shows dynamic discovery of the best matched web service provider selected by the connector based on the aspect-enriched request to the AOUDDI. This simulated assertion proved successful as no counter examples were found.

9.5.5 Scenario 5

In this simulation, a new request is made by the AOWebServiceRequester but the AOConnector fails to find any AOWebServiceProvider that meets 100% of its requirements. So the AOConnector will use the AOComposite that will in turn lookup and connect to a set of AOWebServiceProviders that collectively will be able to provide the functionality required by the requester.

The sequence of events that should happen is listed as follows:-

1. The requester creates a new request (CreateRequest)
2. The requester sends request to the AOConnector (SendRequestToAOConnector)
3. The AO connector will send the request to the AOUDDI (SendRequestToAOUDDI)
4. The AOUDDI will execute the request and transmit the result to the AO connector (ComputeResultAndTransmit)
5. The AO connector will select best AO web service (SelectBestAOWS)
6. Get AOComposite (GetAOComposite)
7. Transmit the AOWSDLs to the AOComposite (TransmitAOWSDLToAOComposite)

8. Finally create an indirect connection to the AO web service providers through the AOComposite (IndirectConnectionToRequestedAOWS)

This sequence of events is simulated in Alloy through the following assertion.

```
assert TestIndirectConnectionToRequestedAOWS {
   all myRequest : Request, aowsRequester : AOWebServiceRequester, myAOConnector :
   AOConnector, myAOUDDI : AOUDDI,
   myResult : Result, myAOWSDL : AOWSDL, myAOComposite : AOComposite,
   myAOComposite' : AOComposite, myAOUDDI' : AOUDDI |
   {
                  CreateRequest ( myRequest, aowsRequester )
                  SendRequestToAOConnector ( aowsRequester, myAOConnector )
                  SendRequestToAOUDDI ( myAOUDDI, myAOUDDI', myAOConnector )
                  ComputeResultAndTransmit ( myResult, myAOUDDI, myAOConnector )
                  SelectBestAOWS ( myAOConnector, myAOWSDL )
                  GetAOComposite( myAOConnector, myAOComposite )
                  InitializeAOComposite( myAOComposite)
                  TransmitAOWSDLToAOComposite ( myAOConnector, myAOComposite )
                  IndirectConnectionToRequestedAOWS ( myAOComposite,
   myAOComposite' )
   }
} check TestIndirectConnectionToRequestedAOWS for 2
```

The outcome was successful as no counter examples were found for this assertion.

9.5.6 Scenario 6

Here we simulate the setting up of specific aspects and their details for service providers that can be dynamically discovered and used by AOWebServiceRequesters based on the AOWSDL document of the provider concerned. The aspects based on its aspect details and other information cater for the searching of hotel rooms from the actual Travel Planner prototype involving AO web services.

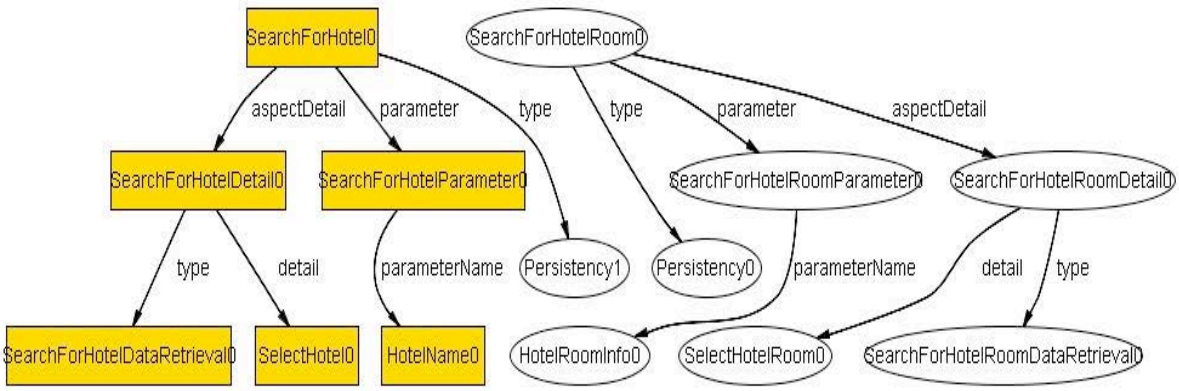


Figure 9.9: Alloy model for the relationship between the various aspects and aspect details involved in searching for a hotel room.

Figure 9.9 above shows the Alloy model that depicts the relationship between the various aspect and aspect details involved in the simulation of its key functions, namely searching for a hotel and a particular room in the hotel.

The sequence of events that should happen in defining and describing aspects is listed as follows and this includes describing, defining and initializing the aspect details and all other information related to the aspect (i.e. its standalone,

aowsentrypoint characteristics etc. that can be stored and retrieved from AOWSDLs)

1. First initialize the "standalone" and "AOWSEntrypoint" elements for SearchForHotel and SearchForHotelRoom aspects (they are initialized to true by default).

2. Set type for the aspect - in this case, both aspects are having the type of persistency.

3. Inside aspect detail, there is attribute called detail. SetAspectDetail is setting the detail for aspect detail.

4. Set AOWSEntryPoint for SearchForHotel. AOWSEntryPoint should be true since it doesn't need any other aspect.

5. Set Standalone to true for both aspects.

6. Set Provided to true for both aspects.

7. Set SearchForHotelRoom as User operation of SearchForHotel

8. Set Return type for both aspects.

9. Set ParameterName

10. Finally set the ParameterType

This sequence of events is simulated in Alloy through the following assertion.

```
assert Setting {  all hotel : SearchForHotel, hotelRoom : SearchForHotelRoom, true : True, false :
False, persistency : Persistency, dataSet : DataSet,
selectHotel : SelectHotel,  selectHotelRoom :  SelectHotelRoom, hotelData :
SearchForHotelDataRetrieval, hotelRoomData : SearchForHotelRoomDataRetrieval, string : String,

hotelName : HotelName, hotelRoomInfo : HotelRoomInfo |
                {
                                Initialize(hotel, false)
                                Initialize(hotelRoom, false)
                                SetType(hotel, persistency)
```

```
                SetType(hotelRoom, persistency)

                SetAspectDetail(hotel, selectHotel)
                SetAspectDetail(hotelRoom, selectHotelRoom)

                SetAOWSEntryPoint(hotel, true)

            SetStandalone(hotel, true)

                SetAspectDetailType(hotel, hotelData)
                SetAspectDetailType(hotelRoom, hotelRoomData)

                SetProvided(hotelRoom, true)
                SetProvided(hotel, true)

                SetUserOperation( hotel, hotelRoom)

                SetReturnType( hotel, dataSet )
                SetReturnType( hotelRoom, dataSet )

                SetParameterName(hotel, hotelName)
                SetParameterName(hotelRoom, hotelRoomInfo)

                SetParameterType(hotel, string)
                SetParameterType(hotelRoom, string)
        }
} check Setting for 3
```

The outcome was successful as no counter examples were found for this assertion.

As no counter-examples were found by the Analyzer in all its check runs for all the scenarios mentioned above that we tested, we believe that our AOWS approach and its abstractions are formally feasible and logically correct (Jackson 02).

9.6 Summary.

We have successfully modeled the AOWS system using the Alloy modeling language and further analyzed and verified that this model and its abstractions are actually correct and feasible. In order to analyze and verify the web service system, we had to create multiple meaningful signatures, predicates, facts and assertions and used the Alloy Analyzer, since this analyzer has the inherent ability to check whether a given assertion is valid or not based on the facts presented. When we checked our assertions with the Alloy Analyzer, we found them all to be valid and no counter examples were found, verifying that our novel AOWS system and its abstractions are logically and mathematically correct.

In summing up we modeled, analyzed and verified the following for the AOWS system:

- that based on the AOWS architecture and its Alloy constructs considered in this chapter, the AOWebServiceRequester can successfully be simulated to connect and make requests to the AOConnector through a series of well defined and meaningful logical steps. The connector will process the request or relay it to the relevant sub-systems connected to it if required.
- that the AOConnector can successfully connect to the AOUDDI and make requests to it to locate the web services which contain the required services, aspects and functions.
- that the AOUDDI will be able to process the request made by the AOConnector and search through the AOUDDI data base to obtain a list of required web services together with their locations. The information will then be delivered back to the AOConnector.

- that when the AOConnector receives the list of AOWebServiceProviders it can accurately determine the most suitable AOWebServiceProvider to connect to.
- that when the AOConnector cannot find a single suitable provider to satisfy the needs of the requester but requires more than one AOWebServiceProvider to carry out the tasks, the connector can successfully link to the AOComposite to construct an aspect-oriented composite object of these services. The AOComposite is able to connect to each of those AOWebServiceProviders and allow the AOConnector to have indirect connection to them through the AOComposite.
- that when any new AOWebServiceProvider comes into existence, it will be able to successfully connect to the AOUDDI to register itself and publish the provider's well defined and structured AOWSDL using the registry.
- that the AOUDDI will hold a set of all the AOConnectors that has made requests in the past and it can successfully notify all of those AOConnectors when a new AOWebServiceProvider has been registered.
- That when the AOConnector receives a notification regarding a new AOWebServiceProvider that better suits its needs, it can successfully make direct connection to the new provider. The AOConnector will also be able to determine when the AOWebServiceRequester has finished with its most recent request and safely disconnect from the old AOWebServiceProvider and redirect the link to the newly discovered AOWebServiceProvider.
- that when aspects need to be fully initialized for dynamic discovery, this can be successfully done by extracting all the information from the relevant AOWSDL document or web service provider. The aspect and its details depend on a variety of parameters including its function, return type, standalone, AOWSEntryPoint elements

etc. and all this can be successfully simulated to setup the aspect correctly for discovery.

Therefore we can conclude that, based on the positive results of the simulations and validations, the model of our AOWS system and its abstractions that were captured using Alloy is formally and logically correct and that the designs and implementation of AOWS are also correspondingly correct and feasible.

10 Tool Support for Aspect-oriented Web Services

The efficient development of Aspect-Oriented Web Service based applications using AOCE techniques necessitates the development of novel and effective software tools and environments to support its analysis, design and development phases. We also need tools to generate code for the aspect-oriented components/subsystems of AOWS and the AOWSDL documents of the aspect-oriented service providers. In this chapter, we discuss a tool, called the AOWSCreator, which we developed specifically for these purposes in this thesis. We refactored and extended an existing meta-modelling tool, Pounamu (Zhu et al 04) to achieve this.

10.1 Overview of Pounamu

Pounamu is a meta-modelling CASE tool that has been developed in the University of Auckland and it can support development using Visual Languages. It is a meta tool for specifications and the generation of multiple view visual tools. The tool permits specification of visual notational elements, tool information models, visual editors, the relationship between notational and model elements, and behaviour (Zhu et al 04). It is a top-level CASE tool, meaning that it can be used to develop and customise a variety of other CASE tools from it based on user's specific needs. Changes to the meta tool specification are immediately reflected in tool instances. The CASE tools created can be used to create models with multi views. However in its existing form, Pounamu could not be used to carry out design and development for projects using the AOCE development methodology. As such, in this thesis we refactored, extended and customised Pounamu to enable the design and development of AOWS based systems using AOCE, including code and AOWSDL generation using this tool.

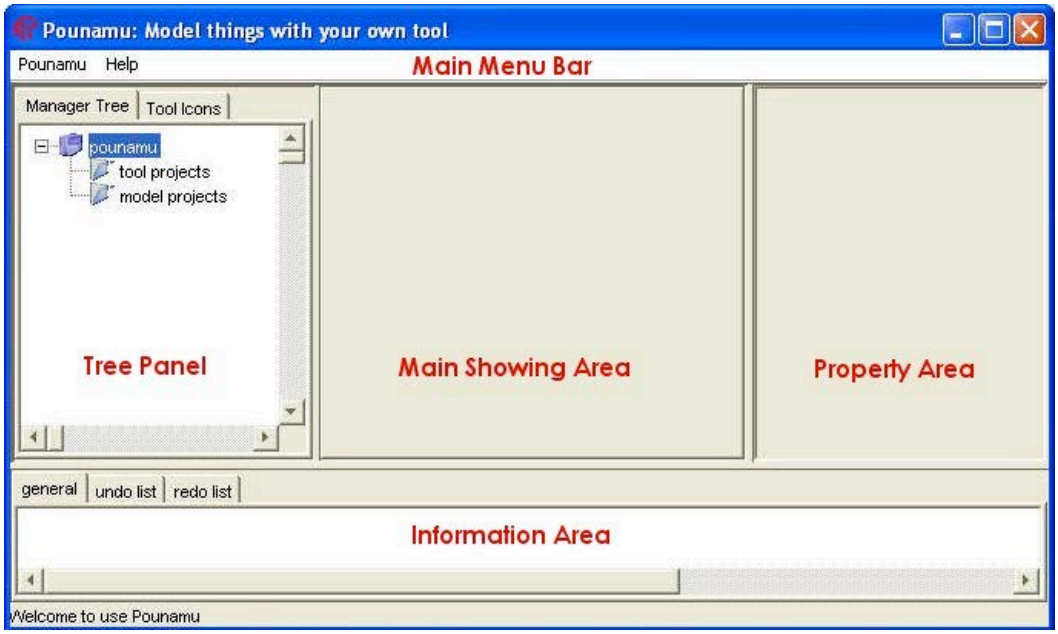


Figure 10.1: The user interface of the Pounamu meta-modelling tool

Figure 10.1 above shows Pounamu's interface which is divided into five main areas, i.e. the Tree Panel, Main Showing Area, Property Area, Information Area and Main Menu Bar. Visual modelling objects (shapes, entities, connectors, associations etc.) are drawn in the Main Showing Area and they are also listed in the tree-structure of the Tree Panel shown on the left of Pounamu. The Property Area contains all the properties associated with the respective objects drawn and allows us to manipulate and change the behaviour and properties associated with the object drawn. The Information Area allows users to review the actions that they have performed using the tool. It records the history of all actions performed during the current session of using the tool.

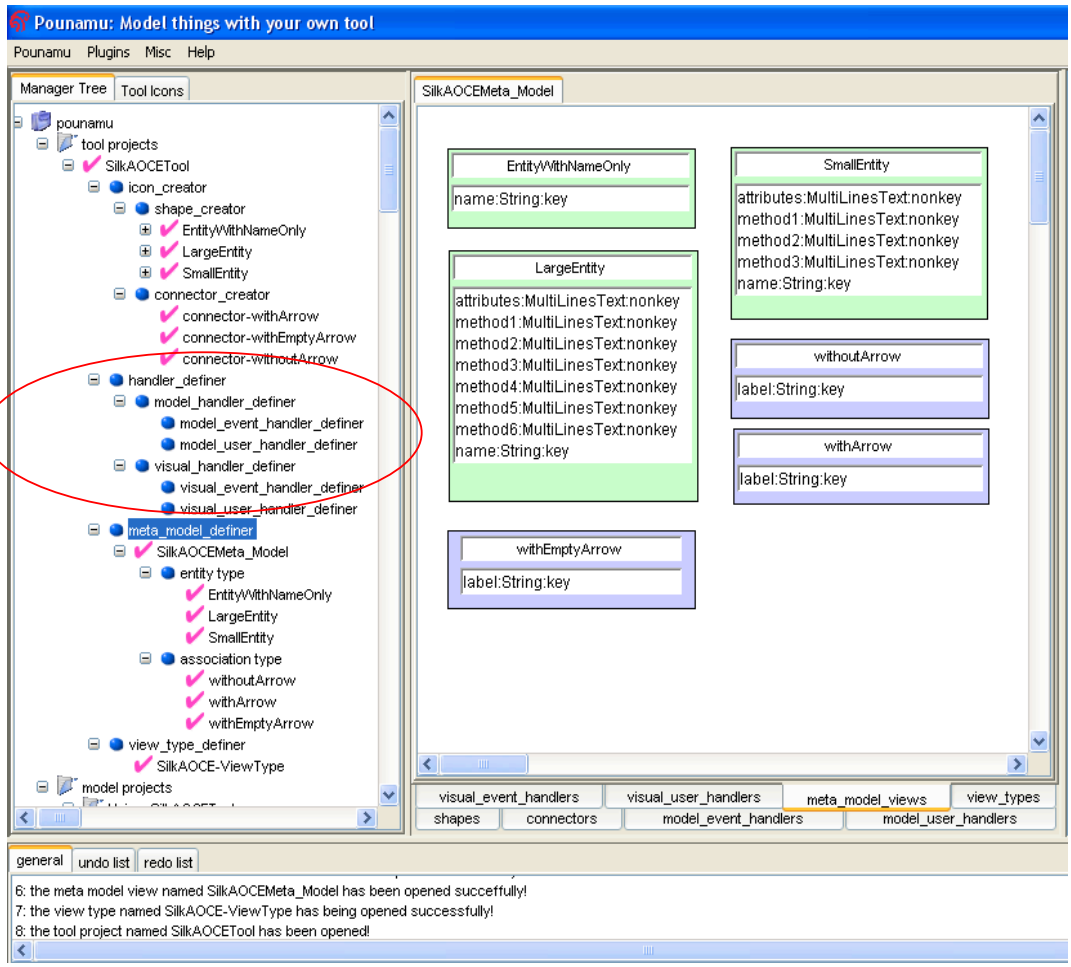


Figure 10.2 The working environment of Pounamu depicting its main components

Figure 10.2 above shows the four main components (depicted as the main nodes of the tree structure in the leftmost panel) of Pounamu, namely the icon creator, handler definer, model definer and view type definer. To develop a visual language CASE tool in Pounamu, users have to first create and classify the entities and associations required for the visual language. These entities and associations are then specified in the meta-model definer. Entities are objects like aspects, classes, components etc. that are defined in the visual languages and associations used to describe the relationships between entities. All the entities and associations are designed using an icon creator. Inherent in Pounamu are a number of elementary shapes that may be used to create

the shapes according to the user's specifications. The Handler Definer (shown circled in red) is a very important mechanism and is used to add extra services or events to the tool. We have used this mechanism extensively to create our AOWSCreator tool. Once the meta-models and their properties are determined, they are mapped in pairs using the view type definer together with the handlers. Handlers are pluggable programs that have to be written into Pounamu to support additional features and functions that were not built into Pounamu.

10.2 Depicting and Manipulating Aspects in Pounamu

We refactored the Pounamu tool and also used handlers to enable us to depict and manipulate aspects in our AOWSCreator tool as describe below.

10.2.1 Depicting aspects visually in designs

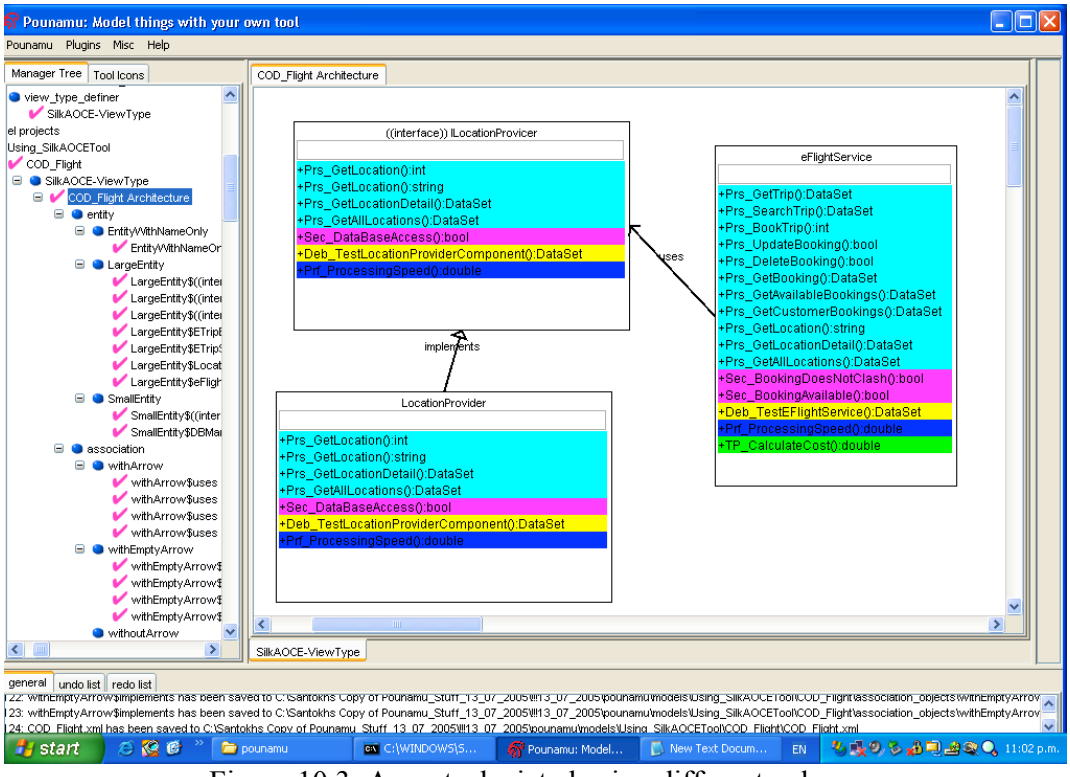


Figure 10.3: Aspects depicted using different colours

Figure 10.3 above shows the aspects within components that are depicted using a variety of colours. Aspects of the same type are of the same colour, while different aspect types are depicted in different colours. For instance, all the aspects of type persistency are depicted using cyan while security aspects are in pink. Using colours to depict aspect types make it very easy for us to identify them and observe and understand how they cross-cut the components and software.

We can also use icons and patterns to depict the aspects besides using colours. The shape of the icons used depicts whether it is provided or required by the component. A square box indicates that the aspect requires the crosscutting information while a diamond shaped one provides it for another component. During implementation the aspects can be sieved out so that we can identify and isolate these cross-cutting modular units in the objects. Abbreviations of the aspects can also be inserted into the design diagrams so that programmers can immediately know what the colours represent. In the figure shown above, the abbreviations Prs, Sec, Deb, TP and Prf represent the aspect types Persistency, Security, Debugging, Transaction Processing and Performance respectively. The positive or negative signs preceding the aspect types denote whether the aspects are provided or required by the component.

10.2.2 Collapsing and Expanding Views

AOWS systems can be very large, complex and made up of numerous classes, components and subsystems, with complex and complicated inter-relationships between them. As such, it is to be expected that the designs of these systems can, accordingly, be equally large and complex. Therefore, we need mechanisms to collapse (hide) parts of the designs that we are not interested in or do not want to

consider at the moment. By collapsing these views we will be able to focus and concentrate on the remaining relevant parts and get a better picture of that section.

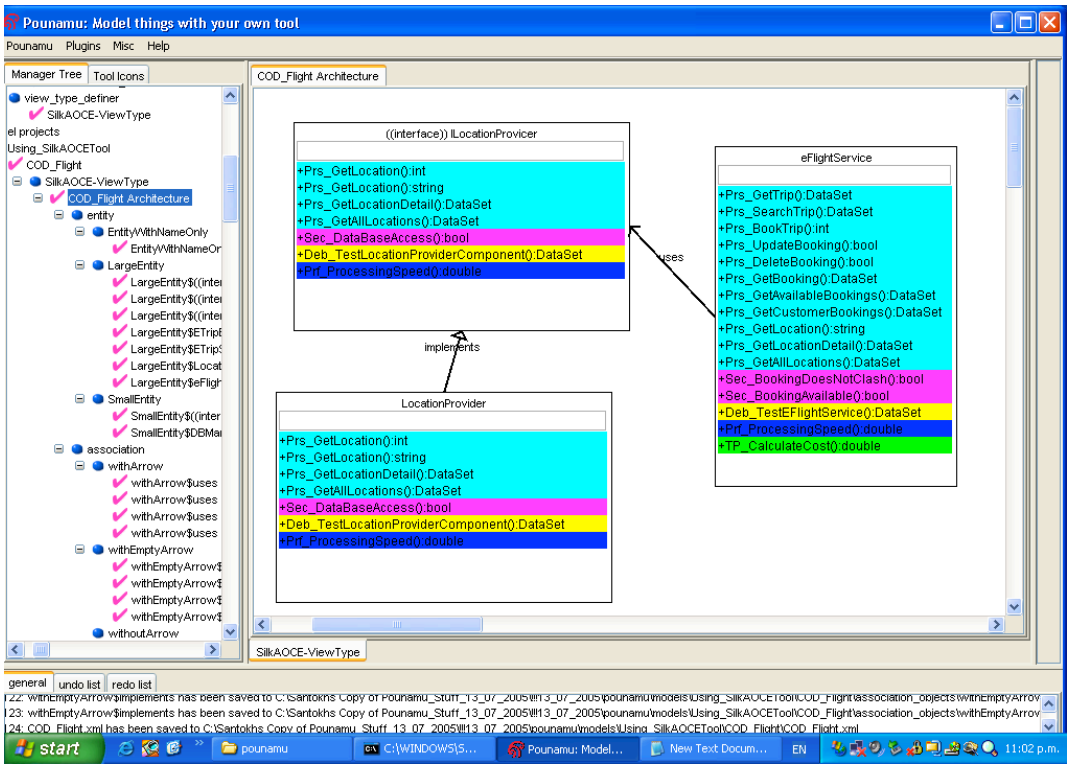


Figure 10.4(a): Design diagram before collapsing class

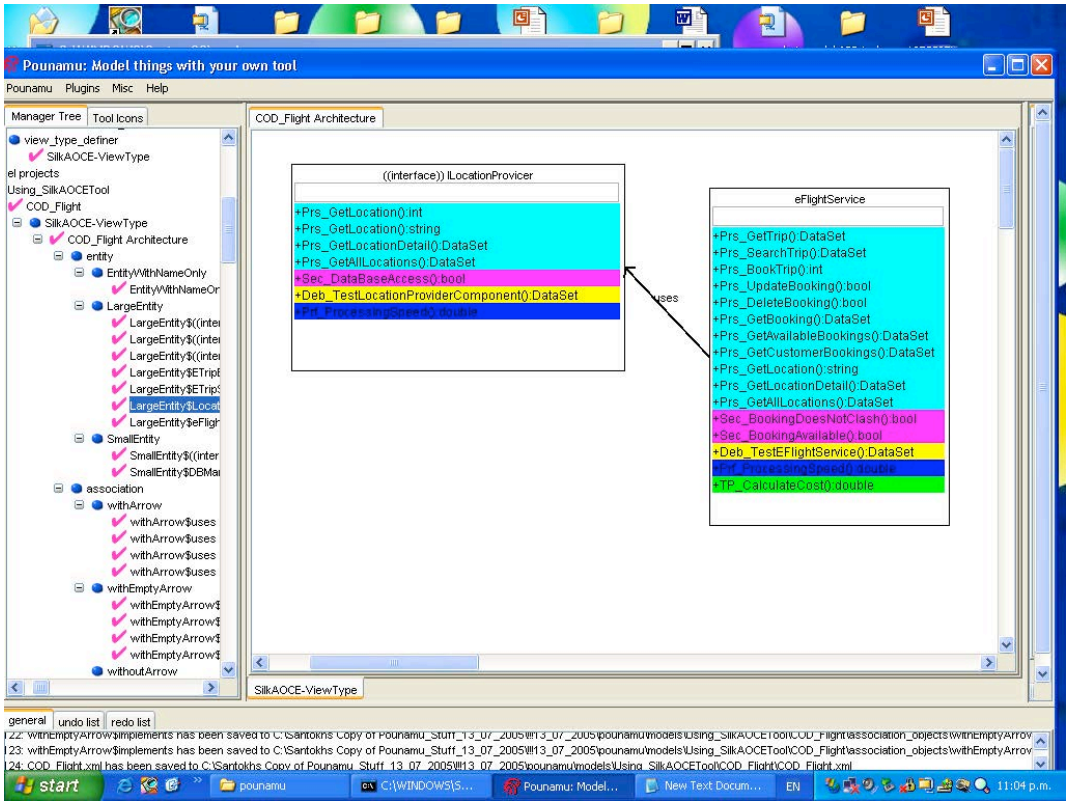


Figure 10.4(b): Design diagram after collapsing derived class

Figure 10.4 above shows a simple diagram depicting the views before and after using the collapsing mechanism in our tool. In large and complex designs, views are collapsed to get a better idea of the remaining parts and also we'll be able to concentrate better on the sections that may be of interest to us by cutting out clutter. All views that were collapsed can be expanded out again to get the whole picture of the design diagrams.

10.2.3 Collapsing aspect types within components/subsystems

There are usually a large number of aspects within complex programs that cross-cut the various components and classes involved. These aspects themselves may be composed of different types, e.g. Persistency, Security, Performance, Transaction

Processing etc. Besides having the capability of hiding design entities like components, classes and interfaces, the aspect types within the remaining visible entities can also be collapsed (hidden) using the AOWSCreator tool so that we can concentrate on the remaining type(s) of aspects that are of interest to us. By hiding the aspects we can concentrate on the remaining aspects that we want to deal with more clearly and edit/refactor accordingly. In this way we can isolate and concentrate on any type(s) of aspects that we are considering, without irrelevant aspects cluttering and obscuring the functions. The hidden aspects can easily be expanded out again (i.e. made visible) to get the whole view of the aspect-oriented designs.

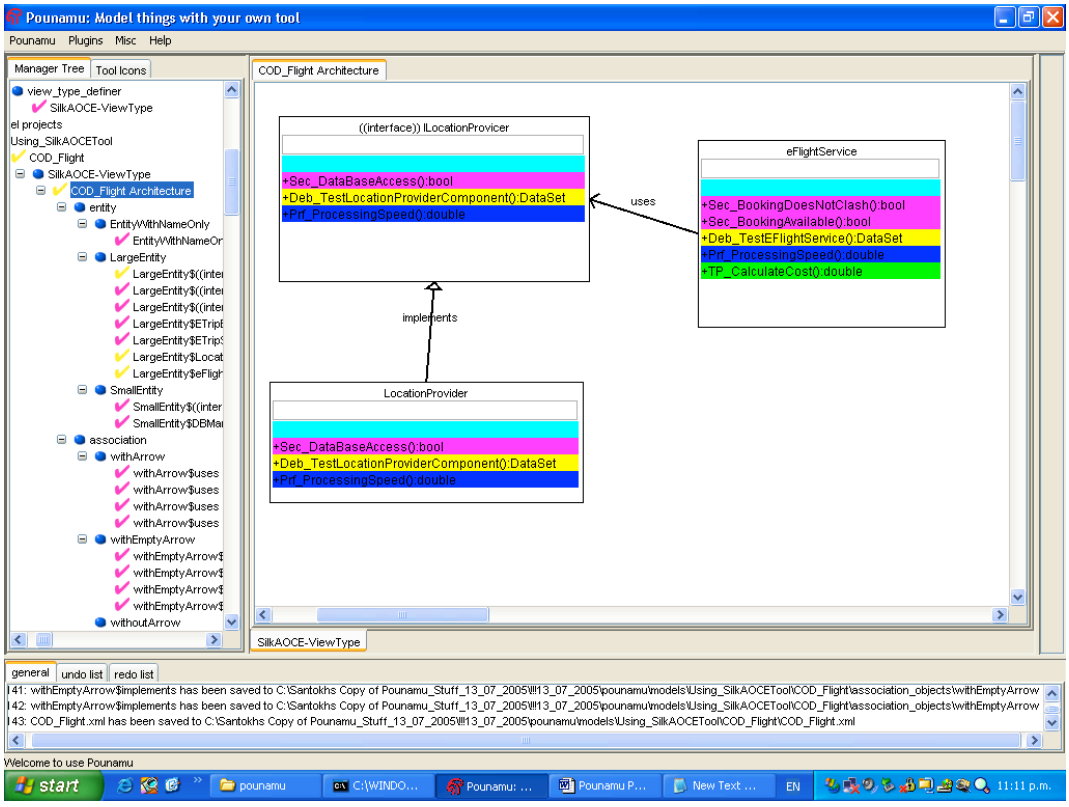


Figure 10.5: View showing persistency aspects collapsed in design diagrams

The earlier figure 10.4(a) depicted the various aspects in a component, each type having its own distinct colour, for instance persistency aspects coloured cyan, security

aspects pink, performance aspects blue etc. But if we are not interested in the aspects that deal directly with data retrieval, editing, storage etc., i.e. of the type persistency, then as shown in Figure 10.5 above, we can collapse (hide) all the persistency aspects and just concentrate on the behaviour of the other aspects. We can also hide any number of aspect types that we wish. For instance we may hide all the other aspects and just view the persistency aspects so that we can understand and modify the issues and concepts connected with data persistency e.g. data retrieval, editing, storage etc. in the software system that is being designed.

10.2.4 Inserting aspect details and code

The amount of information that can be depicted in the design diagrams is limited because if we introduce too much information in them, the diagrams become too large and difficult to comprehend. As such we introduced pop-up frames to contain all the detailed information about the aspects including any code snippets that may be helpful for the implementation stages.

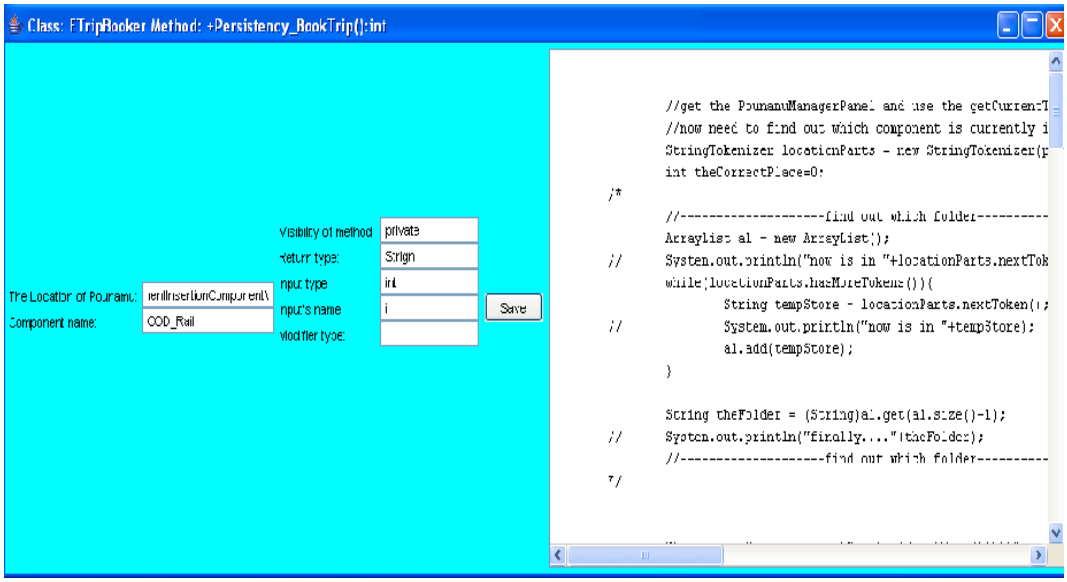


Figure 10.6: Pop-up frame for the details about the aspects, including code or pseudo code insertion

Figure 10.5 above shows an example of the pop-up frame that can be used to store the details about the aspects. The code (or pseudo code) is inserted into the right hand text-area of the frame and can later be generated into the respective classes. The interface is a very comprehensive yet simple enough to understand and use.

10.3 Code Generation in Pounamu

We extended the Pounamu meta-modelling tool so that skeleton C# code for the AOWS application can be generated directly from the design diagrams that are drawn using the AOWSCreator. To generate more comprehensive code, we can insert the operations supported and their details in property boxes of pop-up frames for each and every component. AOWSDL documents written in XML can also be generated on the fly for the AOWS providers based on the APIs that we want to expose for the providers concerned.

Each software component, by its definition, is also associated with an interface that describes its exposed functionalities. These interface acts as a contract with the other components interacting with it. We can even insert code including full business logic of that particular class or aspects into the pop-up property boxes. This additional code will appear in the respective areas of the files containing the classes or aspects concerned when we generate the AOWS code using our tool.

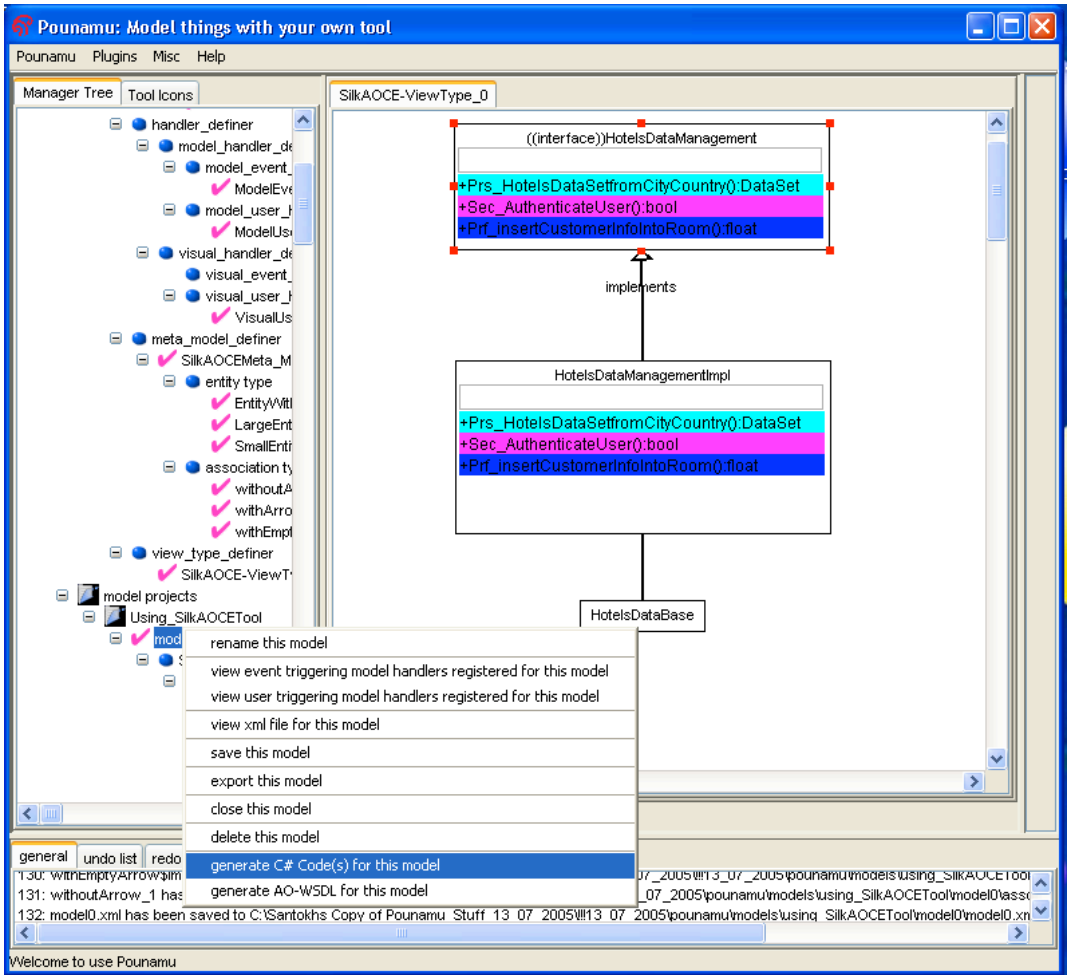


Figure 10.7: Generating C# code using the AOWSCreator

Figure 10.7 above shows the drop down menu list that can be used to generate Visual C# code from the aspect-oriented design diagrams. To generate the code we need to draw the design diagrams and insert any aspectual properties, code snippets etc. into the pop-up property boxes first. Then right click on the model in the tree view and select “Generate C# code for this model”. Shown below in figures 10.8 and 10.9 are some snippets of C# code that was generated using the AOWSCreator. The tool expands out the full name of the aspect type in the generated C# code if we use abbreviations for common aspect types, e.g. Prs (Persistence), Sec (Security), Prf (Performance), TP (Transaction Processing) etc. In future we will extend the tool to give an option to generate skeleton code in other popular languages like Java, Visual

Basic, C++ etc. so that it can be more useful to the wider community of software engineers who may be more comfortable writing code in a language of their own preference.

```
using System;
using System.Collections;
using System.Data;

namespace HotelsDataManagementComponent
{
    /// <summary>
    /// Author:
    /// Date:
    /// Summary description for HotelsDataManagementComponent.
    /// </summary>
    public interface HotelsDataManagement
    {
            DataSet Persistence_HotelsDataSetfromCityCountry(string
strCity, string strCountry);
            bool Security_AuthenticateUser(string strLogin, string
strPassword);
            float Performance_insertCustomerInfoIntoRoom(int numOfInserts);

    }
}
```

Figure10.8: 'HotelsDataManagement' interface generated from its design

```
using System;
using System.Collections;
using System.Data;


namespace HotelsDataManagementComponent
{
    /// <summary>
    /// Author:
    /// Date:
    /// Summary description for HotelsDataManagementComponent.
    /// Code can be added manually or through the pop-up design frames
    /// in the AOWSCreator tool.
    /// </summary>
    public class HotelsDataManagementImpl:HotelsDataManagement
    {
            public HotelsDataManagementImpl()
            {
                    //
                    // Add constructor logic here
                    //
            }
```

```
            public DataSet Persistence_HotelsDataSetfromCityCountry(string
strCity, string strCountry)
            {
                  //
                  // Add business logic here
                  //
                  return null;
            }

            public bool Security_AuthenticateUser(string strLogin, string
strPassword)
            {
                  //
                  // Add business logic here
                  //
                  return false;
            }

            public float Performance_insertCustomerInfoIntoRoom(int
numOfInserts)
            {
                  //
                  // Add business logic here
                  //
                  return 0;
            }


      }
  }
```

Figure10.9: The 'HotelsDataManagementImpl' class type code that was generated

10.4 AOWSDL generation using Pounamu

Each AOWS provider has an AOWSDL associated with it that can be published in the AOUDDI. These AOWSDLs can be discovered and integrated with requesters that require the respective services.

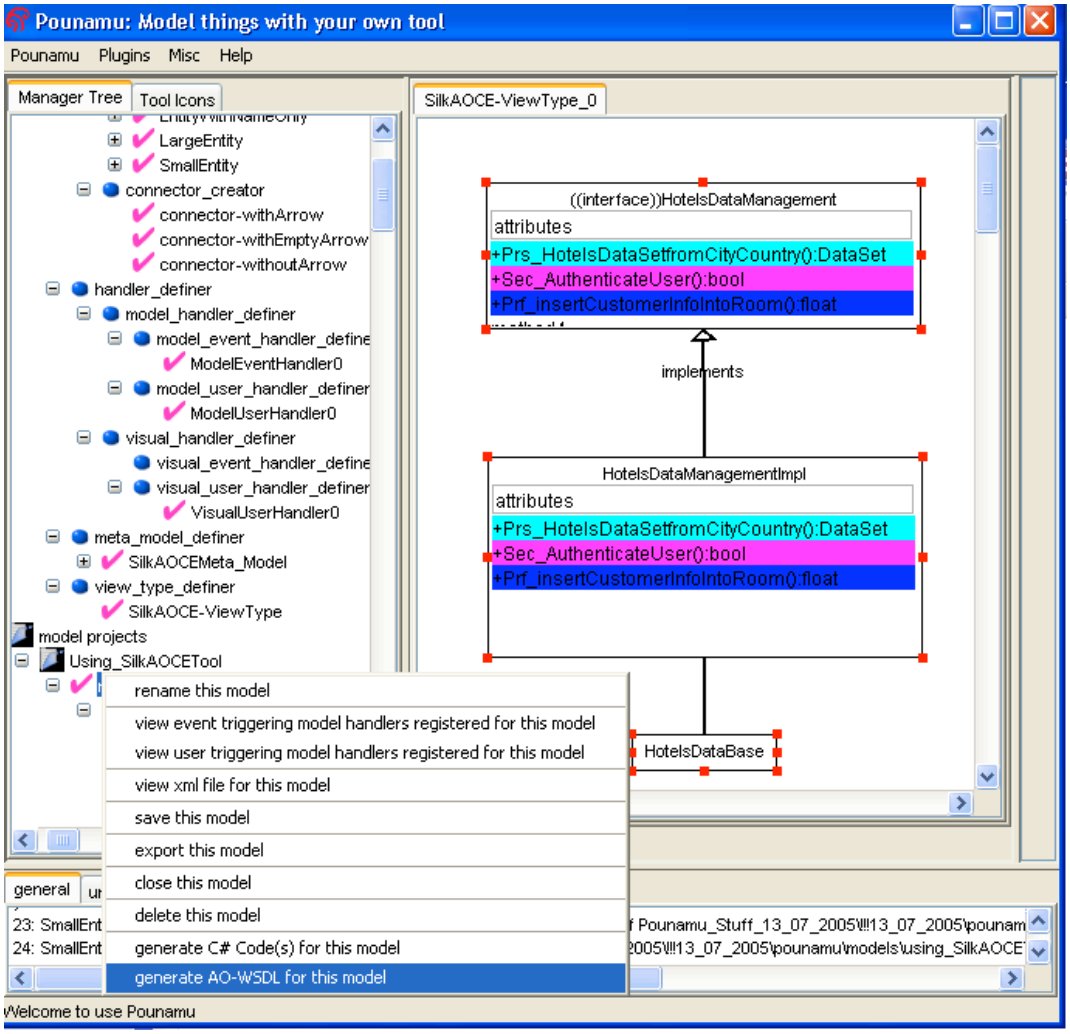


Figure 10.10: Generating AOWSDL from the design model drawn.

Figure 10.10 above shows how the AOWSCreator can be used to generate AOWSDL from the design models drawn in the tool. To generate the AOWSDL, similar steps as that for the generation of C# skeleton code need to be executed, i.e. we need to draw the design diagrams first. Then right click on the aspect-oriented provider in the tree view shown in the left hand panel in the figure and select “Generate AOWSDL for this model”. The AOWSDL generated will be saved to an XML file and saved in a folder called the aowsdl folder of the model. A sample of a snippet from the AOWSDL generated using the AOWSCreator tool is shown in the following pages.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://tempuri.org/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:aowsdl="http://localhost/AOUDDIWebService/bin/aowsdlSchema.xml"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://tempuri.org/" xmlns="http://schemas.xmlsoap.org/wsdl/">
      <types>
            <s:schema elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
                  <s:import namespace="http://www.w3.org/2001/XMLSchema" />
      </types>


     <aowsdl:AOComponents Name="HotelsWebServiceComponents">
            <aowsdl:AODocumentation Information="Exposes aspects to find vacant rooms in
hotels, searches for hotels  based on city or country of interest. After finding rooms
reservations can be made to book the rooms concerned...  All human readable information go
here. This include instructions and high level documentation about the web service for human
consumption." />
            <aowsdl:WSDescription Description="To find, update, delete and insert reservations
or bookings for vacant hotel rooms" />
                  <aowsdl:Component ComponentName="HotelsDataManagementComponent">
                  <aowsdl:ComponentDescription Description="Component to find hotels in various
cities and countries including rooms availability" />
                        <aowsdl:Aspects>
                              <aowsdl:FunctionalAspects>
                                    <aowsdl:Aspect Type="Persistency"
AspectName="HotelsDataSetfromCityCountry" WSEntryPoint="true" Standalone="true">
                                          <aowsdl:AspectDescription Description="To search for hotels based on
city or country query" />
                                             <aowsdl:Parameters>
                                                <aowsdl:Parameter ParameterName="strCity"
ParameterType="string" />
                                                <aowsdl:Parameter ParameterName="strCountry"
ParameterType="string" />
                                             </aowsdl:Parameters>
                                                <aowsdl:Return ReturnType="DataSet" />
                                                <aowsdl:AspectDetail Type="data retrieval" Detail="select"
Provided="true" />
                                                <aowsdl:AspectUserOperations
UsedBy="Persistence_HotelFinder|TransactionProcessing_ItenaryManager" />
                                                <aowsdl:UsesOperations
Uses="Persistence_roomsByHotelID|Persistence_OnSiteFacilities|Persistence_OffSiteFaci
lities|Persistence_placesOfInterest" />
                                          </aowsdl:Aspect>
                                          <aowsdl:Aspect Type="Security" AspectName="AuthenticateUser"
WSEntryPoint="true" Standalone="true">
                                          <aowsdl:AspectDescription Description="To authenticate that it is a
correct password and login used" />
                                             <aowsdl:Parameters>
                                                <aowsdl:Parameter ParameterName="strLogin"
ParameterType="string" />
                                                <aowsdl:Parameter ParameterName="strPassword"
ParameterType="string" />
                                             </aowsdl:Parameters>
                                                <aowsdl:Return ReturnType="Boolean" />
                                                <aowsdl:AspectDetail Type="security access" Detail="access"
Provided="true" />
                                          <aowsdl:AspectUserOperations UsedBy="" />
                                          <aowsdl:UsesOperations Uses="" />
                                    </aowsdl:Aspect>

                              </aowsdl:FunctionalAspects>
                               <aowsdl:NonFunctionalAspects>
```

```
                    <aowsdl:Aspect Type="Performance"
AspectName="insertCustomerInfoIntoRoom">
                        <aowsdl:AspectDescription Description="The time taken to perform the
specified number of insert operations in ms" />
                        <aowsdl:Parameters>
                            <aowsdl:Parameter ParameterName="numOfInserts"
ParameterType="integer" />
                        </aowsdl:Parameters>
                        <aowsdl:Return ReturnType="float" Units="ms" />
                        <aowsdl:AspectDetail Type="performance speed in ms"
Detail="Speed" Value="230" ValueQualifier="lessthan" Provided="true" />
                    </aowsdl:Aspect>
                </aowsdl:NonFunctionalAspects>
            </aowsdl:Aspects>
        </aowsdl:Component>
    </aowsdl:AOComponents>
    <service name="HotelsWebService">
        <port name="HotelsWebServiceSoap" binding="s0:HotelsWebServiceSoap">
            <soap:address
location="http://localhost/WebServiceHRHotels/HotelsWebService.asmx" />
        </port>
        <port name="HotelsWebServiceHttpGet" binding="s0:HotelsWebServiceHttpGet">
            <http:address
location="http://localhost/WebServiceHRHotels/HotelsWebService.asmx" />
        </port>
        <port name="HotelsWebServiceHttpPost" binding="s0:HotelsWebServiceHttpPost">
            <http:address
location="http://localhost/WebServiceHRHotels/HotelsWebService.asmx" />
        </port>
    </service>
</definitions>
```

Figure 10.11: Snippet of the AOWSDL generated from the model defined

Figure 10.11 above shows an example of a snippet of the AOWSDL generated using the AOWSCreator tool. The parameters of the elements of the AOWSDL are obtained from the overall designs as well as from those stored through the pop-up frame containing the aspect details and other information related to the aspect-oriented components. The AOWSDL documents are well formed and can be interpreted by the aspect-oriented parsers in the AOUDDI and other subsystems in the AOWS.

10.5 Summary

We are able to successfully visually depict, construct and manipulate aspect-oriented components to design and develop AOWS software by using the AOWSCreator that we created. We extended and refactored Pounamu, a meta-CASE tool, to create the

AOWSCreator that can be used to support software development using Aspect-Oriented Component Engineering.

By introducing the collapsible views functions we are able to hide or show any entities and aspect types in our designs. These functions are very important because AOWS systems can be very large, complex and made up of numerous classes, components and subsystems, with complex and complicated inter-relationships between them. As such, the designs of these systems can, accordingly, be equally large and complex. Furthermore numerous different types of aspects may exist within the components and this may give rise to more clutter. The collapse mechanisms allows us to hide parts of the designs that we are not interested in or do not want to consider at the moment. By collapsing these views we will be able to focus and concentrate on the remaining relevant parts and get a better picture of the remaining section(s). The collapsed views can easily be expanded out again to get the whole view of the designs.

Using our AOWSCreator, different aspect types can be depicted in different colours so that we can locate and identify these cross-cutting concerns more easily in our designs. We can also generate aspect-oriented skeleton C# code for the AOWS and the AOWSDL documents for the respective aspect-oriented providers. More specific code (e.g. aspects code for business logic, database access snippets etc.) can be inserted through the pop-up frames in our tool so that they appear in the classes generated.

11 Experiences and Evaluation

In this thesis, Aspect-Oriented Component Engineering techniques were used extensively and exclusively to develop the whole collaborative Travel Planner system based on our novel Aspect-Oriented Web Services technology. In this chapter, we share our experience, observations and knowledge gained from using the AOCE methodology in the design and development of the system. We also describe the evaluation carried out on AOCE and the Travel Planner system and further analyse and discuss the results here.

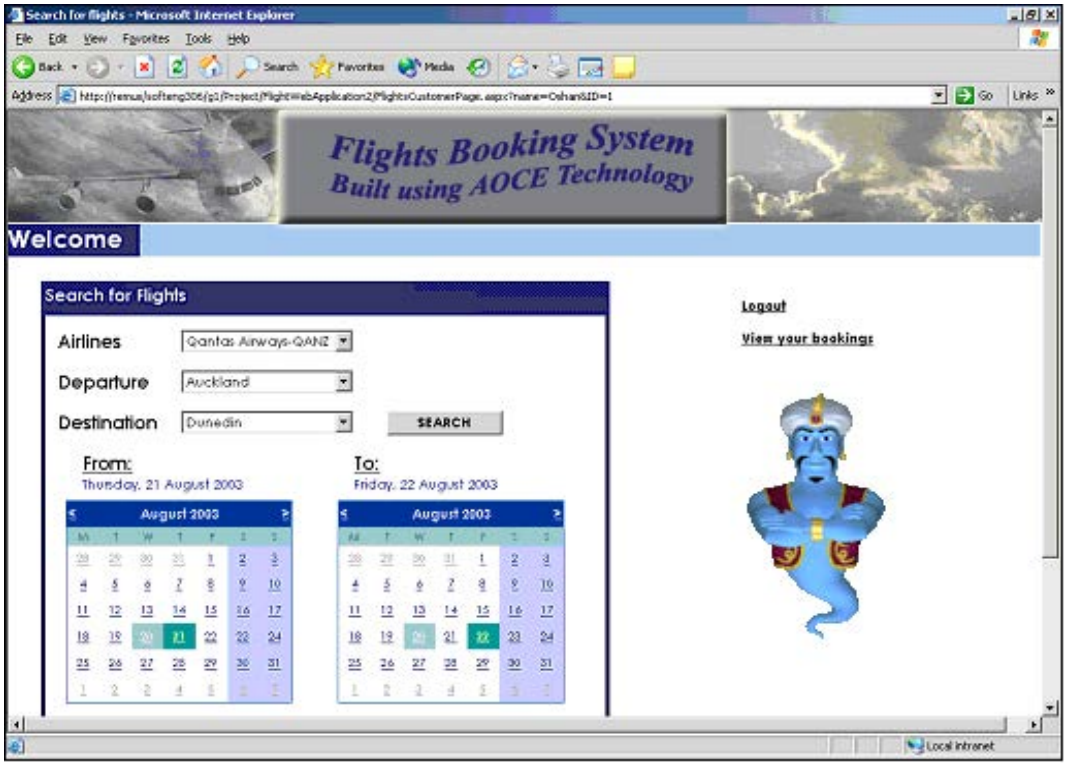


Figure 11.1: AO Travel Planner interface for Flights Booking services

An example of the thin client browser user interface that we implemented of the Aspect-Oriented Travel Planner of the client consuming the Flights Booking web services is shown in Figure 11.1 above. Visual Interactive Agents were utilised in our

Travel Planner application to give added support to the users. This agent is shown as a blue genie on the right side of the figure. The agents have both speech and ballooned text to guide users and help them understand and use the Travel Planner better. The visual agents we used in our system were also implemented as aspects contained in aspect-oriented components so that they can be located, accessed and modified more easily. This is in line with our view to maintain consistency to enable increased understanding and coherency in our designs and code implemented using AOCE. We observed that this also made any refactoring or maintenance of any part of the program easier, faster and less stressful.

11.1 Experiences in Developing the Aspect-oriented Web services Travel Planner using Visual Studio .NET

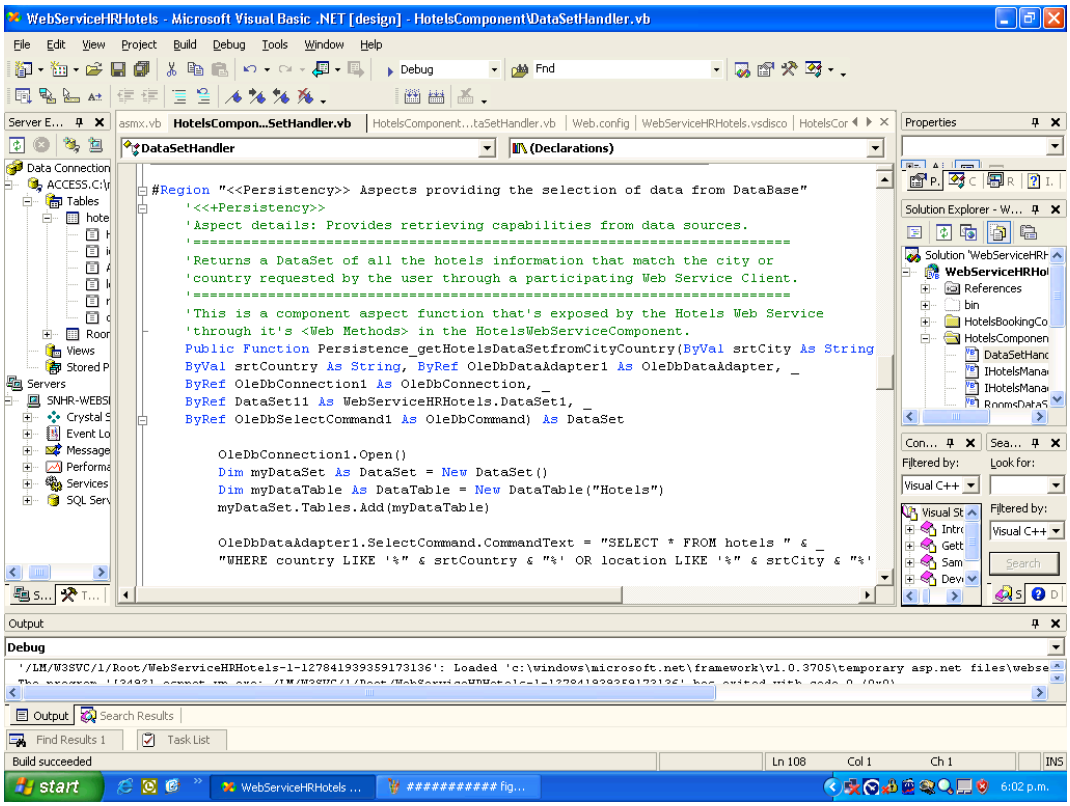


Figure 11.2: Implementation of the AOWS based collaborative Travel Planner system was done using Visual Studio .NET.

Microsoft's Visual Studio .NET (Microsoft .NET website 05), as shown in Figure 11.2, was the Interactive Development Environment (IDE) used to implement the AOWS based collaborative Travel Planner system including the entire collection of aspect-oriented web service providers and requesters. An implementation of our web service client involving a thick client system is shown below in Figure 11.3. This is the thick client's main user interface of the collaborative Travel Planner system that

utilises .NET's Windows Forms. Using this interface, users can search for hotels, airlines, rental cars and trains and make bookings for them.

Figure 11.3: Windows Form version of the main interface for AO-Componentised Travel Planner

The implementation for the AOConnector, AOComposite, AOUDDI, adaptors and validating agents was also done using the VS .NET integrated development environment (IDE). We already have experience building normal web service based applications using java and J2EE, as such so that we can expand our knowledge base and apply a new, different and increasingly useful technology, we did the implementation in this thesis using Microsoft's platform and languages. Visual Studio .NET was chosen because it provides a feature-rich (Ferrara and MacDonald, 02) IDE

through the large number of libraries it contains. It was found to be quite comprehensive and self-contained to a very large extent. There was also help available in Microsoft .NET Help (Microsoft .NET website, 05), books and on the internet relating to VS .NET and the languages it support. It also supports advanced XML Web Services (Torkelson et al 02, Foggon et al 04) development, including facilities for locating and consuming them using the same IDE, and as such, helped us do fast prototyping.

11.1.1 AOCE .NET web services

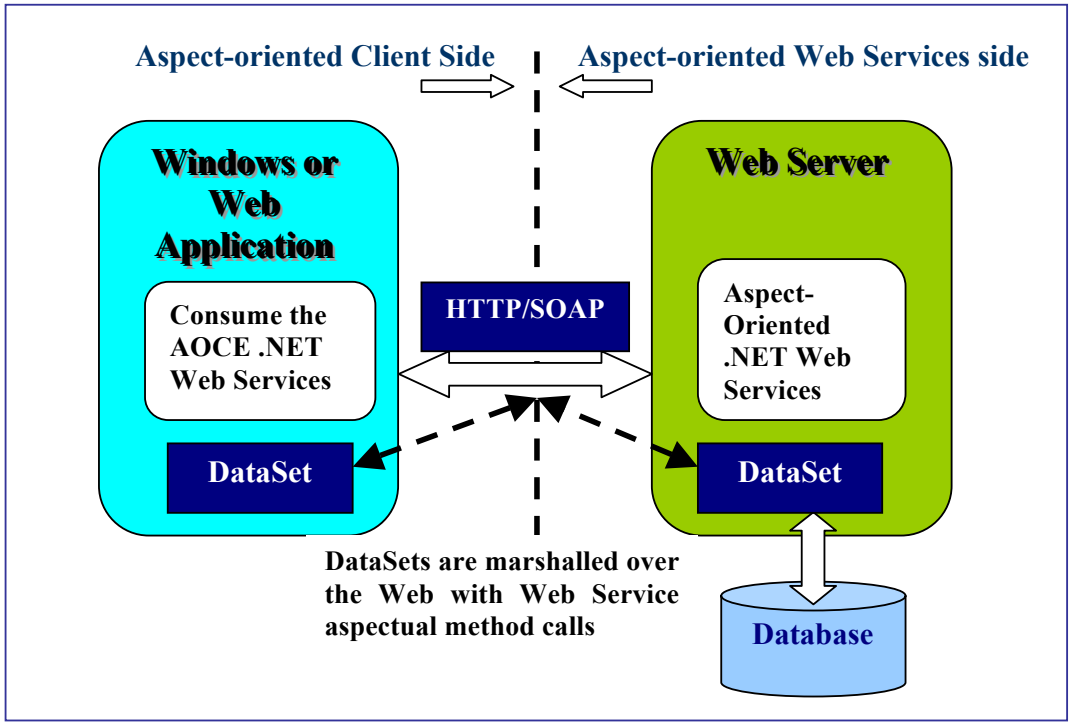


Figure 11.4 –Aspect-oriented .NET Web Services with DataSets as parameters for remote data access

The aspect-oriented XML web services were implemented in C# and Visual Basic. These languages have been enhanced in the new version with their release in Visual

Studio .NET. Remote data access in the aspect-oriented .NET web services was achieved by using DataSet objects as parameters. This is depicted in Figure 11.4. .NET can handle all the details of persisting and restoring a full DataSet object over a Web Service. This includes the ability to merge changes into the database, thus providing a powerful tool for building remote data services over the Web. This could also be achieved within an intranet of any size. The DataSet was very versatile and could be tailor made to suit our needs to contain any data that we wanted it to hold. Further it could easily be sent over the wire and can be manipulated by applications whether online or offline once the dataset has arrived at its destination.

It was found that concurrency issues are yet to be resolved. Other problem areas include situations where a database has been accessed and updated by one client while another client was still considering what changes to make. In this case, if the latter client, for instance tries to make an update on the changed data, he might receive a reply saying that the data does not exist anymore and such a query cannot be executed. This can be very troublesome especially in cases where there is heavy simultaneous use of the service by remote clients. Also data access over the wire using web services was slower as compared accessing data stored locally. Executing the remote functions also takes longer compared to using functions local to the application.

11.1.2 Experience using AOCE terminology in Designs and Implementations

During the design and implementation stages, methods were devised to increase the ease and rate of comprehension of designs and code so that it would be easy to modify, scale-up, refactor or maintain any applications that were made using Aspect-Oriented Component Engineering techniques. These methods include using proper and consistent aspects names and inserting additional notes or comments in the all the designs and code about the aspects to explain their use.

```
<<Performance>> Aspects providing performance based on look-up times for Rooms in Tables.

<<Security>> Aspects providing authentication facilities to users.

<<Persistency>> Aspects requiring the function to store to Hotels DataBase.

#Region "<<Persistency>> Aspects providing for the update of data in DataBase. "
    '<<+Persistency>>
    'Aspect details: Provides updating capabilities for data sources.
    '=========================================================================
    'Updates a DataSet of the Rooms table that matches the room booked by
    'the user utilising a participating Web Service Client.
    '=========================================================================
    'This is a component aspect function that is exposed by the Hotels Web Service
    'through it's <Web Method> in the HotelsWebServiceComponent.
    Public Function Persistence_updateRoomsTablefromRoomID(ByVal srtRoomID As String, _
    ByRef OleDbDataAdapter2 As OleDbDataAdapter, ByRef OleDbConnection1 As OleDbConnection, _
    ByRef DataSet21 As WebServiceHRHotels.DataSet2, ByRef OleDbUpdateCommand2 As OleDbCommand)
        Dim iRoomID As Integer = CInt(srtRoomID.Trim())
        Try
            OleDbConnection1.Open()
            OleDbUpdateCommand2.Connection = OleDbConnection1
            OleDbUpdateCommand2.CommandText = "UPDATE RoomsTable SET Vacant='no more' WHERE RoomID = " & iRoomID & "
            OleDbDataAdapter2.Update(DataSet21, "RoomsTable")
            OleDbDataAdapter2.UpdateCommand.ExecuteNonQuery()
            OleDbConnection1.Close()
        Catch ex As Exception
            Debug.WriteLine("Exception in Persistence_updateRoomsTablefromRoomID = " + ex.ToString())
        End Try
    End Function
#End Region
```

Figure 11.5. Sample of code snippet from Visual Studio .NET showing the collapsible region between the “#Region” and “#End Region” tags.

The IDE of Visual Studio .NET has tags of type “#Region” and “#End Region” that provide a convenient way to encapsulate and isolate our aspects. As can be seen from

Figure 11.5 the code between these two tags is collapsible to hide it. When collapsed, it only shows the aspect's name and brief description of the aspect details required or provided by the component. This region can be expanded to make the aspectual code visible and editable again. All the component aspects were clearly and systematically placed between these collapsible tags throughout code in the program. In the figure, the first 3 aspects, i.e. for performance, security and persistence, are shown collapsed. The persistency aspect that provides for updating of data is shown in its expanded form. The comments about the aspects are also of a different colour as opposed to the code proper. Similar isolation and identification techniques for aspects can also be achieved using Java IDEs like Borland's JBuilder, Eclipse's Java IDE and Sun Microsystems' Sun ONE Studio (Sun Microsystems website 05).

We recommended that comments regarding aspects and their details be standardized and written just above the aspects. The details relating to the class itself were written at the beginning of the class concerned. These commenting format was a necessary feature of all our AOCE programs using Visual Studio .NET. This technique would make it easier for anyone reading the code to understand the code faster and better because it is coherently and consistently used. It would also allow more efficient refactoring to be carried out if necessary. In earlier AOCE implementations using Java, each kind of aspects were coloured with a different background using the Aspect-jEdit (Panas et al 03) tool. The obvious side effect was that there were too many comments in the code. But this was a small price to pay compared to the benefits of improved readability and comprehensibility achieved through using these methods.

We prefixed all aspectual functions with the type of aspects they belong to. For instance, the "selectRoomsFromHotel(string Hotel)" aspect in C# is of the "Persistence" type. We rename this aspect by prefixing the "Persistence" word to it, separated by an underscore character, as rewrite it as "Persistence_selectRoomsFromHotel(string Hotel)". Likewise we use this same convention for all the other types of aspects, e.g. for Distribution, Transaction Processing (TransactionProcessing), User Interface (UserInterface), Resource Utilization (ResourceUtilization), Performance, etc. This allows for rapid and more accurate discovery of the exposed services based on types of aspects by discovery agents and clients.

Clear and concise documentation was also provided about the applications that were designed and implemented using AOCE. These contain direct references on how to use aspects efficiently in the components. The documentation also contained information on how to how to scale up or refactor the application. This included how to modify the code to add or remove functionality from the application.

11.2 Evaluation, Tests and Validations

In the following sections we describe and discuss the extensive evaluations, tests and validations that were performed to evaluate our AOWS based systems and AOCE. This includes also an evaluation through questionnaire feedback that is described below.

11.2.1 AOCE and AOWS evaluation through questionnaire

To test the AOCE methodology, during the summer school at the beginning of 2005, 8 final year software engineering students had volunteered to do development of web service based-systems for mobile applications using the AOCE development methodology and Aspect-Oriented web services. We managed to contact 5 of the students (the others were international students who had returned to their country of origin) and these 5 students were asked to voluntarily give us their feedback through an evaluation questionnaire. We had obtained prior approval from the Ethics Committee of The University of Auckland before approaching the students. The application forms and letter of approval are attached in the Appendix to this thesis.

11.2.1.1 The Evaluation Questionnaire

The questionnaire given to the students is as follows.

Title: USABILITY TESTING QUESTIONNAIRE

1. How much do you know about AOCE?

Nothing Level of expert

☐ ☐ ☐ ☐ ☐

1 2 3 4 5

2. How much do you know about Web Services systems?

Nothing Level of expert

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

3. Did you find it easy to learn the AOCE development methodology?

Very difficult Very easy

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

……

……

4. Did you ever develop software using components or a component based software development methodology prior to using AOCE?

Yes ☐ No ☐

If your response to the question above is "yes", where did you use it? (You may tick more than one option here, if appropriate):

a.) At work ☐

b.) During large scale group projects as part of my coursework at University ☐

c.) During individual software development assignments at University ☐

5. Do you find it easier to use AOCE for the development as compared with other development techniques that you have used?

Very difficult No difference Very easy

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

6. Is it easier to follow and understand your code when you were using AOCE?

Very difficult No difference Very easy

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

7. Was the AOWSDL document that was supplied more useful when compared with the normal WSDL document?

Not useful No difference Very useful

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

8. Does using AOCE make the code better than that written without using AOCE?

Worst No difference Very much better

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

9. Is the AOUDDI better than the normal UDDI?

Worst No difference Very much better

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

10. Did using AOCE allow the applications to be more easily refactored when compared with those developed without using AOCE?

Very difficult No difference Very easy

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

11. Were the web service based applications built using AOCE easier to maintain when compared with those developed without using AOCE?

Very difficult No difference Very easy

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

12. Are the web service based applications built using AOCE more easily scalable when compared with those developed without using AOCE?

Less scalable No difference More scalable

☐ ☐ ☐ ☐ ☐

1 2 3 4 5

Reason(s):

……

……

……

……

13. Are the web service based applications built using AOCE more understandable when compared with those developed without using AOCE?

Less understandable No difference More understandable

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

14. Are the web service based applications built using AOCE more reusable when compared with those developed without using AOCE?

Less reusable No difference More reusable

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

15. Are the aspect-oriented components in the web service based applications built using AOCE more understandable when compared with those developed without using AOCE?

Less understandable No difference More understandable

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

16. Are the aspect-oriented components in the web service based applications built using AOCE more reusable when compared with those developed without using AOCE?

Less reusable No difference More reusable

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

17. Are the aspect-oriented components in the web service based applications built using AOCE better characterized when compared with those developed without using AOCE?

Less characterized No difference More characterized

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

18. Are the aspect-oriented components in the web service based applications built using AOCE better categorized when compared with those developed without using AOCE?

Less categorized No difference More categorized

1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐

Reason(s):

……

……

……

……

19. General Comments

Any general comments you have related to AOCE, AOWS and/or their concepts.

……

……

……

……

……

……

……

……

……

11.2.1.2 The Results and Analysis

The results from the above evaluation are tabulated in Table 11.1 below.

Number of the Response:	1	2	3	4	5
Question 1	0	0	1	1	3
Question 2	0	0	0	2	3
Question 3	0	0	0	2	3
Question 5	0	0	1	3	1
Question 6	0	0	0	3	2
Question 7	0	0	0	1	4
Question 8	0	0	0	2	3
Question 9	0	0	0	1	4
Question 10	0	0	0	2	3
Question 11	0	0	0	3	2
Question 12	0	0	0	2	3
Question 13	0	0	0	3	2
Question 14	0	0	0	1	4
Question 15	0	0	0	1	4
Question 16	0	0	0	1	4
Question 17	0	0	0	0	5
Question 18	0	0	0	0	5

Table 11.1 (a.) All the results except Question 4

The Responses for Q 4:	Yes	No	a	b	c
Question 4	5	0	2	4	3

Table 11.1 (b.) The results for Question 4

Table 11.1: The results from the evaluation done on AOCE and AOWS

The results for question 4 is recorded separately it the Table 11.1(b.) shown above because its responses are of a different format when compared to the rest of the

questions. Also question 19 asks for any general comments and the responses (if any) to this optional question are discussed in the following paragraphs together with the rest of the responses below. Table 11.2 below contains the record of the analysis of the results from the evaluation, written in percentages based on the responses.

The % of Responses	1 (%)	2 (%)	3 (%)	4 (%)	5 (%)
Question 1	0	0	20	20	60
Question 2	0	0	0	40	60
Question 3	0	0	0	40	60
Question 5	0	0	20	60	20
Question 6	0	0	0	60	40
Question 7	0	0	0	20	80
Question 8	0	0	0	40	60
Question 9	0	0	0	20	80
Question 10	0	0	0	40	60
Question 11	0	0	0	60	40
Question 12	0	0	0	40	60
Question 13	0	0	0	60	40
Question 14	0	0	0	20	80
Question 15	0	0	0	20	80
Question 16	0	0	0	20	80
Question 17	0	0	0	0	100
Question 18	0	0	0	0	100

Table 11.2 (a.) The analysis of all the results except Question 4

% of Responses for Q 4	Yes (%)	No (%)	a (%)	b (%)	c (%)
Question 4	100	0	22	44	33

Table 11.2 (a.) The analysis of all the results except Question 4

Table 11.2 The analysis of the results from the evaluation.

The analysis of the results for question 4 is again recorded separately in Table 11.2 above as it is of a different format when compared to the rest of the questions. In the rest of this subsection we will analyse and further discuss the evaluation based on the above results and the additional comments given by those who did the evaluation. We will also present graphs for the results obtained to make our discussions to the evaluation clearer.

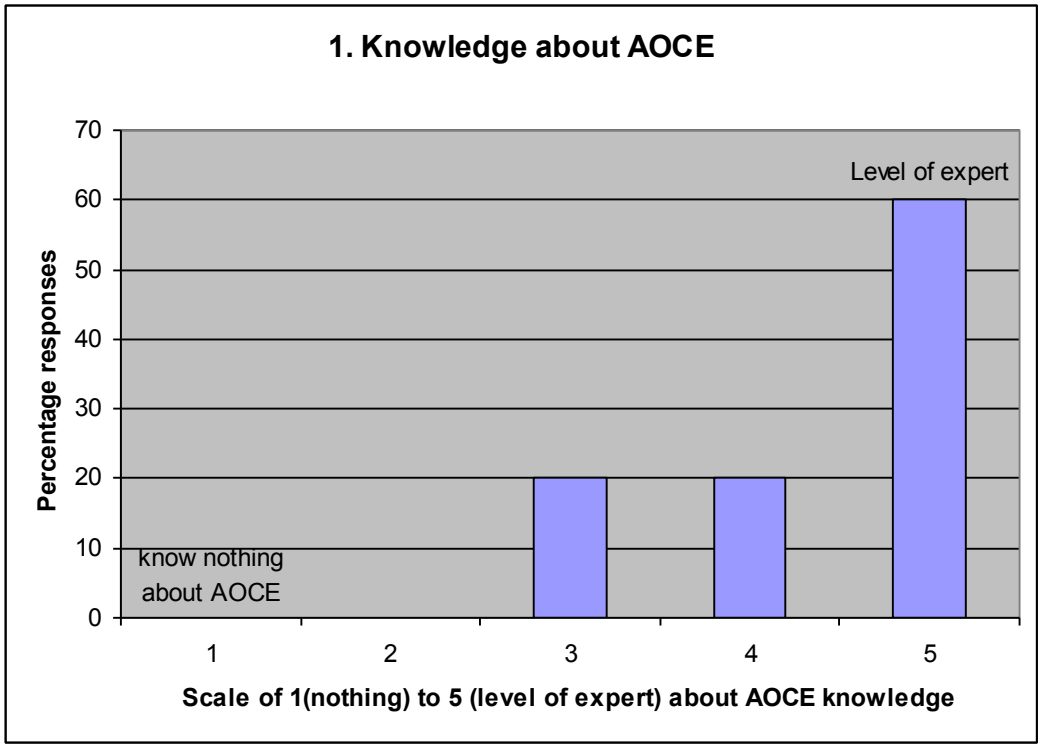


Figure 11.6: Knowledge about AOCE of the software engineers.

Figure 11.6 shows the graph of the knowledge level of the software engineers about AOCE who answered the questionnaire. It shows that all of them have a good working knowledge about AOCE with 60% at the level of expert. This is a good indication that we dealt with the correct select group, i.e. those who possessed proper knowledge and were able to understand the questions posed to them. This make our tests and the outcomes/evaluations of results fair and unbiased, because only people

with the requisite knowledge were approached to answer the questions. It excluded people who do not possess working knowledge about AOCE from our evaluation. If the event occurred whereby the volunteers had forgotten about AOCE, we would have given them a quick revision course on AOCE so that they can recollect what they did earlier in the year on AOCE and mobile systems.

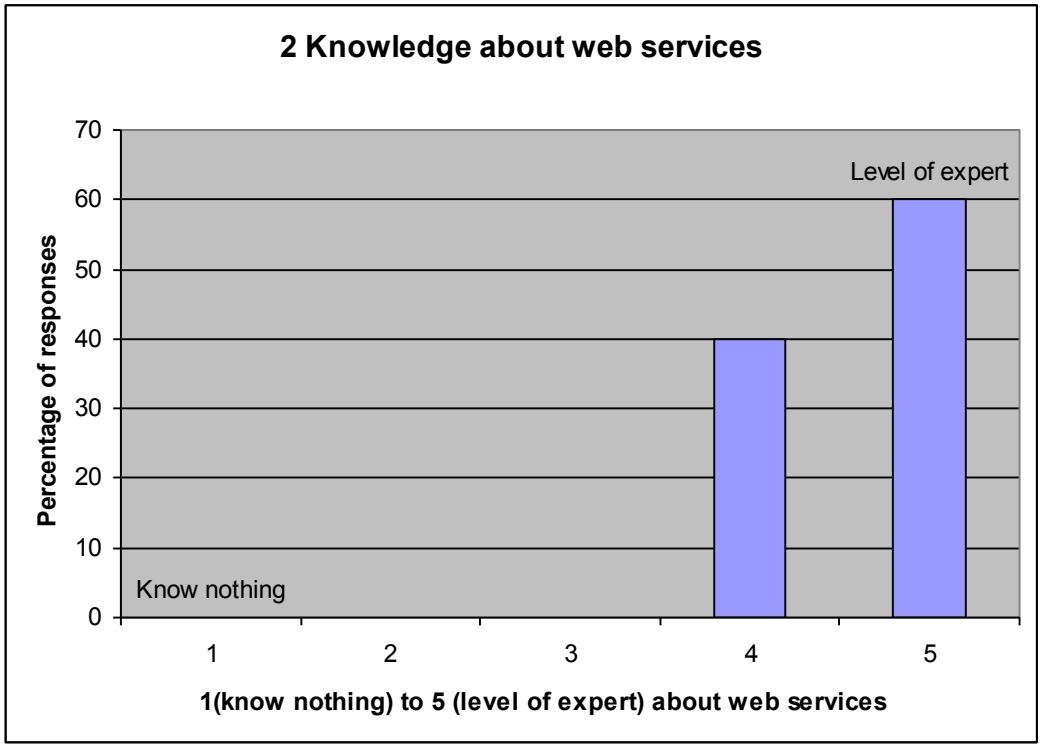


Figure 11.7: Knowledge about web services systems

The graph representing the knowledge level of the software engineers about web services systems is shown in Figure 11.7 above. It shows that all the people have a very good knowledge of web services systems with 60% at the level of expert and the remainder above average. This is a good indication that the select group possessed proper knowledge about both web services and AOCE which is central to our research. This make our tests and their outcomes/evaluations of results fair and unbiased, because only people with the requisite knowledge answered the questions and it excluded people who do not possess proper knowledge about web services systems from this evaluation.

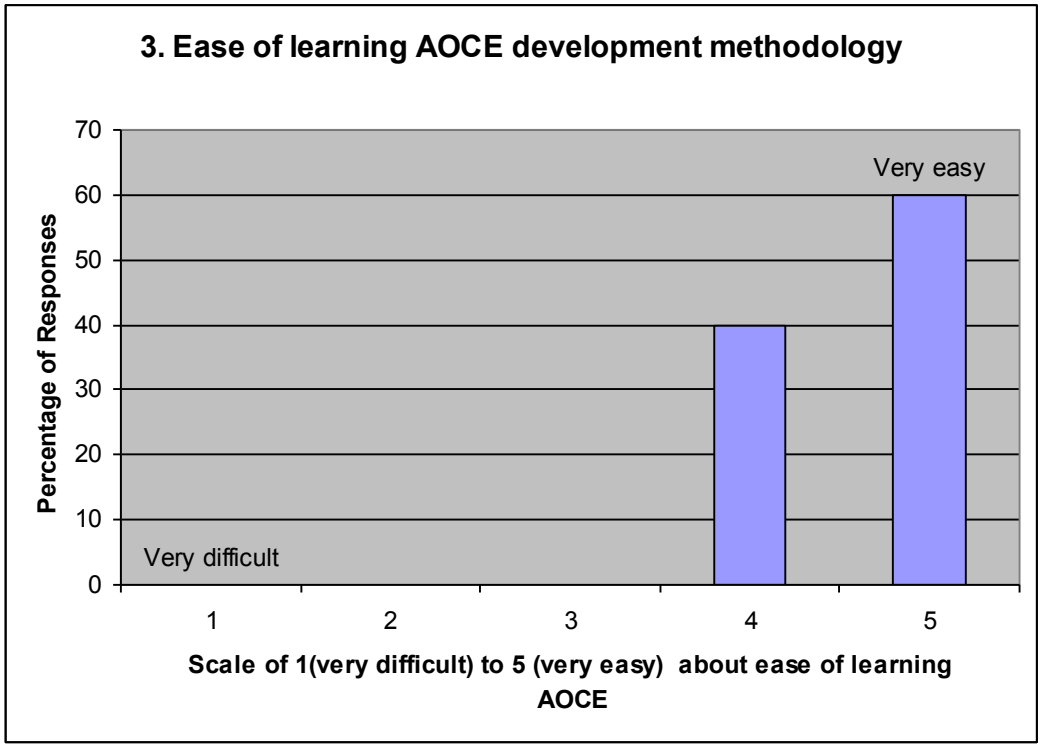


Figure 11.8: Ease of learning AOCE development methodology

Figure 11.8 above shows the graph about the ease of learning the AOCE development methodology. All the responses show that it is not difficult to learn this methodology with 60% saying that it was very easy. The reasons given are as follows:-

> AOCE is a clear and thorough methodology, where aspects could be thought of as 'labels' for methods while components as 'more advanced and clearly defined classes whose services can be ascertained from their interfaces'. Aspects clarify information and categorise the components so that they could be easily 'plugged-in' into software, and there are only few steps you need to follow to use this methodology. Furthermore the ex-students stated that they already had prior knowledge about developing software using components, as such this served more as an extension to the work they had already done and made it relatively easy to learn this new methodology.

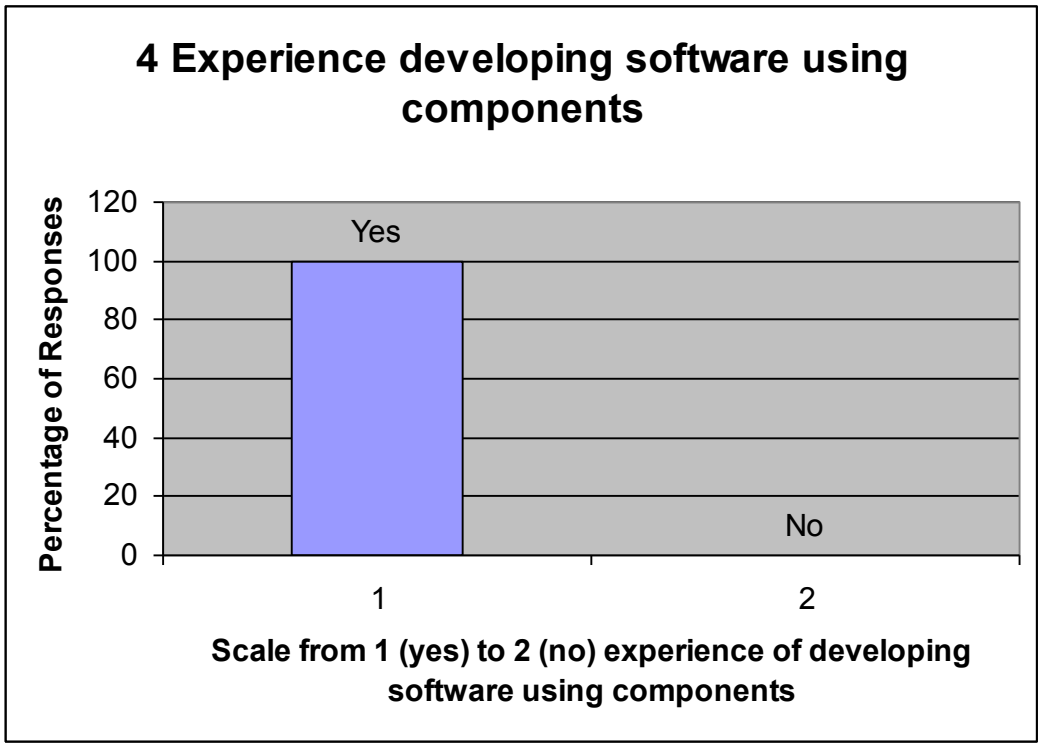


Figure 11.9: Experience of developing software using components or a component based software development methodology prior to using AOCE

From the graph shown in Figure 11.9, all the members of the group of developers answering this questionnaire had experience developing software using components or a component based software development methodology prior to development using AOCE. This is good because they would be able to critically assess the use of AOCE and will have existing knowledge to be able to compare development of software with and without AOCE.

40% of them had developed software using components or a component based software development methodology at work, 80 % during large scale group projects as part of their coursework at University and 60% during individual software development assignments at University. In this part of the questionnaire they were allowed to choose multiple options if necessary.

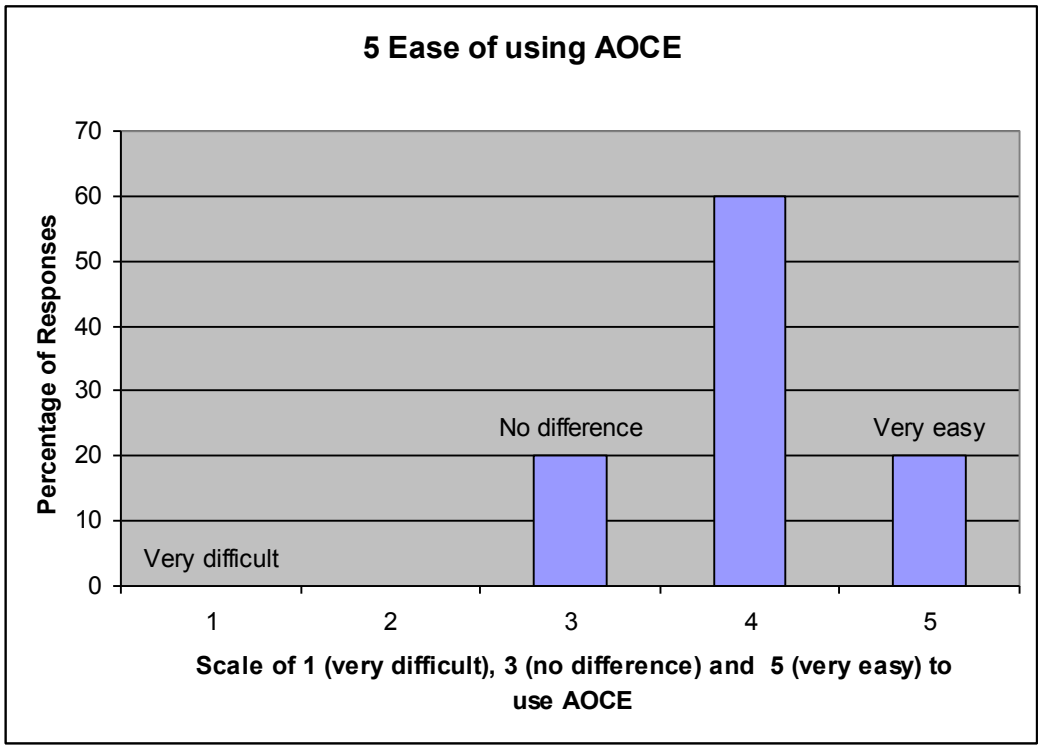


Figure 11.10: Ease of using AOCE for development as compared to other development techniques

The ease of using AOCE for development as compared to other development techniques is shown in Figure 11.10 above. 80% found development with AOCE to be easier when compared to other techniques while 20% found it to be no different. The reasons given are as follows:-

> AOCE is easy to use because the underlying code is aspectized and more understandable when compared to non-AOCE code. It is better and easier when compared to the other methodologies but we need to be more careful and think a bit more when constructing the interfaces of aspect-oriented components. It is easy to differentiate the methods in the code and the components are better characterised and categorised. It is also easy to understand code written in AOCE by others and easy to search and locate methods. One suggestion is to make AOCE even better by combining it with other development methodologies like eXtreme Programming.

The reason given by the one respondent who differed from the rest and indicated that ease of using AOCE was no different to using other techniques is that "AOCE needs more thinking when making the interfaces, as such AOCE offsets its ease of its use".

As such based on the responses, it can be deduced from the overall comments that AOCE is easier to use for software development as compared to other development methodologies.

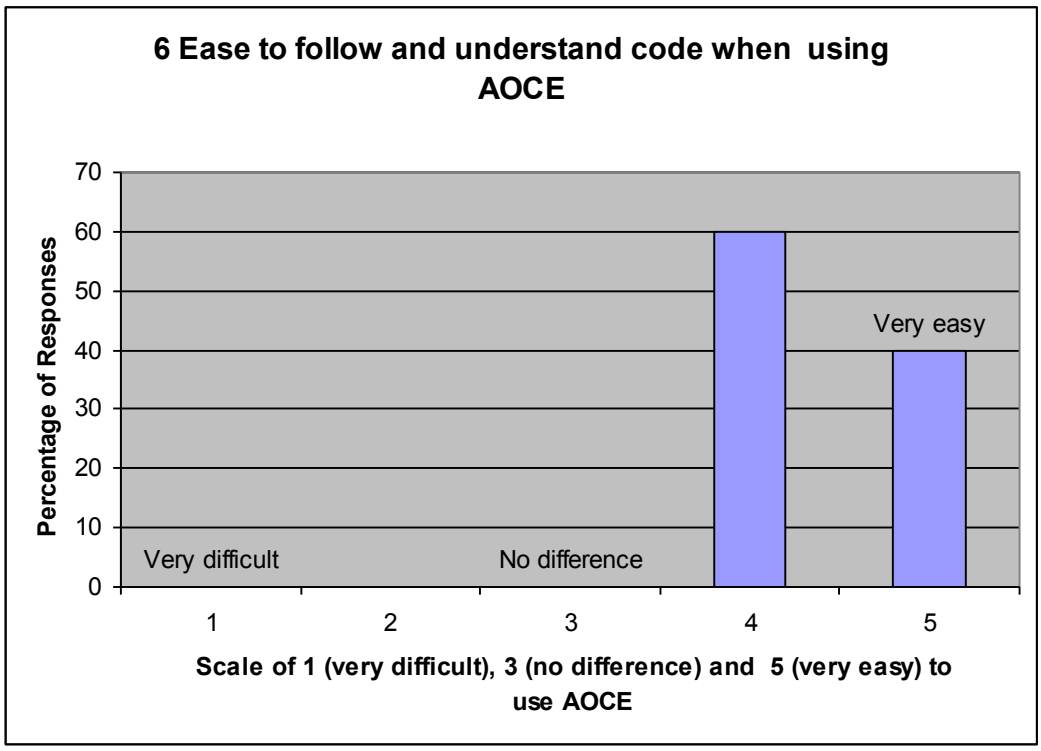


Figure 11.11: Ease to follow and understand code when using AOCE

All the students indicated that it is fairly easy to follow and understand code when using AOCE. As shown in Figure 11.11 above, the graph shows that 40% of those who answered stated that it is very easy to follow while the remainder found it easier than following code written without AOCE techniques. This shows that writing code with AOCE is easier to understand than that written without using it. The reasons given are as follows:-

> Aspects contain more information and help the developers to more easily understand each method, while the component's interface makes the functions of the component easier to follow and understand. The code is decomposed more logically through AOCE and it identifies the cross-cutting issues involved. Categorisation and characterisation using AOCE help programmers read the code easier and more comprehensively, but it also increased the time required to read the methods since their names become longer due to the

aspect tags applied before their names and additional aspectual information inserted in comments.

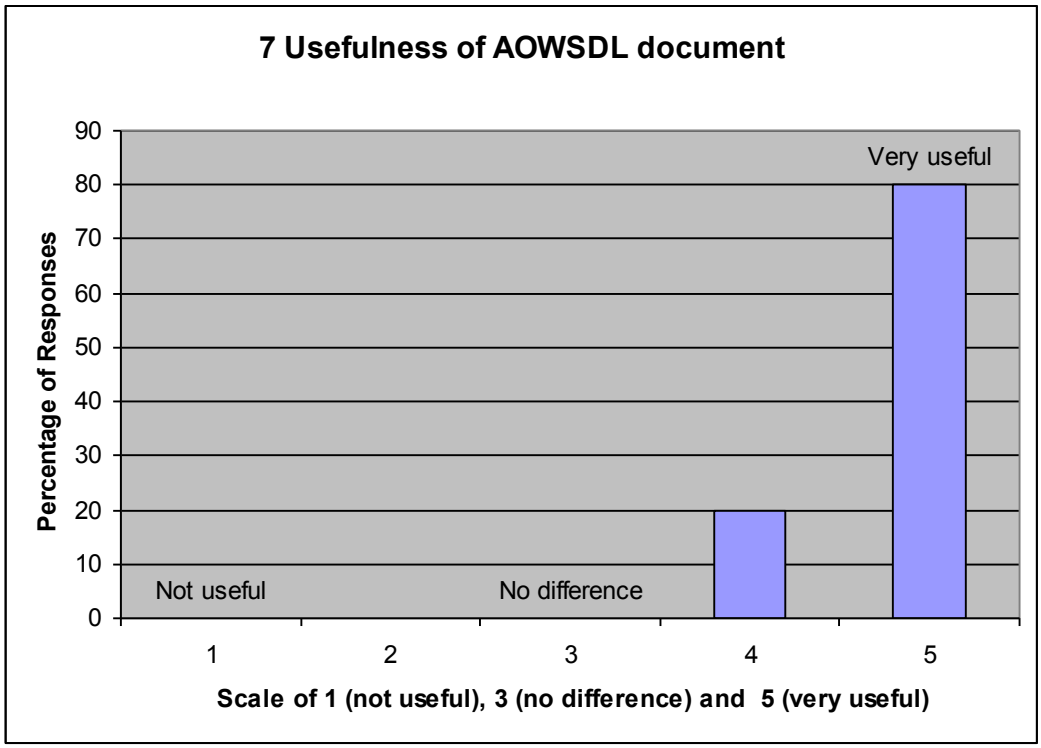


Figure 11.12: Usefulness of AOWSDL document that was supplied when compared to the normal WSDL document

Figure 11.12 shows the graph about the usefulness of AOWSDL document that was supplied when compared to the normal WSDL document. All those answering the questionnaire claim that the AOWSDL document is more useful than normal WSDL document with 80% saying that it is indeed very useful. The reasons given are as follows:-

> It provides additional important information about the components used by the webservices and as such allows the components to be advertised and/or reused with ease. More description in discovery documents means more autonomy can be can be supported by the system because it becomes more comprehensive and informative. By containing more information, which is well described and non-abstract, the WSDL becomes more understandable and thus more useful.

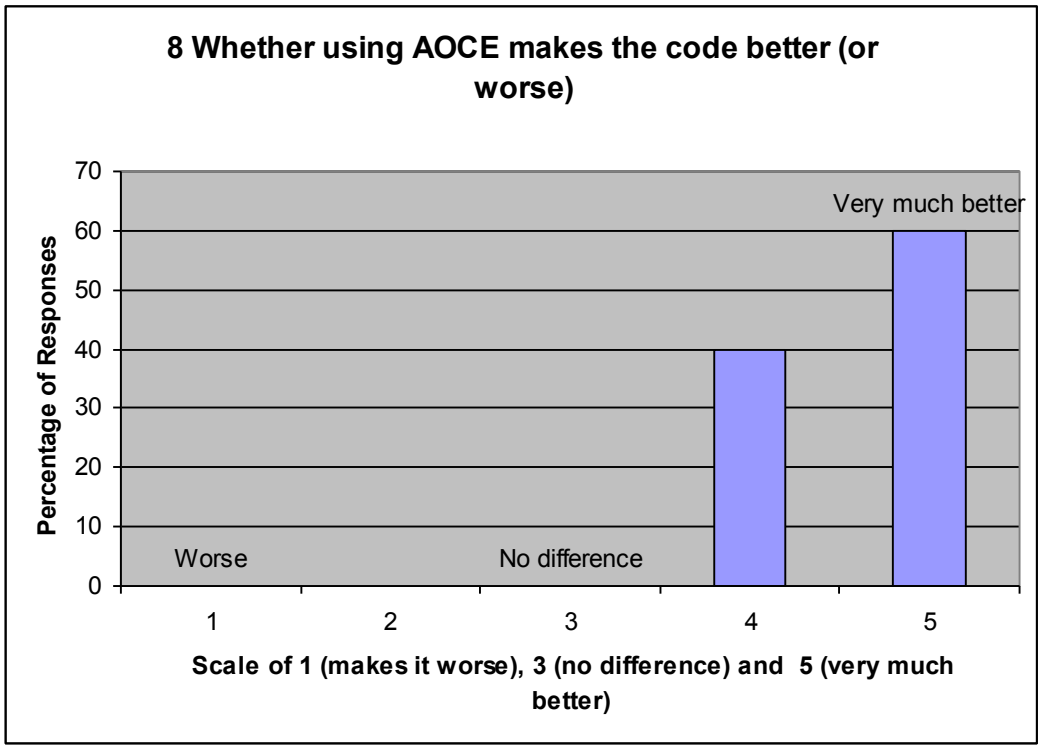


Figure 11.13: Whether using AOCE makes code better compared to that written without using AOCE

Figure 11.3 shows the graph about the quality of code written using AOCE as compared to that written without using AOCE. All those answering the questionnaire claim that the code written using AOCE is better, with 60% saying that it is very much better. The reasons given are as follows:-

> AOCE makes the code much easier to understand and is more organised and much more reusable. AOCE is only better for medium or larger sized projects, otherwise the overhead is slightly higher and may be unnecessary in small projects as non-AOCE code is relatively easy to understand in trivial applications. It is easier to understand the code and get more information from the code itself. AOCE makes the code compact (reduces code by clearly identifying, categorising and characterising the cross-cutting concerns) and thus making it better and easier to understand and maintain.

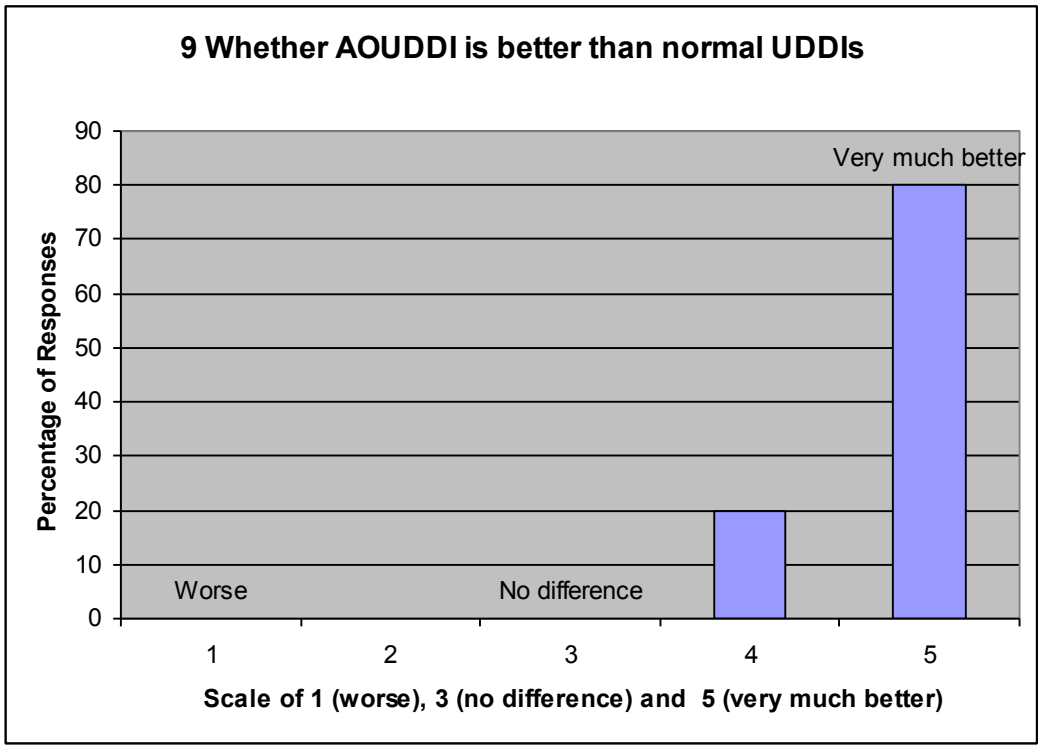


Figure 11.14: Whether the AOUDDI is better than normal UDDIs

Figure 11.4 shows the graph about the whether or not the AOUDDI is better than normal UDDI. All those answering the questionnaire claim that the AOUDDI is better with 80% claiming that it is very much better. The reasons given are as follows:-

> AOUDDI allows people to search for specific components/aspects/methods based on clearly defined aspects and descriptions making the AOUDDI smarter than the normal UDDI. The AOUDDI itself is categorised and characterised, thus making the AOUDDI more understandable. More advanced search can be done compared to UDDI (e.g. using aspects to search). More details about web-services can be provided (AOUDDI can understand AOWSDL better than normal UDDIs because, for example, UDDIs cannot interpret the information in the aocomponents element of the AOWSDL. AOUDDI can provide relevant information to the user by extracting and interpreting this relevant information from AOWSDL.) AOUDDI contains a repository of all registered AOWSDLs and as such is more likely to get desired AOWSDL because all these documents are registered in one central registry.

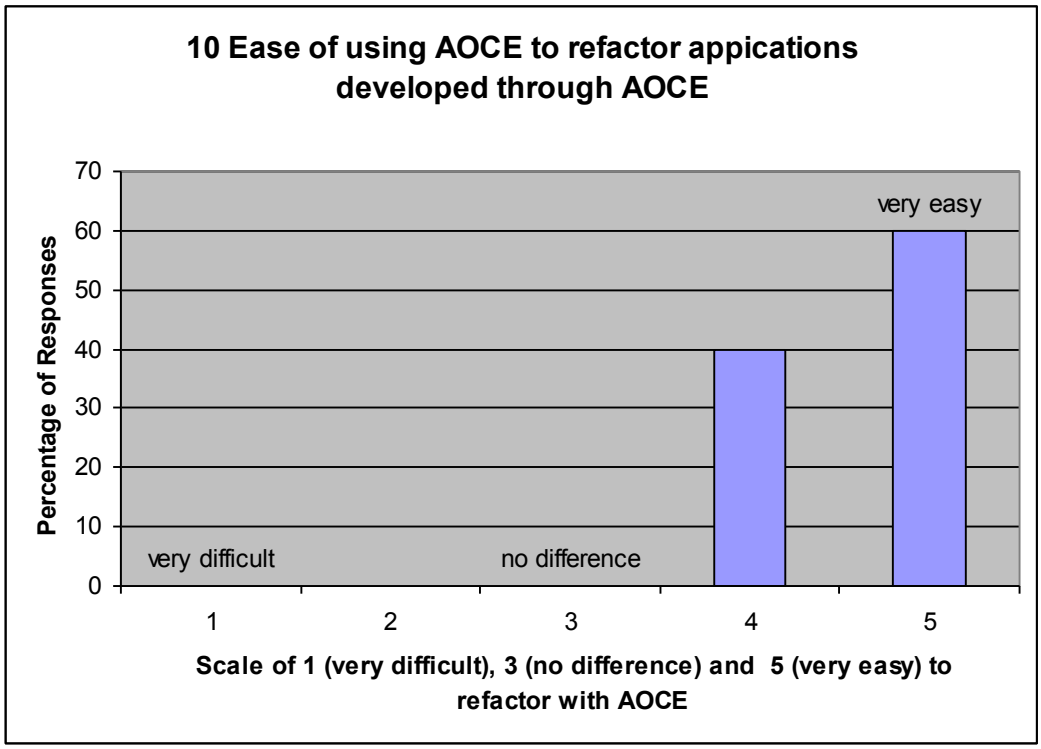


Figure 11.15: Whether using AOCE allows the applications to be more easily refactored as compared to those developed without using AOCE

Figure 11.5 shows the graph about the ease of refactoring applications as compared to those developed without using AOCE. All agree that it is easier to refactor using AOCE with 60% claiming that it is very easy to do it with AOCE. The reasons given are as follows:-

> Each component can easily be broken down (into smaller components) if it is too big. Components may be broken down with the help of the aspects used. It is also possible to scrap (remove) any old and inefficient component and replace it with new one(s). The code is more well defined and the cross-cutting issues are identified easily. Can easily refactor via aspect types but can lead to complicated code if complexity is high because didn't design components well.

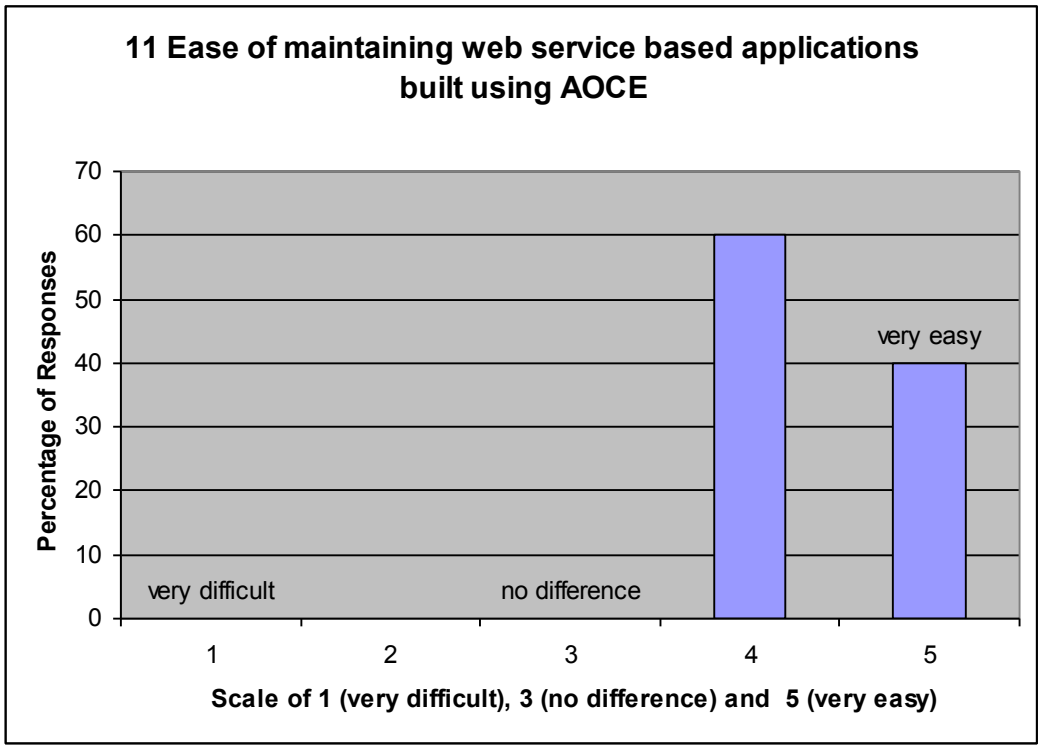


Figure 11.16: Ease of maintaining web service based applications built using AOCE compared to those developed without using AOCE

Figure 11.6 shows the graph about the ease of maintaining web service based applications built using AOCE compared to those developed without using AOCE. All the software engineers agree that it is easier to maintain the applications built using AOCE compared to those developed without using AOCE with 40% saying that it is indeed very easy to do so. The reasons given are as follows:-

> It is easier to maintain aspect-oriented components as compared to a whole bunch of classes or non-aspectized components as they allow for faster updates and are easier to integrate and add new features. The aspect-oriented components and code are also more well defined making it easier to search for and understand methods (aspect-oriented services). Because it is easier to understand, other engineers can maintain it better and develop it further.

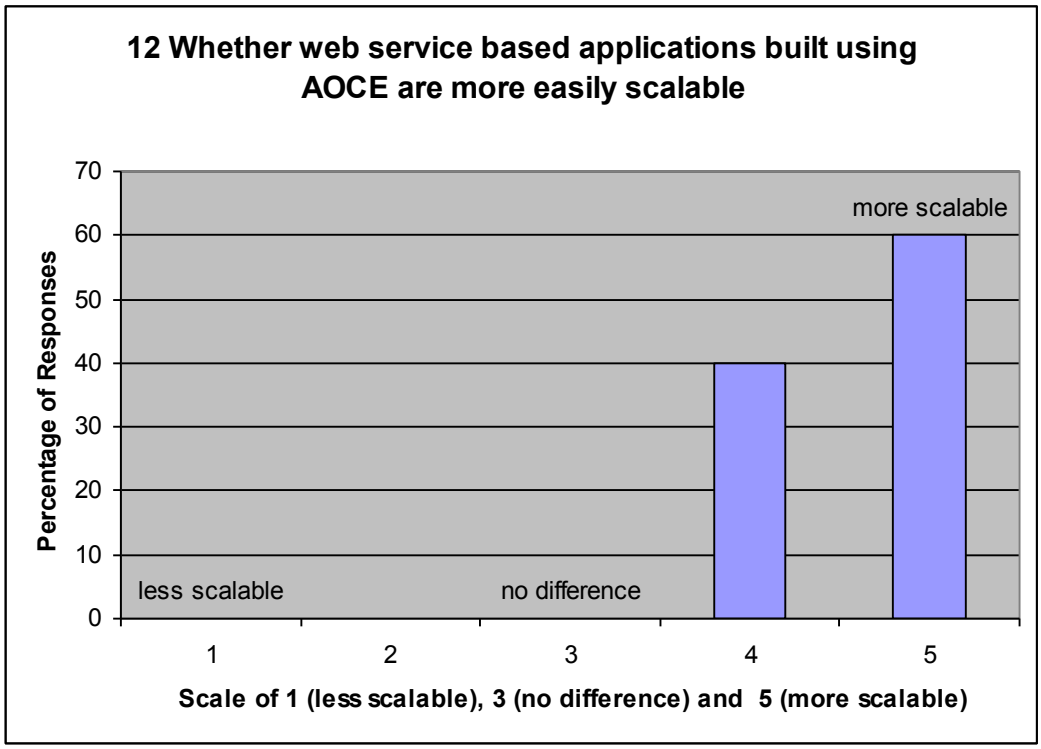


Figure 11.17: Whether web service based applications built using AOCE are more easily scalable when compared with those developed without using AOCE

Figure 11.17 shows the graph about how scalable web service based applications developed using AOCE are as compared to those developed without using AOCE. All the software engineers who answered the questionnaire claim that they are indeed scalable. The reasons given are as follows:-

> Web service based applications built using AOCE are more easily scalable because their components can easily be plugged-in/replaced and it is easy to visualize them thus making the application more manageable to scale. The code is better componentized and this allows us to add new functions and/or new components into the application more easily. They are more scalable due to the use of aspect-oriented components but scalability can become hard when required to extend the code of an interface that has already been implemented and used by other components because we might break the system (or get undesired results) if we are not careful enough.

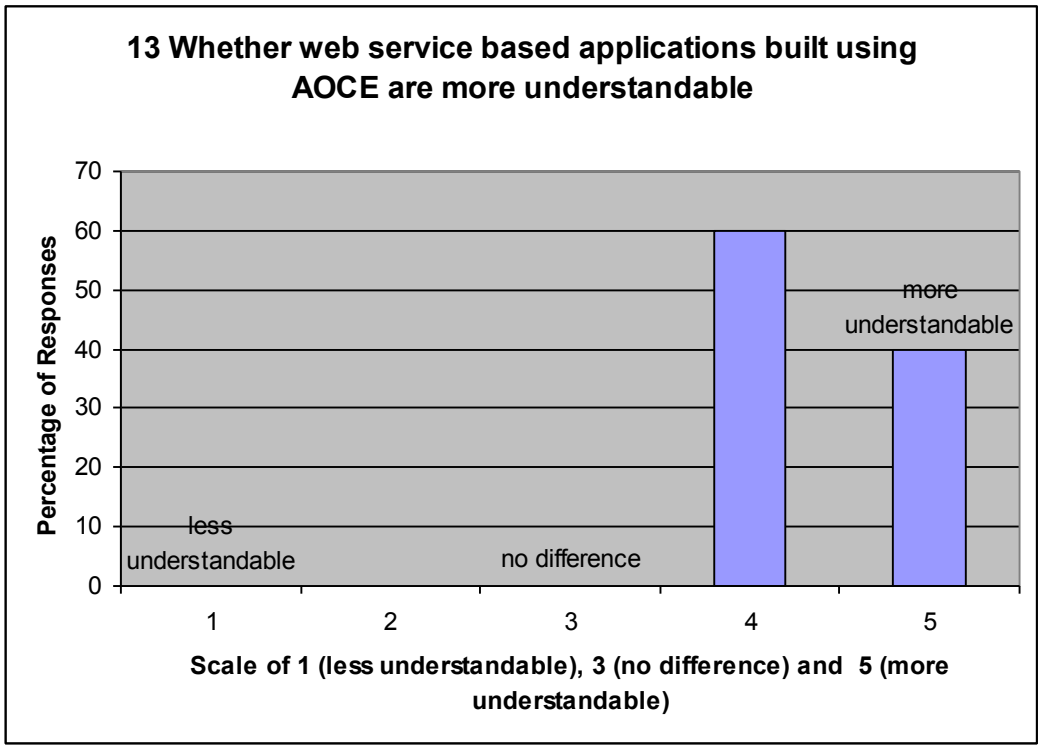


Figure 11.18: Whether web service based applications built using AOCE are more understandable when compared with those developed without using AOCE

Figure 11.18 shows the graph about how understandable the web service based applications built using AOCE are when compared with those developed without using AOCE. All those answering the questionnaire claim that web service based applications built using AOCE are more understandable. The reasons given are as follows:-

> The aspects and components make the code more characterised and categorised. Aspect-oriented components make the system clearer and better structured. Similarly, when developing stand-alone software, the application is very logically decomposed because it is aspectized. More details about components and methods are provided by the aspects and their details. It is easier to grab (locate) desired web-services, e.g. by using AOConnector, AOComposite, AOUDDI, etc. and the AOWSDL of the Aspect-Oriented Web Services.

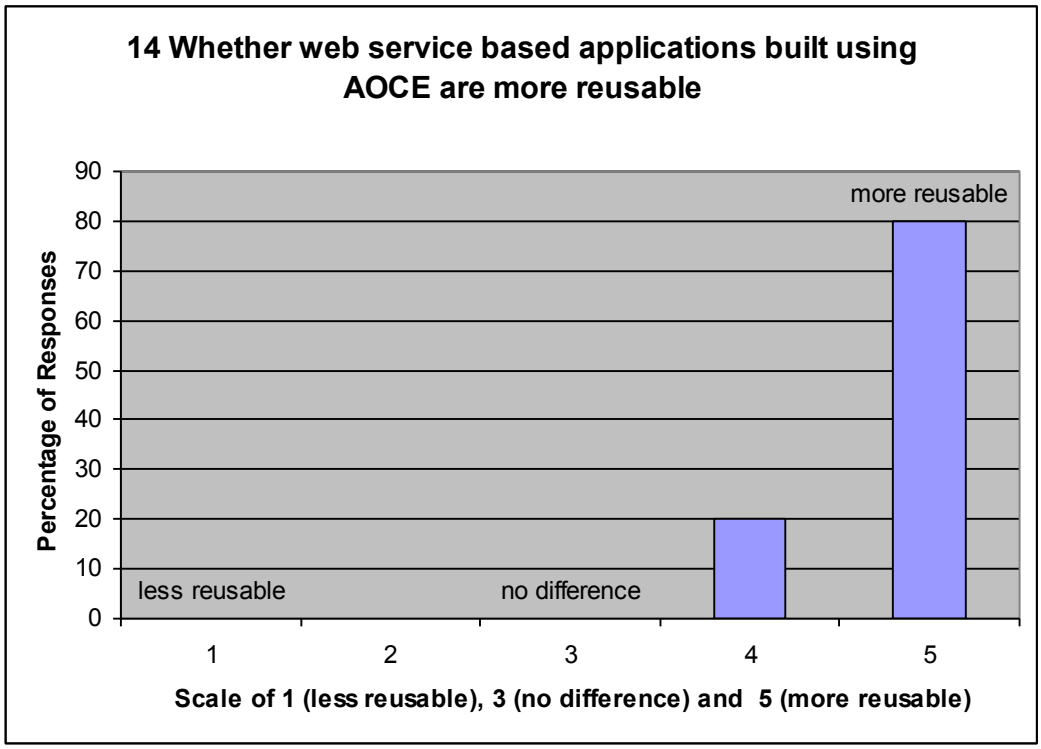


Figure 11.19: Whether web service based applications built using AOCE are more reusable when compared with those developed without using AOCE

Figure 11.19 shows the graph about whether web service based applications built using AOCE are more reusable when compared with those developed without using AOCE. All those answering the questionnaire claim that web service based applications built using AOCE are more reusable. The reasons given are as follows:-

> Aspect-Oriented Components themselves are easy to reuse, the methods in the web service are better defined with the help of aspects, hence they are easier to reuse in other application. Componentized applications are easier to reuse. The components being aspectised make them more understandable and as such more reusable. Reusability is enhanced since unwanted component's code can be easily dropped (removed) and as such only the required characteristics of the web services are reused, with the unwanted methods excluded.

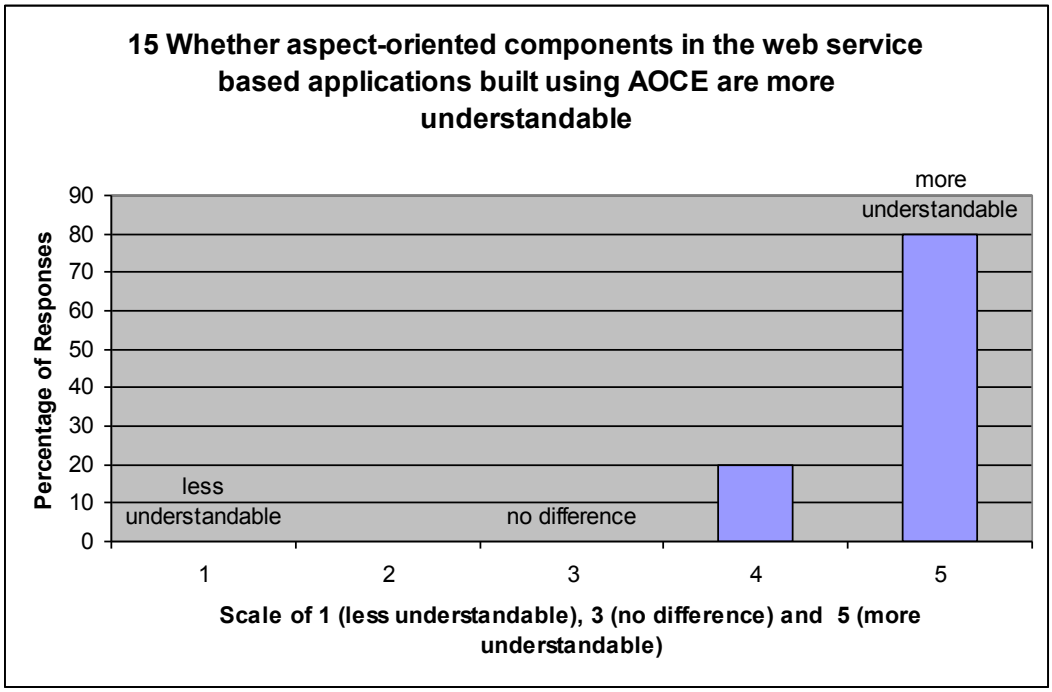


Figure 11.20: Whether the aspect-oriented components in the web service based applications built using AOCE are more understandable when compared with those developed without using AOCE

Figure 11.20 shows the graph about whether the aspect-oriented components in the web service based applications built using AOCE are more understandable when compared with those developed without using AOCE. All say that the aspect-oriented components in the web service based applications built using AOCE are more understandable. The reasons given are as follows:-

> They stated that aspect-oriented components are more understandable because there are more characteristics and details about the components and methods provided. The methods within the components are also characterised and categorised. Other reasons given are that the components built with AOCE are far more understandable due to the use of aspects, as it would take the developers a lot less time to figure out what the methods inside a component can do and what its general coding structure is. Also each component provides an interface, which becomes more readable by using the aspects, showing only required information and hiding unwanted information.

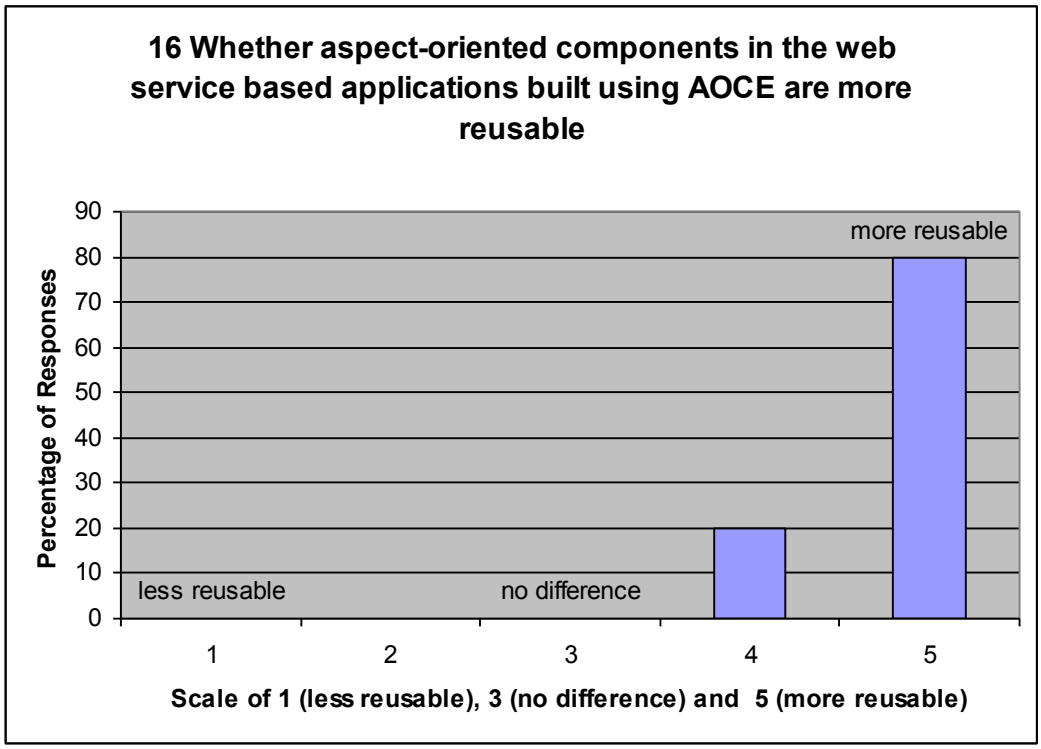


Figure 11.21: Whether the aspect-oriented components in the web service based applications built using AOCE are more reusable when compared with those developed without using AOCE

Figure 11.21 shows the graph about whether the aspect-oriented components in the web service based applications built using AOCE are more reusable when compared with those developed without using AOCE. All state that the aspect-oriented components in the web service based applications built using AOCE are more reusable. The reasons given are as follows:-

> Aspects in the componentized web service based applications make the components more reusable. With the use of aspects the methods inside the component can be easily recognised and be extracted out of the component to be reused if necessary. The aspects also categorize the component's methods which makes it even more reusable. Can reuse and modify the component easily the way developers desire to satisfy their requirements. AOCE enforces the use of aspects and good commenting, therefore ao-components will be made more characterized, i.e. aspect-oriented components are actually more reusable compared to non aspect-oriented components because aspect-oriented components are aspectized and well commented thus making them better characterised.

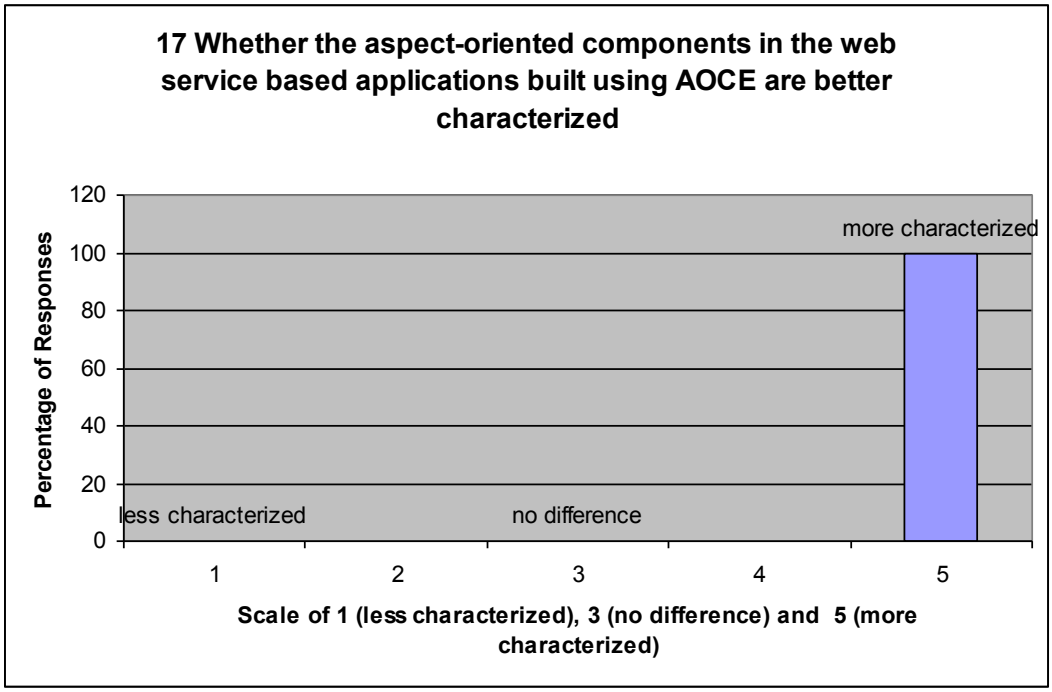


Figure 11.22: Whether the aspect-oriented components in the web service based applications built using AOCE are better characterized when compared with those developed without using AOCE

Figure 11.22 shows the graph about whether the aspect-oriented components in the web service based applications built using AOCE are better characterized when compared with those developed without using AOCE. All state that the aspect-oriented components in the web service based applications built using AOCE are better characterized. The reasons given are as follows:-

> More details are given and a standardized way of showing the details, not just normal documentation, make them better characterised. Aspects added to the methods enhances this. Each method has own aspect type which tells the characteristics of the method concerned. Aspects inside components are well described and added information about aspects and aspect detail information in the components cause them to be better characterised. The aspects identify each and every method in the component by their functionality, thus making the whole component a lot easier to be understood, therefore, aspect-oriented components are far more characterized compared to those developed without AOCE.

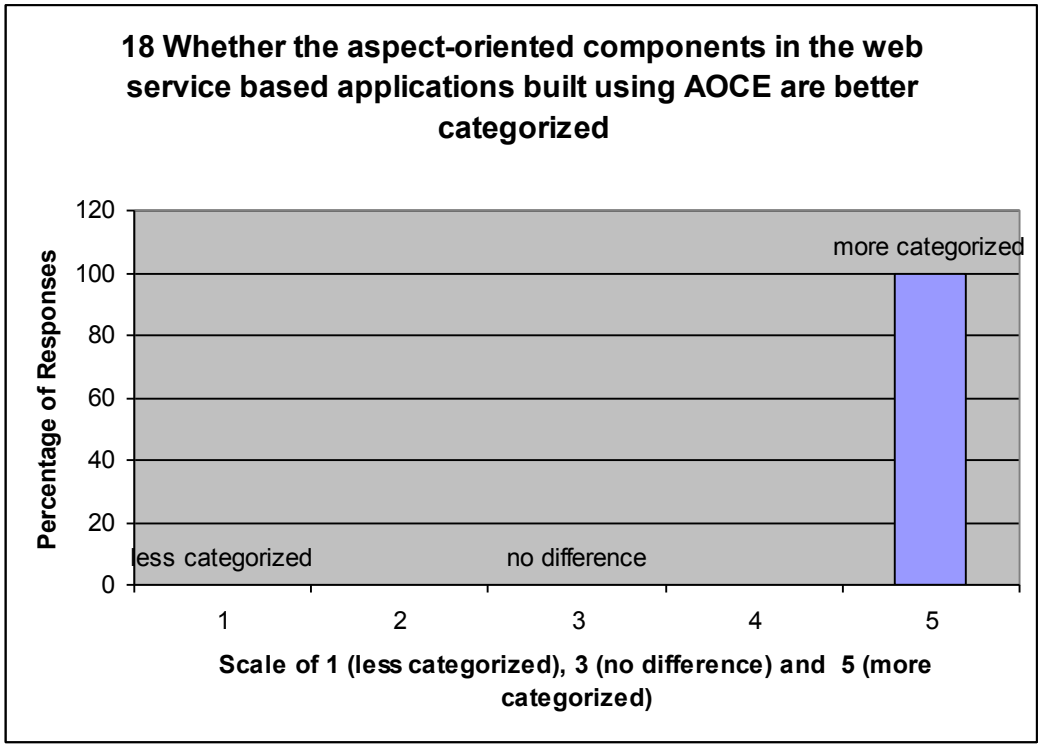


Figure 11.23: Whether the aspect-oriented components in the web service based applications built using AOCE are better categorized when compared with those developed without using AOCE.

Figure 11.23 shows the graph about whether the aspect-oriented components in the web service based applications built using AOCE are better categorized when compared with those developed without using AOCE. All state that the aspect-oriented components in the web service based applications built using AOCE are better categorized. The reasons given are as follows:-

> Using the functionality of the component to categorise is a very good way (idea) as human brains tends to think the same, i.e. better categorisation is important for understanding and aspect-oriented components are better categorised. Each method is aspectized (labelled) according to its functions. Because the methods are categorised with clearly defined aspect types and details, the aspect-oriented components are better categorised. The use of aspects categorized the methods within the component according what they can do, by having this the methods in the aspect-oriented components are clearly classified or categorized, this clarity of categorization cannot be achieved without the use of AOCE.

19 General comments:

Given below are the additional/general comments that were given by those who answered the questionnaire.

> To learn AOCE, the benefits of aspects should be introduced sooner with real working examples so that designers and developers can better apply AOCE. Not all systems suit the style of AOCE, e.g. small systems, and this should be introduced as well. To have people practising AOCE they need to see its usefulness, some working examples or case studies may be useful. Users will notice the difference when they start to use it as AOCE is very effective. As methods are categorised by aspects in the components, it is much easier to understand the purpose of the method. But a downside to this is that the name of each method tends to be become longer because we need to write the aspect type in front of the method name (which can be a bit annoying especially when it is categorised under two or more aspect types). It may also be hard to find the appropriate aspect type for some methods. Since this is a new methodology we need a stronger IDE for rapid development, e.g. IDEs with better GUI, more automation support, interfaces, standards and templates for AOWS should be provided. The code quality is better when we use AOCE compared to other methodologies. To use AOCE, developers must agree on the definition and use of the aspect type and expand the aspect type notations if new aspect types are discovered besides the currently used ones like Persistency, Security etc. If developers have different views about the use of an aspect type, then this won't be helpful. AOCE allows other developers to understand the application code easily and agreeing on the aspect type is especially useful for projects that have many developers.

This comprehensive evaluation shows that AOCE and AOWS are indeed very useful and has great potential to empower engineers to develop novel and more autonomous, useful and efficient web services based systems that are better, more understandable, scalable and maintainable and can further address the issue of tangling code raised by cross-cutting concerns.

11.2.2 Black Box Testing

Besides thorough line by line run-checks of code and debugging using the Visual Studio .NET IDE, automatic validation of the aspects by validating agents and black box testing were also carried out on the web services based collaborative Travel Planner system. These tests were done to verify that the system and its components performed according to their specifications. Black box testing was carried out on the collaborative Travel Planner system developed using AOCE techniques for system integration purposes. A sample of the tasks that we carried out, the output and results obtained during the Black Box testing are tabulated in Table 11.3 as shown below.

Task	Description	Output	Result
Run Collaborative Travel Planner Client.	Click on "UserInterface.exe" file icon.	The Travel Planner main user interface window opened.	Correct output produced.
Choose the window to search for hotels in a particular locality.	Click on the "Search For Hotels" button in the main user interface.	A window called "Hotels Finder" that allows to search for hotels based on locality was displayed.	Correct output produced.
Choose the window to search for airlines.	Click on the "Search For Airlines" button in the main user	A window called "Airlines Finder" that allows to search for airlines	Correct output produced.

	interface.	was displayed.	
Choose the window to search for trains services.	Click on the "Search For Trains" button in the main user interface.	A window called "Trains Finder" that allows to search for trains services was displayed.	Correct output produced.
Choose the window to search for car rentals.	Click on the "Search For Car Rentals" button in the main user interface.	A window called "Rental Cars Finder" that allows to search for car rentals was displayed.	Correct output produced.
Search for hotels in a particular country or city.	Enter the full or partial name of the country or city where the hotels location is required and click on the "Search for Hotels" button in the "Hotels Finder" window.	A window called "HotelsFound" was opened and it displayed all the hotels found together with their information based on the country or city entered in the query.	Correct output produced.
Search for rooms and their	Select a hotel from the list of hotels	A window called "Rooms and	Correct output produced.

information in a particular hotel.	displayed in the datagrid of the "HotelsFound" window. Click the "View Rooms" button.	Details" was opened and it displayed all the rooms in the selected hotel together with their information, e.g. availability of rooms, cost, facilities etc.	
Book a particular room from the list of rooms displayed.	Select a vacant room from the list of rooms displayed in the "Rooms & Details" window. Click the "Book Room Now" button.	A message box indicating that the room has just been booked for the user appears. The rooms database is also updated reflecting this action.	Correct output produced.

Table 11.3: Black Box tests and their results

The tests show that the collaborative Travel Planner system built using AOCE techniques conforms to the specifications and gives the correct responses. It was also found that its user interface features and functions work correctly and produces the expected output.

11.2.3 Aspects Validating Agent

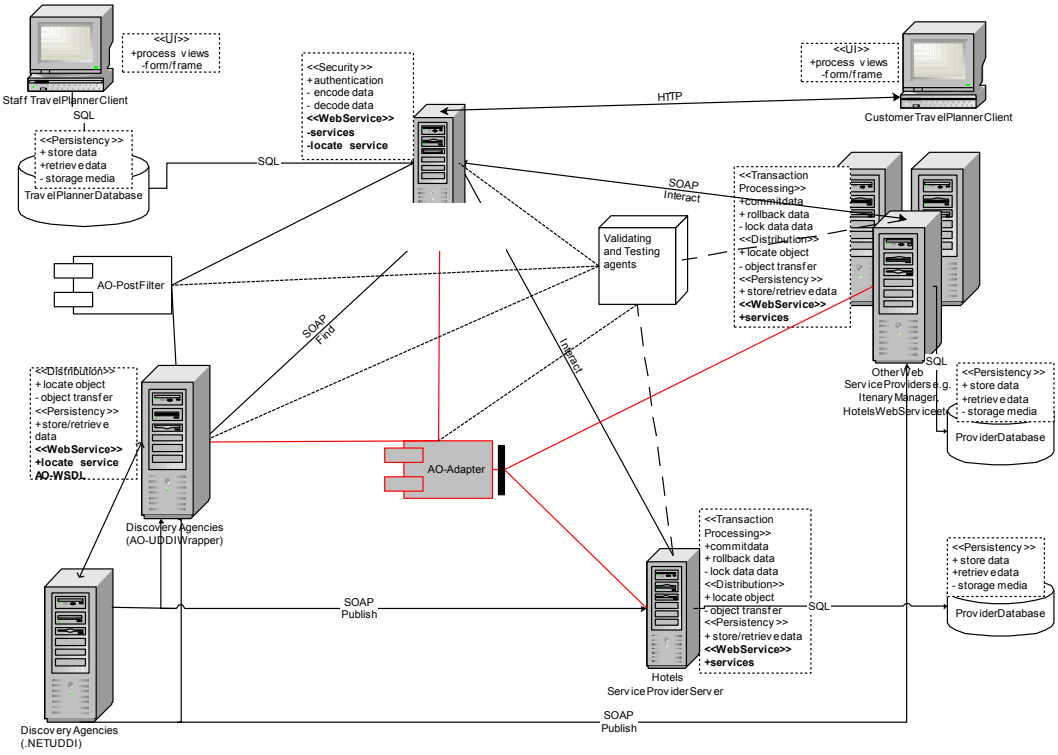


Figure 11.24: Aspects Validating Agent connected to the other subsystems in the web services system

An Aspects Validating Agent was designed and implemented to test and validate that the components deployed actually met the specification requirements of their aspects. This Aspects Validating Agent is shown in Figure 11.24 connected to the other subsystems in the web services system. The tool makes use of XML queries over the wire to test the web services performance and returns the results which are displayed on its user interface. It can be programmed to do any number tests remotely on the

collaborative travel planner system to verify whether the aspects and components that make up the system are functioning according to its specifications or not. The web services of the collaborative Travel planner application can be in use simultaneously by other users while the validating agent is running the tests.

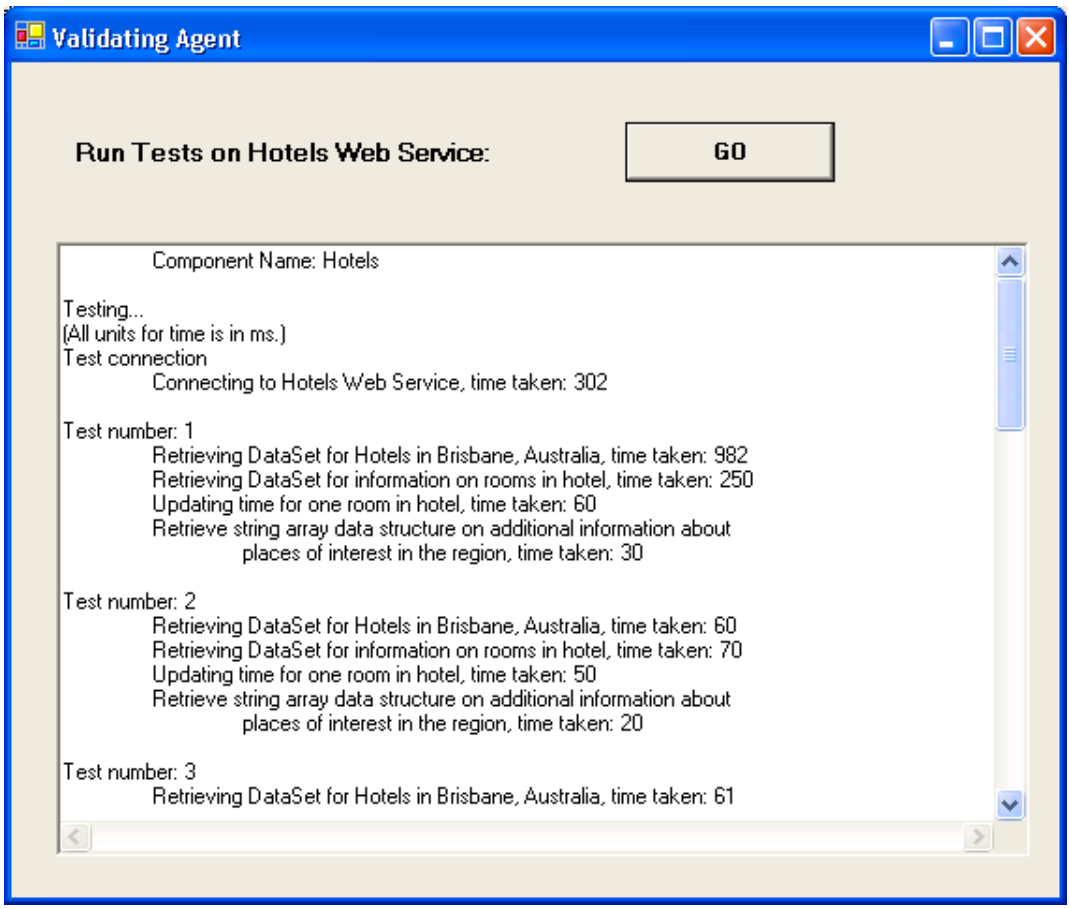


Figure 11.25: Sample output from the validating agent running tests on the Hotels web service.

A sample of the output produced by the validating agent running tests on the Hotels web service is depicted in Figure 11.25. It was noticed that a longer duration was

needed to execute an aspectual-operation the first time an access to any table in the DataBase was made. This was because objects have to be created to facilitate the process of making connections initially. Any subsequent access can reuse these objects and as such requires less time to execute the commands. For example, referring to Test Number 1 in Figure 11.25, it can be seen that the execution of the component aspect to retrieve a Dataset containing hotels information for the first time required 982 ms. But any subsequent retrieval based on executing this component aspect in the other tests requires approximately only 60 ms as shown.

Tests on applications built without AOCE showed similar behaviour, i.e. needing more time to execute the first data access operation, showing that this initial extra time required is not peculiar to aspect-oriented applications. Furthermore both the initial and subsequent values shown in Figure 11.25 are well within the allowable limits of performance as required in the specifications of the Travel Planner application.

These component aspects testing results will give component developers a good understanding of component systemic crosscutting aspects information, from which developers will benefit to improve components service qualities. The aspect-oriented travel planner system produced was also found to be quite robust as subjecting it to repeated testing and "trashing" didn't cause it or the server involved to crash.

11.3 Summary

Through experience and in line with our view to maintain consistency to enable increased understanding, consistency and coherency in our AOWS designs and code implemented using AOCE, we designed and implemented all our aspect-oriented components using the same consistent terminology and style. We observed that this also made any subsequent refactoring, scaling or maintenance of any part of the program easier, faster and less stressful, and the evaluation shows that this is indeed true. Even the agents used in our system were implemented using aspects contained in aspect-oriented components so that they can be located, accessed, modified and tested more easily and efficiently. Line by line code checks, black box testing and validation tests by testing agents showed that our prototype works well according to the specifications for our collaborative Travel Planner application. Furthermore the evaluation carried out by subjects who had specialised knowledge about web services and AOCE concurred that our AOWS components are more reusable and understandable. The evaluation also showed that the AOWSDL and AOUDDI is more useful and comprehensive as compared to the normal WSDL and UDDI that does not support aspects and aspects-oriented components framework.

12 Conclusions and Future Work

We have shown that Aspect-Oriented Component Engineering can be successfully applied to develop complete Aspect-Oriented Web Services systems (AOWS) composed of highly efficient and reusable aspect-oriented components and services that are better characterised, categorised and modularised when compared to non-AOWS systems. AOCE for web services brought about a definite enhancement in reusability and reconfigurability of the software system as a whole and increased understandability during the whole software development cycle. This is also true for any subsequent maintenance or refactoring of the software system. As can be seen from our work on AOWS, Aspect-Oriented Component Engineering provides a new and very beneficial framework for describing and reasoning about component capabilities from multiple aspect-oriented perspectives. This valuable and necessary aspect information in component implementations will allow developers, end users and other components of the web services to access high level knowledge and utilise these more thorough and comprehensive descriptions about component capabilities.

We also extended and enriched the web service description language into AOWSDL documents to support and utilise the rich aspect-oriented features. These extended descriptions in the AOWSDL can be indexed by using AOUDDI registries and the aspect information used to assist better description, discovery, testing and integration of aspect-oriented web service components.

Adaptors and Aspect-Oriented Composites could be dynamically located using the AOConnector and AOUDDI subsystems to combine multiple aspect oriented web service providers to satisfy the requirements of the web service clients requesting such services. Validating agents were also successfully implemented to carry out validity tests on the web services system, including on located web service providers to verify that they conform to the specifications required by the web service requestors.

We successfully designed and developed a CASE tool, called the AOWSCreator that can be used to support the analysis, design and development of AOWS based systems using the AOCE methodology. This tool allows us to better depict and manipulate aspects, aspect-oriented components and other entities in the AOWS designs and imlpementations. The AOWSCreator can also be used to generate C# code for the aspect-oriented components/subsystems of the AOWS applications and output the XML based AOWSDL documents of all the aspect-oriented service providers of the system. This novel tool made the development cycle of designing and creating large and complex AOWS-based systems easier to visualise, manage and control, thus increasing efficiency and effectiveness during its development process using AOCE.

We had also designed and developed another novel breed of Intelligent Aspect-Oriented Web Services systems (or IAOWS for short) that can support more automation in the area of dynamic discovery, integration and subsequent consumption by clients through the use of Multi-Agents and aspectual features. It allows us to come nearer at realizing the dream that web services can indeed cater for more extensive dynamic application to application communication without human

intervention. The Multi-Agents deployed here not only addressed the issues that hampered dynamic look-up and integration of web-service based systems, they also made such systems more modular, maintainable, reusable and scalable.

The model of our Aspect-Oriented Web Service based system and its abstractions were captured and analysed using Alloy to show that AOWS is formally and logically correct and that the design and implementation of the AOWS are also correspondingly correct.

As such the combination of the AOWSDL and AOUDDI used in conjunction with the rest of the AOWS subsystems including the connectors, adaptors, validating agents, and web service providers and requesters that were developed using AOCE techniques supports richer, clearer and more efficient and superior web services systems as compared to those built without using this technique. We had used a collaborative Travel Planner application as an example to describe how AOCE can be applied to the design, characterisation, location and integration of web service-based software components. It is our conviction that these aspect-oriented approaches could equally well have been applied to any other types of web services systems of any complexity besides a collaborative Travel Planner application to support the web services systems specification through the use of Aspect Oriented Component Engineering as described in this thesis.

Future Research Work

We developed, refined and applied the AOCE methodology to web services development so as to characterise and categorise components' high-level perspective information by specifying both functional and non-functional aspect services. This methodology was found to have worked very well at developing more reusable, efficient and understandable components for web services systems.

In our research we discovered that the following areas for future research are both required and necessary as they promise practical solutions to better understand and use the AOCE technology more efficiently:

- Participate in the open-standard UDDI development project by contributing ideas and expertise to introduce and support richer aspect-oriented features in the UDDI. In addition to its existing functions, the UDDI should also be able to function similar to an AOUDDI that can cater for systems that support aspect-oriented styled web services.
- Tool support to generate code for other popular programming languages besides C# (for instance Java, Visual Basic, C, C++, etc.). This is with the objective that people well versed in those languages will also find the tool very beneficial, i.e. not just for analysis and design purposes only but also for code generation with a choice in a number of languages.
- Further develop tool support for aspect-oriented web services development that will include both design tools having an extended UML modelling approach and implementation tools using an extended development environment which generates AOWSDL descriptions of web service

component implementations. These tools should be made pluggable into existing IDEs like Visual Studio .NET and Eclipse to cater for a wider audience.

- Replacement of one aspect-oriented component with another can be a time-consuming and an arduous task since the new component will never be identical to its predecessor and must be thoroughly tested, both in isolation and in combination with the rest of the system. The system/component may also need to be "tweaked" to accommodate the new component if changes are made to its interface. A more efficient autonomous approach and strategy is required here so that this task or reuse/refactoring can be carried out more easily.
- Standardize the use of notations and terminology for all types of AOCE designs and implementations. This will streamline the use of AOCE notations and terminology so that the various AOCE developers will be able to understand the process and programs faster and better without wasting time learning what they represent in different systems/platforms. Standardizing these features will also appeal to a wider community of developers.
- Apply Aspect-oriented Components Engineering for work involving the use of intelligent agents in AOConnectors, (including extending their use in AOAdaptors, AOComposites and postfilters linked to the connector object). The AOConnectors and intelligent agents will allow us to build more light weight clients through the use of loose coupling and Inversion of Control mechanisms and at the same time also help with automatic dynamic lookup, interpretation, translation, composition and integration involving the web services.

- Research into methods to efficiently re-engineer existing non-AOCE web service based software applications using AOCE techniques. This has the aim of making the existing software more reusable, understandable, easier to refactor and maintain through using the highly efficient and characterised aspect-oriented components.
- Contribute to WSDL development by undertaking further research, providing ideas and technical expertise on how to include and use rich aspect-oriented features in WSDL documents. This will give an option to web service requesters and providers whether they want to take advantage of the aspect-oriented features that are so useful for better description, discovery and integration of web services. WSDL developers have shown interest in exchanging ideas based on our work that may lead to the introduction and handling of aspects, components and their descriptors in service documents.

Refereed International Conference papers during PhD study

(1.) Grundy J, Panas T, Singh S, Stöckle H., 'An Approach to Developing Web Services with Aspect-oriented Component Engineering' NCWS03, Växjö, Sweden, 20th -21st November, 2003.

(2.) Singh, S., Grundy, J., Hosking, J., 'Developing .NET Web Service-based Applications with Aspect-Oriented Component Engineering', AWSA'04, Australia, 13th – 16th April 2004.

(3.) Singh, S., Hosking, J., Grundy, J., 'Deploying Multi-Agents for Intelligent Aspect-Oriented Web Services', Pacific Rim International Workshop on Multi-Agents (PRIMA 05), 8th Pacific Rim International Workshop on Multi-Agents, Kuala Lumpur, Malaysia, 26th – 28th September 2005.

(4.) Singh, S., Grundy, J., Hosking, J., Sun, J., 'An Architecture for Developing Aspect-Oriented Web Services', European Conference on Web Services (ECOWS 05), 3rd IEEE European Conference on Web Services, Växjö, Sweden, 14th -16th November 2005.

(5.) Singh, S., Chen, H.C. ,Hunter, O., Grundy, J., Hosking, J., 'Improving Agile Software Development using eXtreme AOCE and Aspect-Oriented CVS', Asia-Pacific Software Engineering Conference (APSEC), 12th IEEE Asia-Pacific Software Engineering Conference, Taipeh, Taiwan, 15th -17th December 2005.

References

Adams and Boeyen, 02

Adams, C., Boeyen, S. "UDDI and WSDL extensions for Web Services: a Security Framework", In Proc. 2002 ACM workshop on XML security, Fairfax, VA, 2002.

Allen and Frost, 98

Allen, P., and Frost, S. "Component-Based Development for Enterprise Systems: Applying the Select Perspective", Addison-Wesley, 1998.

Alloy_homepage, 05

Alloy homepage http://alloy.mit.edu

Alloy_tutorial, 05

Alloy tutorial web page:

http://web.mit.edu/~rseater/www/tutorial3/alloy-tutorial.html

Alur et al, 03

Alur, D., Malks, D., Crupi, J., "Core J2EE Patterns: Best Practices and Design Strategies", Second Edition, Sun Microsystems Press 2003

Arkin, A et al, 02

Arkin, A et al, "Web Service Choreography Interface (WSCI) 1.0" http://www.w3.org/TR/wsci/

W3C Note 8 August 2002

AspectJ_homepage, 05

http://www.parc.com/research/projects/aspectj/default.html

AspectWerkz homepage, 05

http://aspectwerkz.codehaus.org/index.html

Ballinger 03,

Ballinger, K., ".NET Web Services: Architecture and Implementation", Addison-Wesley, 2003.

Bennet et al, 99

Bennet, S., McRobb, S., Farmer, R., "Object-Oriented Systems Analysis and Design using UML" McGraw Hill 1999

Booth et al, 04,

Booth, D., et al, "Web Services Architecture" W3C

http://www.w3.org/TR/ws-arch/

2004

Brown and Wallnau, 96

Brown, A., W., Wallnau, K., C. "Engineering of Component-Based Systems," 7-15. Component-Based Software Engineering: Selected Papers from the

Software Engineering Institute. Los Alamitos, CA: IEEE Computer Society Press, 1996.

Brown and Wallnau, 98

Brown, A., W., Wallnau, K., C. “The Current State of CBSE” IEEE Software, Volume 155, Sept-Oct 1998.

Cai et all, 00

Cai, X., Michael R., Lyu, Wong, K., F., Ko, R., “Component-based software Engineering: Technologies, Development Frameworks and Quality Assurance Schemes” IEEE 2000.

Cerami, 02

Cerami, E., “Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL” February 2002, O'Reilly

Chappell and Jewell, 02

Chappell, D., Jewell, T., “Java Web Services” March 2002, O'Reilly

Christensen et al, 01

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., “Web Services Description Language (WSDL)” W3C http://www.w3.org/TR/wsdl, 2001

Cibran et al, 04

Cibrán, M., A., Verheecke, B., Suvée, D., Vanderperren, W., Jonckers, V., "Automatic Service Discovery and Integration using Semantic Descriptions in the Web Services Management Layer", NCWS 2004.

Clark, 05

Clark, J., "Inside Indigo: Infrastructure for Web Services and Connected Applications", Microsoft Publishers 2005.

Colyer and Clement, 04

Colyer A., Clement, A., "Large-scale AOSD for Middleware", AOSD 04, ACM.

Dani et al, 05

Dani, A.R., Radha Krishna, P., Subramanian, V., "An electronic payment system architecture for composite payment transactions" In the EEE 2005. Proceedings.

Dhesiaseelan, 04,

Dhesiaseelan, A., Ragunathan, V., "Web Services Container Reference Architecture (WSCRA)". In Proceedings of the IEEE International Conference on Web Services (ICWS'04)

Ding, 02

Guoliang Ding, "Aspect-Oriented Component Engineering" Masters Thesis, University of Auckland, February 2002

Dong et al, 03

Dong, J.S., Sun, J., and Wang, H., "Checking and Reasoning about Semantic Web through Alloy", FME 2003, LNCS 2805.

D'Souza et al, 99

D'Souza, D., F., Wills, A., C., "Objects, Components and Frameworks with UML, The Catalysis™ Appproach" 1999, Addison Wesley Longman Inc.

Eclipse-AspectJ_homepage, 05

http://eclipse.org/aspectj/

Eclipse_homepage, 05

http://www.eclipse.org/

Elrad et al, 01

Elrad, T., Aksits, M., Kiczales, G., Lieberherr, K., Ossher, H., "Discussing aspects of AOP", Communications of the ACM, Volume 44 , Issue 10 (October 2001), Pages: 33 - 38

Evjen, 02

Evjen, B., "XML Web Services for ASP.NET" 2002, Wiley Publishers.

Ferrara and MacDonald, 02

Ferrara, A., MacDonald, M.., “Programming .NET Web Services” September 2002, O’Reillly

Foggon, 04

Foggon, D., Maharry, D., Ullman, C., Watson K., “Programming Microsoft .NET XML Web Services (Pro-Developer)”, Microsoft Press, 2004 Microsoft Corporation

Fowler, 04

Fowler, M., “Inversion of Control Containers and the Dependency Injection pattern”, 2004

http://www.martinfowler.com/articles/injection.html

Gailey, 04

Gailey, J., H., “Understanding Web Services Specifications and the WSE” Microsoft Press, 2004 Microsoft Corporation

Gannod and Bhatia, 04

Gannod, C., Bhatia, S. “Facilitating Automated Search for Web Services”, In Proc. IEEE International Conference on Web Services, ICWS’04, IEEE.

Grundy, 00

Grundy, J., "Multi Perspective Specification, Design and Implementation of Software Components using Aspects" *International Journal of Software Engineering and Knowledge Engineering*. Vol. 10, No. 6 (2000) 713-734 World Scientific Publishing Company

Grundy and Ding, 02

Grundy, J., Ding, G., "Automatic Validation of Deployed J2EE Components Using Aspects" Department of Computer Science, University of Auckland. john-g@cs.auckland.ac.nz, 2002

Grundy and Hosking, 02

Grundy J.C., and Hosking J.G., "Engineering plug-in software components to support collaborative work." Software – *Practice and Experience* 2002; **32**:983-1013 (DOI: 10.1002/spe.472)

Grundy and Patel, 01

Grundy, J., Patel, R.,. "Developing Software Components with UML, Enterprise Java Beans and Aspects" Proceedings of the 2001 Australian Software Engineering Conference, Canberra, Australia 26-28 August 2001, IEEE CS Press.

Grundy et al 98

Grundy, J.C. Mugridge, W.B. Hosking, J.G. and Apperley,M.D. "Tool Integration, Collaboration and User Interaction Issues in Component-based

Software Architectures" Proc. of TOOLS Pacific 1998. Melbourne, Australia (24-26 November 1998), IEEE CS Press.

Grundy et al, 03

Grundy J, Panas T, Singh S, Stöckle H. "An Approach to Developing Web Services with Aspect-oriented Component Engineering" NCWS03, 2003

Hannemann and Kiczales, 02

Hannemann, J., Kiczales, G., "Design Pattern Implementation in Java and AspectJ", OOPSLA '02, November 4-8, 2002, Seattle, Washington, USA.

Hausmann, 04

Hausmann, J.H.H., Heckel, R., Lohmann, M., "Model-based Discovery of Web Services", In Proc. ICWS'04.

Haines et al, 97

Haines, G., Carney, D., Foreman, J., "Component-Based Software Development / COTS Integration" Website:http://www.sei.cmu.edu/str/descriptions/cbsd.html, 2005

Heisell et al, 02

Heisel1, M., Santen, T., Souqui`eres, J., "Toward a Formal Model of Software Components" Formal Methods and Software Engineering, 4th International Conference on Formal Engineering Methods, ICFEM 2002. Proceedings

Heuvel and Maamar, 03

Heuvel, W., V., D., Maamar, Z,. "Moving toward a framework to compose Intelligent Web Services", Communications of the ACM, Volume 46 Issue 10 Publisher: ACM Press, October 2003

HiveMind_webpage, 05

http://jakarta.apache.org/hivemind/ioc.html

Jackson, 02

Jackson, D., "Alloy: A Lightweight Object Modelling Notation". ACM Transactions on Software Engineering and Methodology, 2002.

Jackson MITLab, 02

Jackson, D., "A micromodels of software: Lightweight modelling and analysis with Alloy". MIT Laboratory for Computer Science, Cambridge, MA, 2002.

Jackson et al, 02

Jackson, D., Shlyakhter, I.,Sridharan., M., "A micromodularity mechanism". In Proceedings of the 8th European software engineering conference held together with the 9th ACM SIGSOFT international symposium on Foundations of software engineering, pages 62–73, Vienna, Austria, 2001, ACM Press.

Jamsa, 03

Jamsa, K., ".NET Web Services Solutions" 2003 SYBEX Inc.

JBoss_homepage, 05

http://www.jboss.com/developers/index

Katara, 03

Katara, M., Katz, S., Architectural Views of Aspects*, In Proc. AOSD 2003, Boston, MA USA, ACM 2003.

Kiczales et al, 97

Kiczales, G., et al, Aspect-oriented Programming, in *Proc. of the 1997 European Conf. on Object-Oriented Programming (ECOOP)*, Finland (June 1997), Springer-Verlag, LNCS 124

Kiczales et al, 05

Kiczales, G., et al, "Crosscutting Objects for Better Modularity" Sponsored by Palo Alto Research Center, Defense Advanced Research Projects Agency and NIST Advanced Technology Program. Affiliated to University of British Columbia Software Engineering Research Group and University of California San Diego Software Evolution Group http://aspectj.org/servlets/AJSite

Kopecký and Parsia, 05

Kopecký, J. and Parsia, B., "Web Services Description Language (WSDL) Version 2.0: RDF Mapping" W3C 2005

http://www.w3.org/TR/wsdl20-rdf/

Kvale et al, 05

Kvale, A., A., Li, J., Conradi, R., "A case study on building COTS-based system using Aspect-Oriented Programming", in Proceedings of the 2005 ACM Symposium on Applied Computing, Santa Fe, New Mexico

Language list, 05

Language list: http://home.nvg.org/~sk/lang/lang.html

Lee et al, 99

Lee, S., D., Yang, Y., J., Cho, E., S., Kim, S., D., Rhew, S., Y., "COMO: A UML-Based Component Development Methodology", Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia pacific, 1999 Pages: 54 – 61.

Lieberherr, 99

Karl Lieberherr "Connections between Demeter/Adaptive Programming and Aspect-Oriented Programming (AOP)" http://www.ccs.neu.edu/home/lieber/connection-to-aop.html 1999

Lieberherr et al, 99

Karl Lieberherr, David Lorenz, Mira Mezini, “Programming with Aspectual Components”, 1999.

Liu and Liu, 02

Liu, J., Liu, C., “A declarative way of extracting XML data in XSL” Advances in Databases and Information Systems. 6th East European Conference, ADBIS 2002. Proceedings. Bratislava, Slovakia. 8-11 Sept. 2002

Mathew, 05

Mathew, S., “Examining the Validity of Inversion of Control”, Feb 2005 http://www.theserverside.com/articles/content/IOCandEJB/article.html

McKie, 02

McKie, S., “Ready, set, compete: Web services promise to fill gaps in strategic business apps for the purpose of collaborative commerce. How can you take advantage?.” Intelligent Enterprise, March 8, 2002 v5 i5 p25(5)

Mei, 04

Mei, H,, “ABC: Supporting Software Architectures in the Whole Lifecycle”, Proceedings of the Second International Conference on Software Engineering and Formal Methods (SEFM’04), IEEE

Microsoft .NET website, 05

Microsoft, Visual Studio and .NET

http://www.microsoft.com/homepage/ms.htm03 Microsoft Corporation.

http://www.microsoft.com/net/ http://msdn.microsoft.com/vstudio/

2003 Microsoft Corporation

MicrosoftsUBR, 05

Microsoft's UDDI Business Registry (UBR) node 2005 homepage: http://uddi.microsoft.com/default.aspx

Mockford, 04

Mockford K, "Web services architecture", BT Technology Journal, vol.22, no.1, Jan. 2004, pp.19-26. Publisher: British Telecommunications plc, UK.

Moskewicz et al, 01,

Moskewicz, M. W., Madigan, C. F.., Zhao, Y., Zhang, L. and Malik, S. Chaff: "Engineering an efficient SAT solver". In J. Rabaey, editor, Proc. 38th conference on Design automation, pp. 530–535, Las Vegas, Nevada, 2001, ACM.

Nagano et al, 04,

Nagano, S., Hasegawa, T., Ohsuga, A., "Dynamic Invocation Model of Web Services Using Subsumption Relations". Proceedings of the IEEE International Conference on Web Services (ICWS'04)

Newcomer, 02

Eric Newcomer, E., “Understanding Web Services: XML, WSDL, SOAP, and UDDI”, Addison-Wesley 2002

Pallmann, 05

Pallmann, D., “Programming Indigo: The Code Name for the Unified Framework for Building Service-Oriented Applications on the Microsoft Windows Platform”, Microsoft Publishers 2005.

Panas et al, 03

Panas, T., Karlsson, J. and Högberg, M. “Aspect-jEdit for Inline Aspect Support” Proceedings of the 3rd German Workshop on Aspect Oriented Software Development, Technical Report of the University of Essen, March 2003

PicoContainer_homepage_05

http://www.picocontainer.org/

Ran, 03

Ran, S., “A model for web services discovery with QoS” ACM SIGecom Exchanges , Volume 4 , Issue 1 Spring, 2003

Rashid and Ghitchyan, 03

Rashid, A., Ghitchyan, R., “Persistence as an Aspect”

AOSD 2003 Boston, MA USA

Rashid et al, 03

Rashid, A., Moreira, A., Araojo, J., "Modularisation and Composition of Aspectual Requirements" AOSD 2003 Boston, MA USA Copyright ACM 2003 1-58113-660 -9/03/002.

Rational_Rose_homepage, 05

http://www.rational.com/products/rose/index.jtmpl

Safonov and Grigoryev, 05

Safonov V.O., Grigoryev D.A., "Aspect.NET – an aspect-oriented programming tool for Microsoft.NET" – Proceedings of St. Petersburg Regional IEEE conference, St. Petersburg, May 2005.

SAPsUDDI, 05

SAP UDDI Business Registry 2005 homepage: http://uddi.sap.com/

Shen et al, 05

Shen, J., Weber, I., Lesser, V., "OAR: A Formal Framework for Multi-Agent Negotiation, American Association for Artificial Intelligence", AAAI 2005

Shukla et al, 02

Shukla, D., Fell, S., Sells, C., "Aspect-Oriented Programming Enables Better Code Encapsulation and Reuse", MSDN Magazine 02, http://msdn.microsoft.com/msdnmag/issues/02/03/aop/

Siegel 02,

Siegel, J., "Using OMG's Model Driven Architecture (MDA) to Integrate Web Services" by Jon Siegel
Vice President, Technology Transfer
Object Management Group
http://www.omg.org/mda/mda_files/MDA-WS-integrate-WP.pdf
2002

Singh et al 04,

Singh, S., Grundy, J., Hosking, J., "Developing .NET Web Service-based Applications with Aspect-Oriented Component Engineering", AWSA'04, Australia, 13th – 16th April 2004.

Singh et al 05,

Singh, S., Hosking, J., Grundy, J., "Deploying Multi-Agents for Intelligent Aspect-Oriented Web Services", Pacific Rim International Workshop on Multi-Agents (PRIMA 05), 8th Pacific Rim International Workshop on Multi-Agents, Kuala Lumpur, Malaysia, 26th – 28th September 2005.

Singh et al 05,

Singh, S., Grundy, J., Hosking, J., Sun, J., "An Architecture for Developing Aspect-Oriented Web Services", European Conference on Web Services (ECOWS 05), 3rd IEEE European Conference on Web Services, Växjö, Sweden, 14th -16th November 2005.

Singh et al 05,

Singh, S., Chen, H.C. ,Hunter, O., Grundy, J., Hosking, J., "Improving Agile Software Development using eXtreme AOCE and Aspect-Oriented CVS", Asia-Pacific Software Engineering Conference (APSEC), 12th IEEE Asia-Pacific Software Engineering Conference, Taipeh, Taiwan, 15th -17th December 2005.

Spring_framework_homepage, 05

http://www.springframework.org

http://www.springframework.org/docs/reference/aop.html

Stearns and Piccinelli, 02

Stearns, M., Piccinelli, G., Managing Interaction Concerns in Web-Service Systems, Proc. 22nd Int. Conf. on Distributed Computing Systems Workshops, 2002 pp. 424 - 429.

Strahl, 01

Strahl, R., http://www.west-wind.com/presentations/dotnetwebservices/DotNetWebServicesData.asp

Passing data over .Net Web Services

by Rick Strahl

West Wind Technologies

Sun Microsystems website, 05

The Source for Java Technology and J2EE

http://java.sun.com/

http://java.sun.com/j2ee/

Sun Microsystems, Inc

Copyright © 1995-2005

Sycara, 98

Sycara, K.P., Multiagent Systems, 1998, American Association for Artificial Intelligence.

Topcoder_homepage, 05

http://www.topcoder.com/index?t=development&c=comp_meth

Torkelson et al 02

Torkelson, L., Petersen, C., Torkelson, Z., "Programming the Web with Visual Basic .NET" SoftMedia Artisans Inc 2002 Apress

UDDI_website, 05

Universal Description, Discovery and Integration of Web Services

http://www.uddi.org/

http://uddi.org/specification.html

http://www.oasis-open.org/

Visio_homepage, 05

http://www.mvps.org/visio/

Vitharana et al, 03,

Vitharana, P., Mariam, F., and Jain, H., “Design, Retrieval, And Assembly in Component-based Software Development”, CACM, vol. 46, no. 11, Nov. 2003.

w3schools_soap_example website, 05

http://www.w3schools.com/soap/soap_example.asp

Wiedemann, 02

Wiedemann, M. Web Services and collaborative commerce. Information Management & Consulting, vol.17, no.3, Aug. 2002, pp.57-60.

Zhang et al, 02

Zhang, L.J. Li, H., Chang, H., Chao, T. XML-based advanced UDDI search mechanism for B2B integration, Proceedings of the Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, IEEE CS Press, 2002, pp.9-16.

Zhu et al, 04

Zhu, N., Grundy, J., Hosking, J., “Pounamu: A Meta-Tool for Multi-View Visual Language Environment Construction”, Visual Languages and Human Centric Computing, 2004 IEEE Symposium on 26-29 Sept. 2004 Page(s):254 - 256

14. Appendixes

Appendix A

AOWSDL Schema

Figure 14.1 below shows the XML schema for the Aspect-Oriented Web Services Description Language (AOWSDL) document of our Aspect-Oriented Web Services providers detailing the major element called the "AOComponents" element that is imported into the WSDL document to enrich it with aspects and their details. It is published in the Aspect-Oriented Universal Description, Discovery and Integration (AOUDDI) registry.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://tempuri.org/" xmlns="http://schemas.xmlsoap.org/wsdl/"
 elementFormDefault="qualified">
   <xs:element name="AOComponents">
     <xs:attribute name="Name" type="xs:string" use="required" />
     <xs:element name="AODocumentation">
       <xs:attribute name="Information" type="xs:string" use="required" />
     </xs:element>
     <xs:element name="WSDescription">
       <xs:attribute name="Description" type="xs:string" use="required" />
     </xs:element>
     <xs:element name="Component" type="xs:string" use="required">
       <xs:attribute name="ComponentName" type="xs:string" use="required" />
       <xs:element name="ComponentDescription">
         <xs:attribute name="Description" type="xs:string" use="required" />
       </xs:element>
        <xs:element name="Aspects">
          <xs:element name="FunctionalAspects">
            <xs:element name="Aspect">
              <xs:sequence>
                <xs:attribute name="Type" type="xs:string" use="required" />
                <xs:attribute name="AspectName" type="xs:string" use="required" />
                <xs:attribute name="WSEntryPoint" type="xs:string" use="required" />
                <xs:attribute name="Standalone" type="xs:string" use="required" />
                <xs:element name="AspectDescription">
                   <xs:attribute name="Description" type="xs:string" use="required" />
                </xs:element>
                <xs:element name="Parameters">
                   <xs:element name="Parameter">
                      <xs:sequence>
                           <xs:attribute name="ParameterName" type="xs:string"
use="required" />
                           <xs:attribute name="ParameterType" type="xs:string"
use="required" />
                      </xs:sequence>
                   </xs:element>
                </xs:element>
                <xs:element name="Return">
                   <xs:attribute name="ReturnType" type="xs:string" use="required" />
```

```
                    </xs:element>
                    <xs:element name="AspectDetail">
                       <xs:sequence>
                          <xs:attribute name="Type" type="xs:string" use="required" />
                          <xs:attribute name="Detail" type="xs:string" use="required" />
                          <xs:attribute name="Provided" type="xs:string" use="required" />
                       </xs:sequence>
                    </xs:element>
                    <xs:element name="AspectUserOperations">
                       <xs:attribute name="UsedBy" type="xs:string" use="required" />
                    </xs:element>
                    <xs:element name="UsesOperations">
                        <xs:attribute name="Uses" type="xs:string" use="required" />
                    </xs:element>
                 </xs:sequence>
              </xs:element>
           </xs:element>
            <xs:element name="NonFunctionalAspects">
               <xs:element name="Aspect">
                  <xs:sequence>
                    <xs:attribute name="Type" type="xs:string" use="required" />
                       <xs:attribute name="AspectName" type="xs:string" use="required" />
                          <xs:element name="AspectDescription">
                             <xs:attribute name="Description" type="xs:string" use="required" />
                          </xs:element>
                          <xs:element name="Parameters">
                              <xs:element name="Parameter">
                                 <xs:sequence>
                                     <xs:attribute name="ParameterName" type="xs:string"
use="required" />
                                     <xs:attribute name="ParameterType" type="xs:string"
use="required" />
                                 </xs:sequence>
                              </xs:element>
                          </xs:element>
                          <xs:element name="Return">
                             <xs:sequence>
                                  <xs:attribute name="ReturnType" type="xs:string"
use="required" />
                                  <xs:attribute name="Units" type="xs:string" use="required" />
                             </xs:sequence>
                          </xs:element>
                          <xs:element name="AspectDetail">
                             <xs:sequence>
                                  <xs:attribute name="Type" type="xs:string" use="required" />
                                  <xs:attribute name="Detail" type="xs:string" use="required" />
                                  <xs:attribute name="Value" type="xs:string" use="required" />
                                  <xs:attribute name="ValueQualifier" type="xs:string"
use="required" />
                                  <xs:attribute name="Provided" type="xs:string" use="required"
/>
                             </xs:sequence>
                          </xs:element>
                       </xs:sequence>
                    </xs:element>
                 </xs:element>
              </xs:element>
           </xs:element>
        </xs:element>
     </xs:element>
  </xs:element>
</xs:schema>
```

Figure 14.1: AOWSDL schema for describing aspect-oriented web services.

As shown above, all the aspect-oriented elements and descriptors are enclosed within a major “AOComponents” element and added to the standard WSDL document as shown in the schema. The inclusion of these aspectual elements transformed the WSDL into Aspect-Oriented Web Service Description Language, or AOWSDL for short. The purpose of this is to enable the description and capture of the rich and highly characterised aspectual features of web services in a systematic manner.

Complete documentation for human consumption about the web service’s aspects and components is placed within the “AODocumentation” element. This documentation also includes high level instructions to software developers about the web service and how to access and consume it.

Another element, the “WSDescription”, gives crisp instructions that are machine understandable and is used for automatic discovery and integration of the web service. Web service requestors first access this “WSDescription” element and match its description with their requirements.

Each and every component of the web service provider is nested within the “AOComponents” element as shown. These components contain all the aspects that are exposed to the clients. The clients can make further XML queries to verify whether or not their detailed needs match those provided by the components and their

aspects. These descriptions are written in clear and concise language to allow for automatic querying and discovery.

Each component exposes one or more aspects. Each aspect element contains the full details about all its cross-cutting features. It is specified as a functional or non-functional type of aspect. It also has an aspect type associated with it, e.g. the aspect type could be Persistency, Distribution, Transaction, Security etc.

If the aspect can be used without resorting to the use of another aspect first, i.e. there is no precondition that it need to be used subsequent to another aspect, its "WSEntryPoint" attribute is set to true in the AOWSDL. All the aspect descriptors shown are used to facilitate automation.

As shown in Figure 14.1, each aspect has one or more aspect details associated with it. If this aspect detail is provided, its "Provided" element is set to "true", if it is required from others, it is set to "false". This enables clients to understand the aspects in more detail and query whether it serves their needs or not. AO-WSDL as such supports better description, characterisation and categorisation of web services than plain WSDL without the aspectual support.

The Aspect-Oriented Web Services Description Language (AOWSDL) as such represents a contract between an aspect-oriented service provider and a service requestor. AOWSDL is also platform and language independent and is used to describe aspect oriented web services. Using AOWSDL a web services client can more easily discover and dynamically locate a web service and invoke any of its publicly available functions. AOWSDL also acts as a platform to automatically integrate the services provided with the requesting client.

Figure 14.2 below shows a complete section of the AOWSDL produced from the hotels web service. It shows regions enriched with aspects in the aspect-oriented components and conforms to the XML schema shown in Figure 14.1.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://tempuri.org/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:aowsdl="http://localhost/AOUDDIWebService/bin/aowsdlSchema.xml"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://tempuri.org/" xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        <s:schema elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
            <s:import namespace="http://www.w3.org/2001/XMLSchema" />
.....
    </types>


    <aowsdl:AOComponents Name="HotelsWebServiceComponents">
        <aowsdl:AODocumentation Information="Exposes aspects to find vacant rooms in
hotels, searches for hotels  based on city or country of interest. After finding rooms
reservations can be made to book the rooms concerned...  All human readable information go
here. This include instructions and high level documentation about the web service for human
consumption." />
        <aowsdl:WSDescription Description="To find, update, delete and insert reservations
or bookings for vacant hotel rooms" />
            <aowsdl:Component ComponentName="HotelsDataManagementComponent">
            <aowsdl:ComponentDescription Description="Component to find hotels in various
cities and countries including rooms availability" />
                <aowsdl:Aspects>
                    <aowsdl:FunctionalAspects>
                        <aowsdl:Aspect Type="Persistency"
AspectName="HotelsDataSetfromCityCountry" WSEntryPoint="true" Standalone="true">
                            <aowsdl:AspectDescription Description="To search for hotels based on
```

```
city or country query" />
                                <aowsdl:Parameters>
                                  <aowsdl:Parameter ParameterName="strCity"
ParameterType="string" />
                                  <aowsdl:Parameter ParameterName="strCountry"
ParameterType="string" />
                                </aowsdl:Parameters>
                                  <aowsdl:Return ReturnType="DataSet" />
                                  <aowsdl:AspectDetail Type="data retrieval" Detail="select"
Provided="true" />
                                  <aowsdl:AspectUserOperations
UsedBy="Persistence_HotelFinder|TransactionProcessing_ItenaryManager" />
                                  <aowsdl:UsesOperations
Uses="Persistence_roomsByHotelID|Persistence_OnSiteFacilities|Persistence_OffSiteFaci
lities|Persistence_placesOfInterest" />
                              </aowsdl:Aspect>
                              <aowsdl:Aspect Type="Security" AspectName="AuthenticateUser"
WSEntryPoint="true" Standalone="true">
                                <aowsdl:AspectDescription Description="To authenticate that it is a
correct password and login used" />
                                  <aowsdl:Parameters>
                                    <aowsdl:Parameter ParameterName="strLogin"
ParameterType="string" />
                                    <aowsdl:Parameter ParameterName="strPassword"
ParameterType="string" />
                                  </aowsdl:Parameters>
                                     <aowsdl:Return ReturnType="Boolean" />
                                     <aowsdl:AspectDetail Type="security access" Detail="access"
Provided="true" />
                                  <aowsdl:AspectUserOperations UsedBy="" />
                                  <aowsdl:UsesOperations Uses="" />
                              </aowsdl:Aspect>

                          </aowsdl:FunctionalAspects>
                           <aowsdl:NonFunctionalAspects>
                              <aowsdl:Aspect Type="Performance"
AspectName="insertCustomerInfoIntoRoom">
                                <aowsdl:AspectDescription Description="The time taken to perform the
specified number of insert operations in ms" />
                                <aowsdl:Parameters>
                                    <aowsdl:Parameter ParameterName="numOfInserts"
ParameterType="integer" />
                                </aowsdl:Parameters>
                                   <aowsdl:Return ReturnType="float" Units="ms" />
                                   <aowsdl:AspectDetail Type="performance speed in ms"
Detail="Speed" Value="230" ValueQualifier="lessthan" Provided="true" />
                              </aowsdl:Aspect>
                           </aowsdl:NonFunctionalAspects>
                      </aowsdl:Aspects>
                  </aowsdl:Component>
                   <aowsdl:Component ComponentName="PaymentComponent">
                        <aowsdl:ComponentDescription Description="Component to enable payment for
services by credit card" />
....
                  </aowsdl:Component>

....
            </aowsdl:AOComponents>
           <service name="HotelsWebService">
                <port name="HotelsWebServiceSoap" binding="s0:HotelsWebServiceSoap">
                  <soap:address
location="http://localhost/WebServiceHRHotels/HotelsWebService.asmx" />
                </port>
                <port name="HotelsWebServiceHttpGet" binding="s0:HotelsWebServiceHttpGet">
                    <http:address
location="http://localhost/WebServiceHRHotels/HotelsWebService.asmx" />
                </port>
                <port name="HotelsWebServiceHttpPost" binding="s0:HotelsWebServiceHttpPost">
                    <http:address
location="http://localhost/WebServiceHRHotels/HotelsWebService.asmx" />
```

```
        </port>
    </service>
..
</definitions>
```

Figure 14.2: A complete section of the AOWSDL produced from the hotels web service showing regions enriched with aspects and aspect-oriented components.

This AOWSDL document shown here is enriched with aspects and component aspect characterisations from the Aspect-Oriented Hotels Web Service provider. It has a major element called the HotelsWebServiceComponents XML element that encapsulates all the components, their aspects and details including the complete information and definition of each and every element to support more autonomous behaviour. The component's various properties, operations and event support are also defined in this figure. It is further characterised by the low-level data management features and high-level business support e.g. payment management. It also shows some of its required services and their characteristics e.g. its low-level persistency aspectual features; its medium-level transaction support approach and its high-level travel booking support. The components shown (i.e. Payment component, HotelsDataManagement component etc.) are just a few examples from the various components from the collaborative travel planner that we characterized and categorized using AOCE.

Appendix B: Use Case descriptions

The Use Case descriptions of the travel planner application are listed and described in the following tables.

Hotels system:

Book room	
Actor	Customer or staff using a web service client.
Precondition	User enters name of hotel and room number of a vacant room that he desires to book depending on the user's room preferences (e.g. facilities available, location, type of room – *single, double, superior, deluxe* etc.) and them clicks the "Book Room" button.
Postcondition	System displays the results of the booking operation on the screen. It states whether the booking was successful or not
Description	User books for vacant room of his choice from the hotels database.
Basic course of action	User sends a request to book the room of his choice that is shown to be vacant. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If the user is using the system for the first time, then the system will request the user to register first.

Table 13.1: Use case descriptions for "Book room"

Authenticate user	
Actor	Customer or staff wanting to use the web service systems' functionalities.
Precondition	User enters login name and password into web user interface.
Postcondition	System displays the results of the authentication operation. It states whether the authentication was successful or not.
Description	System verifies whether user is a valid user of the web service system.

Basic course of action	User sends a request containing his login name and password to web server. If successful the system displays welcome message and allows user to proceed with further actions.
Alternative courses of action	If user is a first-time user, then the system will request the user to register first by supplying his personal details like name, email address, gender, age etc. including a new password and login of the user's choice. Staff will have additional permissions and privileges that will be granted to them by the system's administrator so that they can do maintenance and editing functions on the relevant database entries to keep the records updated.

Table 13.2: Use case descriptions for "Authenticate user"

View Bookings	
Actor	Customer or staff using a web service client
Precondition	User clicks on the "View Bookings" button to view the customer's current bookings.
Postcondition	System displays the results of the "View Bookings" operation on the screen.
Description	User queries the database for all the customer's current bookings, if any.
Basic course of action	User sends a request view the customer's current bookings to the web service system. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If no bookings exist, the system will display that there are no bookings and direct user on how to make bookings for rooms.

Table 13.3: Use case descriptions for "View Bookings"

Edit Bookings	
Actor	Customer or staff using a web service client
Precondition	User clicks on the "View Bookings" button first and then the "Edit

	Bookings" button to edit the customer's current bookings.
Postcondition	System displays the results of the Edit Bookings operation on the screen.
Description	User edits the database containing the customer's current bookings, if any.
Basic course of action	User sends a request to edit the customer's current bookings to the web service system. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If no bookings done system will direct user to make a booking first before any editing can be carried out.

Table 13.4: Use case descriptions for "Edit Bookings"

Retrieve Room Information	
Actor	Customer using a web service client
Precondition	Customer enters name of hotel and room number of a room that he desires to retrieve information about.
Postcondition	System displays the results of the "Retrieve Room Information" operation on the screen. It also states whether the operation was successful or not
Description	Customer retrieves information about a particular room from the hotels database.
Basic course of action	Customer sends a request to retrieve information about a particular room of his choice. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If customer is a new customer, then the system will request the customer to register first.

Table 13.5: Use case descriptions for "Retrieve Room Information"

Make Payment	
Actor	Customer using the web service client
Precondition	Customer enters payment details including mode of payment for a room that he has booked and wishes to pay for.
Postcondition	System displays the results of the payment operation on the screen. It also states whether the payment made was successful or not.
Description	Customer makes payment for a room that he has booked..
Basic course of action	Customer sends a request to make payment for a room that he has booked. The result of the 'Make Payment' request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If customer is a new customer, then the system will request the customer to register first.

Table 13.6: Use case descriptions for "Make Payment"

View Customer Data	
Actor	Staff using a web service client
Precondition	Staff enters name and unique ID of customer and clicks "View Customer" button.
Postcondition	System displays the results of the search on the screen. It contains all the particulars about the customer.
Description	Staff searches for the details of a particular customer stored in the Customers database.
Basic course of action	Staff queries using customer's name and unique ID. The results of the query are returned in a dataset and are displayed on the screen.
Alternative courses of action	If customer is non existent the database, then the system will request that the customer be registered first.

Table 13.7: Use case descriptions for "View Customer Data"

Edit Customer Data	
Actor	Staff using a web service client

Precondition	Staff clicks on the "View Customer Data" button first and then the "Edit Customer Data" button to edit the customer details.
Postcondition	System displays the results of the "Edit Customer Data" operation on the screen.
Description	Staff edits the database containing Customer data, if any.
Basic course of action	Staff sends a request edit a particular Customer data to the system. The result of the request whether it is successful or not is returned and displayed on the screen.

Table 13.8: Use case descriptions for "Edit Customer Data"

Flights system:

View Flights	
Actor	Customer using a web service client.
Precondition	Customer enters departure date, departure place and name of city or country that he wishes to visit.
Postcondition	System displays the results of the search on the screen.
Description	Customer searches the information stored in the Flights database about the Flights he wishes to take to a particular country or city.
Basic course of action	Customer queries using departure date, departure place and name of a city or country where he wishes to visit. The results of the query are returned in a dataset and are displayed on the screen.

Table 13.9: Use case descriptions for "View Flights"

View Seats	
Actor	Customer using a web service client
Precondition	Customer enters the ID number of the Flight after doing a Flights search first. He then clicks the "View Seats" button.
Postcondition	System displays the results of the search on the screen, it contains the details of all the Seats in the particular Flight.

Description	Customer searches for the details of all the Seats stored in the Flights database about the Flight he wishes to take.
Basic course of action	Customer queries using the ID number of Flight after doing a Flights search first. The results of the query are returned in a dataset and displayed on the screen.
Alternative courses of action	1.) If customer is a new customer, then the system will request the customer to register first. 2.) If customer already knows the Flight's ID number he can query the web service about the Seats directly without performing a Flights search first. The results of the query are returned in a dataset and are displayed on the screen.

Table 13.10: Use case descriptions for "View Seats"

Book Seat	
Actor	Customer or staff using a web service client.
Precondition	User enters the ID number of the Flight and Seat number of a vacant Seat that he desires to book and clicks the "Book Seat" button.
Postcondition	System displays the results of the booking operation on the screen. It states whether the booking was successful or not
Description	User books for vacant Seat of his choice from the Flights database.
Basic course of action	User sends a request to book the Seat of his choice that is shown to be vacant. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If the user is using the system for the first time, then the system will request the user to register first.

Table 13.11: Use case descriptions for "Book Seat"

Authenticate user	
Actor	Customer or staff wanting to use the web service systems'

	functionalities.
Precondition	User enters login name and password into web user interface.
Postcondition	System displays the results of the authentication operation. It states whether the authentication was successful or not.
Description	System verifies whether user is a valid user of the web service system.
Basic course of action	User sends a request containing his login name and password to web server. If successful the system displays welcome message and allows user to proceed with further actions.
Alternative courses of action	If user is a first-time user, then the system will request the user to register first by supplying his personal details like name, email address, gender, age etc. including a new password and login of the user's choice.

Table 13.12: Use case descriptions for "Authenticate user"

View Bookings	
Actor	Customer or staff using a web service client
Precondition	User clicks on the "View Bookings" button to view the customer's current bookings.
Postcondition	System displays the results of the "View Bookings" operation on the screen.
Description	User queries the database for all the customer's current bookings, if any.
Basic course of action	User sends a request view the customer's current bookings to the web service system. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If no bookings exist, the system will display that there are no bookings and direct user on how to make bookings for Seats.

Table 13.13: Use case descriptions for "View Bookings"

Edit Bookings	
Actor	Customer or staff using a web service client
Precondition	User clicks on the "View Bookings" button first and then the "Edit Bookings" button to edit the customer's current bookings.
Postcondition	System displays the results of the Edit Bookings operation on the screen.
Description	User edits the database containing the customer's current bookings, if any.
Basic course of action	User sends a request to edit the customer's current bookings to the web service system. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If no bookings done system will direct user to make a booking first before any editing can be carried out.

Table 13.14: Use case descriptions for "Edit Bookings"

Make Payment	
Actor	Customer using the web service client
Precondition	Customer enters payment details including mode of payment for a seat that he has booked on the flight and wishes to pay for.
Postcondition	System displays the results of the payment operation on the screen. It also states whether the payment made was successful or not.
Description	Customer makes payment for a seat that he has booked.
Basic course of action	Customer sends a request to make payment for a seat that he has booked. The result of the 'Make Payment' request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If customer is a new customer, then the system will request the customer to register first.

Table 13.15: Use case descriptions for "Make Payment"

View Customer Data	
Actor	Staff using a web service client
Precondition	Staff enters name and unique ID of customer and clicks “View Customer” button.
Postcondition	System displays the results of the search on the screen. It contains all the particulars about the customer.
Description	Staff searches for the details of a particular customer stored in the Customers database.
Basic course of action	Staff queries using customer’s name and unique ID. The results of the query are returned in a dataset and are displayed on the screen.
Alternative courses of action	If customer is non existent the database, then the system will request that the customer be registered first.

Table 13.16: Use case descriptions for “View Customer Data”

Edit Customer Data	
Actor	Staff using a web service client
Precondition	Staff clicks on the “View Customer Data” button first and then the “Edit Customer Data” button to edit the customer details.
Postcondition	System displays the results of the “Edit Customer Data” operation on the screen.
Description	Staff edits the database containing Customer data, if any.
Basic course of action	Staff sends a request edit a particular Customer data to the system. The result of the request whether it is successful or not is returned and displayed on the screen.

Table 13.17: Use case descriptions for “Edit Customer Data”

Retrieve Seat Information	
Actor	Customer using a web service client
Precondition	Customer enters the ID number of the Flight and Seat number of a

	seat that he desires to retrieve information about.
Postcondition	System displays the results of the "Retrieve Seat Information" operation on the screen. It also states whether the operation was successful or not
Description	Customer retrieves information about a particular seat from the Flights database.
Basic course of action	Customer sends a request to retrieve information about a particular seat of his choice. The result of the request whether it is successful or not is returned and displayed on the screen.
Alternative courses of action	If customer is a new customer, then the system will request the customer to register first.

Table 13.18: Use case descriptions for "Retrieve Seat Information"

Use case event flow for "**Book room**".

1. **Used by:** customers via web service clients to make bookings of vacant rooms in a particular hotel.
2. **Event flows:**

2.1 Repeat until customer makes a booking of a vacant room or leaves the web service.

2.1.1 Customer enters name of hotel and room number of a room that is vacant that he desires to book.

2.1.2 Customer clicks a button on a requesting client to execute the request.

2.1.3 The result of the request whether it is successful or not is returned and displayed on the screen.

2.2 If server error – error message displayed. Go to 2.1.1

2.3 If room cannot be booked – error message displayed. Go to 2.1.1

3 **Related Actors/Use cases:** Used by Travel Planner client. Can only use this function after searching for vacant rooms in a particular hotel.

4 **Special conditions:** uses Web Services Technology and SOAP or HTTP protocol.

Must be used by a web service requestor.

Appendix C

The explanations of the Alloy program and constructs that were used to model, analyse and verify AOWS and its abstractions are given in this Appendix.

Part 1: Description of the Signatures used to model AOWebServices systems

The AOWS was modelled, analyzed and verified using the Alloy tool by creating simulations of the system. The simulations are created by first generating the main objects in the system as signatures. The signatures are also associated with facts, assertions and predicates so as to simulate any sort of relationships or functions for the system. These signatures are defined below together with their descriptions and significance to the system.

```
sig AOWebServiceProvider{
    aowsdl : AOWSDL
}
```

Shown above is the signature of an AOWebServiceProvider and it represents a service provider in our AOWS model. Each AOWebServiceProvider has an AOWSDL file associated with it whereby the AOWSDL file is an extension of a WSDL file that is enriched with aspect-oriented elements and components. The signature here is associated with an Alloy *Fact* that for all AOWSDL files to exist usefully they must first be registered and deposited in the AOUDDI Registry. This

will be later tested by the predicate *RegisterNewWebService(aowsdl:AOWSDL)*. The AOWSDL file is passed in as an argument and should be successfully registered and stored in the AOUDDI Repository.

```
sig AOWSDL{
    aoComponents : AOComponents
}
```

The above signature defines an AOWSDL file. It contains a field called AOComponents, which contains the set of AOComponent(s). Each AOComponent in turn contains all the information about the aspect-oriented web services that are exposed by the provider and the corresponding details about the aspects.

```
sig AOComponents{
    name : String,
    aoComponent : set AOComponent,
    aoDocumentation : AODocumentation,
    aoWSDescription : AOWSDescription
}
```

Depicted above is the signature of the AOComponents, it contains a name, a set of AOComponent(s), an AODocumentation, and an AOWSPDescription. The AODocumentation and AOWSPDescription are Aspect Oriented Documentation

and Aspect Oriented Description respectively of the Web Service Provider, and their signatures are described below.

```
sig AODocumentation{
    description : String
}
```

The AODocumentation signature shown above contains summarized textual information about the whole Aspect-Oriented Web Service Provider. This documentation is verbose and human readable so that people can access it to find out what the web service provides including important information about the aspect-oriented components that exist inside the particular web service provider.

```
sig AOWSDescription{
    description : String
}
```

The above signature defines the AOWSDescription and it is the service provider's description. Just like the AODocumentation, it contains information about the whole web service provider, but the AOWSDescription is for intelligent agents to decipher and as such it contains less descriptive language. This makes it machine readable and serves to enable autonomous discovery and integration of the providers in aspect-oriented web services systems to be realised.

```
sig AOComponent{
```

```
    name : String,
    aoComponentDescription : AOComponentDescription,
    functionalAspect : set FunctionalAspect,
    nonFunctionalAspect : set NonFunctionalAspect
}
```

Shown above is the signature for a single Aspect-Oriented Component. An AOComponent contains a set of functionalAspects and a set of nonFunctionalAspects. Each AOComponent also has an AOComponentName, which is used as an identity for the AOComponent, and an AOComponentDescription. The set of functionalAspects in the AOComponent are those set of FunctionalAspects (functional aspects) which are related to specific business functionality, e.g. for *search, update, delete and insert* operations. On the other hand, the set of nonFunctionalAspects are the set of NonFunctionalAspects (non functional aspects) which are not core functionily, e.g. *performance*.

```
sig AOComponentDescription{
    description : String
}
```

The above AOComponentDescription signature contains information about the AOComponent and it is summarised machine readable textual description of the component for dynamic discovery and integration purposes.

```
sig FunctionalAspect {
```

```
    type : String,
    aspectName : String,
    aspectDescription : AspectDescription,
    parameter : Parameter,
    aoWSEntryPoint : String,
    standalone : String,
    return : FunctionalAspectReturn,
    aspectDetail : FunctionalAspectDetail,
    aspectUserOperations : AspectUserOperations,
    usesOperations : UsesOperations
}
```

The above signature defines a FunctionalAspect for AOComponent in AOWebServiceProviders. A FunctionalAspect consists of a type, an aspectName, AspectDescription, aoWSEntryPoint, standalone, FunctionalAspectReturn, FunctionalAspectDetail, AspectUserOperations, and UsesOperations. These are further defined and explained below.

```
sig AspectDescription{
description : String
}
```

The signature for AspectDescription is machine readable and gives crisp and clear description of the particular aspect.

```
sig Parameter{
    parameterName : String,
    parameterType : String
}
```

The Parameter signature shown above contains information about the parameters of the particular aspect so that the service can be accessed and consumed correctly.

```
sig FunctionalAspectDetail{
   type : String,
   detail : String,
   provided : Boolean
}
```

The signature FunctionalAspectDetail contains detailed information about the type of aspect involved, further details about the aspect, and whether the aspect provides or requires a particular function from other components.

```
sig FunctionalAspectReturn{
    returnType : String
}
```

The signature above defines the return type of functional aspect.

```
sig AspectUserOperations{
    usedBy : String
}
```

The signature above defines other operations (if any) within the provider that use the particular aspect. If the aspect is a top entry level operation, then there would not be any operations that need to be used before using this aspect because it is the first to be used.

```
sig UsesOperations{
    uses : String
}
```

The signature above defines other aspects that are needed by the aspect in order to operate. If the aspect is a terminal operation, then there would not be any more operations to be used after this aspect and the string would be a null.

```
sig NonFunctionalAspect{
    type : String,
    aspectName : String,
    aspectDescription : AspectDescription,
    parameter : Parameter,
    return : NonFunctionalAspectReturn,
    aspectDetail : NonFunctionalAspectDetail
```

```
}
```

The above signature defines a NonFunctionalAspect for an AOComponent in the service provider. It has similar corresponding fields as in the functional aspects except that the all details, types and descriptions are for non functional aspects.

```
sig NonFunctionalAspectReturn{
returnType : String,
units : String
}
```

The above signature NonFunctionalAspectReturn contains details about the information/value that is returned by the non-functional aspect.

```
sig NonFunctionalAspectDetail{
    type : String,
    detail : String,
    value : String,
    valueQualifier : String,
    provided : String
}
```

The above signature for NonFunctionalAspectDetail contains information about details about the aspect, the type of data that is retrieved from the aspect, other aspects that the aspect is providing service to, value and valueQualifier that can be used to check whether the system meets the specified requirement, e.g. the

qualification (based on the valueQualifier) could be that the performance of a data search operation displays the results within 2 seconds.

```
sig AOWebServiceRequester{
    aoconnector : AOConnector,
    newlyAdvertisedAOWSDL : lone AOWSDL,
    request : set Request
}
```

The AOWebServiceRequestor signature above has fields comprised of an AOConnector, a newlyAdvertisedAOWSDL (of type AOWSDL) and a set of Requests. The newlyAdvertisedAOWSDL represents the AOWSDL a new web service provider that has just registered with the AOUDDI. The Request object is any new requests made by the requester. The AOConnector in the AOWebServiceRequestor takes care and keeps track of all the communications and connections between the AOWebServiceRequestor and other objects or subsystems in the aspect-oriented web service system.

```
sig AOConnector{
    aocomposite : lone AOComposite,
    request : set Request,
    newlyAdvertisedAOWSDL : lone AOWSDL,
    result : lone Result,
    chosenAOWSDL : lone AOWSDL,
```

```
    aowsdl : set AOWSDL,
    oldAOWSDL : lone aowsdl,
    aouddi : AOUDDI
}
```

The above signature simulates the AOConnector object in the Aspect-Oriented Web Services System. The connector has fields comprised of an AOComposite, a set of Requests, the AOUDDI, newlyAdvertisedAOWSDL, oldAOWebServiceToDisconnect, Result, and a set of AOWebServiceProviders that are directly connected. The newlyAdvertisedAOWSDL contains the AOWSDL document of any new web service provider that has just been registered with the AOUDDI. Request objects contain any new request made by the requester and is defined below.

```
sig Request{
    request : String
}
```

The signature of a Request is comprised of request in textual (String) format made by an AOWebServiceRequestor to the AOConnector connected to it. The requester can only communicate with the other subsystems in the AOWS system through the connector object.

```
sig AOComposite{
    aowsdl : set AOWSDL,
```

```
    newAOWSDL : lone AOWSDL,
    oldAOWSDL : lone aowsdl
}
```

This AOComposite signature shown above contains the set of related and complimentary AOWSDLs of all AOWebServiceProviders that the particular requester needs to consume to carry out a myriad of tasks. These tasks could not be satisfied by a single provider. It also allows for the simulation of replacing a service provider in the AOComposite with one that is better.

```
sig AOUDDI{
    aowsdls : set AOWSDL,
    newAOWSDL : lone AOWSDL,
    aoconnectors : set AOConnector,
    request : set Request,
    result : lone Result,
    selectedAOConnector : lone aoconnectors
}
```

The above represents the signature of the AOUDDI, i.e. the Aspect-Oriented Universal Description Discovery Integration tool. The AOUDDI contains information about all available AOWebServiceProviders based on the AOWSDL files. The AOUDDI also has a field which stores the AOWSDL file of any newly registered AOWebServiceProvider. The new AOWSDL file will then be

advertised and made available to the AOConnectors. The AOUDDI has a Request field which is textual in nature made by the client and relayed to the AOUDDI through the connector. The AOUDDI also has a field called the result which includes in it the AOWSDL that best matches the request made by the AOWebServiceRequestor through its AOConnector.

```
sig Result{
    result : some AOWSDL
}
```

Shown above is signature of a Result that contains some AOWSDLs that matches a request for service provider(s) made by the client. (This Result originated from the AOUDDI in response to a request through the AOConnector).

Part 2: Description of facts and other constructs used for AOWS

This gives the list of facts that impose meaningful constraints on our model. Furthermore these facts have been applied to the system in order to generate simulations of the relationships between the main objects in the system. They have been categorized into *four* groups for modularity of the system design. The symbol '--' shown below before comments represents the comments' symbol for Alloy, it is used to provide summarised explanations for the facts and is not part of the executable code. The code itself is in blue ink.

Facts about AOUDDI

-- All AOWSDLs of the providers must be deposited in the AOUDDI so that they can be advertised and made available to the clients:

```
fact { all myAOWSDL : AOWSDL | (some myAOUDDI : AOUDDI |
myAOWSDL in myAOUDDI.aowsdls) }
```

-- There might be a new AOWSDL that is to be published in the AOUDDI from a new AOWebServiceProvider (i.e. when a new provider is registered in the AOUDDI):

```
fact { all myAOWSDL : AOWSDL | (lone myAOUDDI : AOUDDI |
myAOWSDL in myAOUDDI.newAOWSDL) }
```

-- All AOUDDIs must keep records of all AOConnectors (necessary for example when they need to be notified if any new provider gets becomes available):

```
fact {all myAOConnector : AOConnector | (one myAOUDDI : AOUDDI |
myAOConnector in myAOUDDI.aoconnectors) }
```

-- There might be a request from the AOWebServiceRequester that needs to be processed by the AOUDDI:

```
fact { all myRequest : Request |  (one myAOUDDI : AOUDDI | myRequest in
myAOUDDI.request) }
```

-- There might be a result (response) to a request that needs to be transmitted:

```
fact { all myResult : Result | (lone myAOUDDI : AOUDDI | myResult in
myAOUDDI.result) }
```

-- The Result object for a request for service providers will contain some AOWSDL :

```
fact { some myAOWSDL : AOWSDL | (some myResult : Result | myAOWSDL
in myResult.result) }
```

Facts about Aspect-Oriented Web Service Provider

Given below are the Alloy facts about the Aspect-Oriented Web Service Providers.

-- No two different service providers can have same AOWSDL:

```
fact { no aowsProvider1, aowsProvider2 : AOWebServiceProvider |
aowsProvider1.aowsdl = aowsProvider2.aowsdl }
```

-- All AOWSDL must be inside the AOWebServiceProvider

```
fact { all myAOWSDL : AOWSDL | (one aowsProvider :
AOWebServiceProvider | myAOWSDL in aowsProvider.aowsdl) }
```

-- No two different AOWSDLs can have same AOComponents object because, in the very least their identity will be different, as the identity of the AOComponents object derives in part from the unique AOWSDL location:

```
fact {no myAOWSDL1, myAOWSDL2 : AOWSDL |
myAOWSDL1.aoComponents = myAOWSDL2.aoComponents }
```

-- All AOComponents must be inside the AOWSDL as per the AOWSDL schema:

```
fact { all myAOComponents : AOComponents | (one aowsdl : AOWSDL |
myAOComponents in aowsdl.aoComponents) }
```

-- All AOComponent objects must be inside the AOComponents:

```
fact { all myAOComponent : AOComponent | (some myAOComponents :
AOComponents | myAOComponent in myAOComponents.aoComponent) }
```

-- All AODocumentation must be inside the AOComponents

```
fact { all myAODocumentation : AODocumentation| (one myAOComponents :
AOComponents | myAODocumentation in
```

```
myAOComponents.aoDocumentation) }
```

-- All AODocumentation must be inside the AOComponents:

```
fact { all myAOWSDescription : AOWSDescription| (one myAOComponents :
AOComponents | myAOWSDescription in
myAOComponents.aoWSDescription) }
```

-- All functional aspects must be inside AOComponent (as the functionalAspect objects of the AOComponent):

```
fact { all myFuncAspect : FunctionalAspect | (some myAOComponent :
AOComponent | myFuncAspect in myAOComponent.functionalAspect) }
```

-- All non-functional aspects must be inside AOComponent (exists as the nonFunctionalAspect of the component):

```
fact { all myNonFuncAspect : NonFunctionalAspect | (some myAOComponent :
AOComponent | myNonFuncAspect in myAOComponent.nonFunctionalAspect)
}
```

-- All AOComponentDescription must be inside AOComponent (and exists as its aoComponentDescription):

```
fact { all myAOComponentDescription : AOComponentDescription| (one
myAOComponent : AOComponent | myAOComponentDescription in
myAOComponent.aoComponentDescription) }
```

-- All AspectDescription objects are inside FunctionalAspects and NonFunctionalAspects:

```
fact { all myAspectDescription : AspectDescription | (one myFunctionalAspect : FunctionalAspect | myAspectDescription in myFunctionalAspect.aspectDescription)  || (one myNonFunctionalAspect : NonFunctionalAspect | myAspectDescription in myNonFunctionalAspect.aspectDescription)}
```

-- All FunctionalAspects and NonFunctionalAspects have Parameter inside them:

```
fact { all myParameter : Parameter | (one myFunctionalAspect : FunctionalAspect | myParameter in myFunctionalAspect.parameter) || (one myNonFunctionalAspect : NonFunctionalAspect | myParameter in myNonFunctionalAspect.parameter) }
```

-- All FunctionalAspects have FunctionalAspectDetail inside them:

```
fact { all myFunctionalAspectDetail : FunctionalAspectDetail | (one myFunctionalAspect : FunctionalAspect | myFunctionalAspectDetail in myFunctionalAspect.aspectDetail )}
```

-- All NonFunctionalAspects have NonFunctionalAspectReturn inside them:

```
fact { all myNonFunctionalAspectReturn : NonFunctionalAspectReturn | (one myNonFunctionalAspect : NonFunctionalAspect | myNonFunctionalAspectReturn in myNonFunctionalAspect.return) }
```

-- All NonFunctionalAspects have NonFunctionalAspectDetail inside them:

```
fact { all myNonFunctionalAspectDetail : NonFunctionalAspectDetail | (one
myNonFunctionalAspect : NonFunctionalAspect |
myNonFunctionalAspectDetail in myNonFunctionalAspect.aspectDetail)}
```

-- All NonFunctionalAspect and FunctionalAspect must have usesOperation inside them:

```
fact { all myUsesOperations : UsesOperations | (one myFunctionalAspect :
FunctionalAspect | myUsesOperations in myFunctionalAspect.usesOperations) }
```

-- All NonFunctionalAspect and FunctionalAspect must have usesOperation inside them:

```
fact { all myAspectUserOperations : AspectUserOperations | (one
myFunctionalAspect : FunctionalAspect | myAspectUserOperations in
myFunctionalAspect.aspectUserOperations) }
```

Facts about Aspect Oriented Web Service Requester, AOConnector and AOComposite

-- All AOConnectors are connected to the AOWebServiceRequester:

```
fact { all myAOConnector : AOConnector | (one aowsRequester :
AOWebServiceRequester | myAOConnector in aowsRequester.aoconnector) }
```

-- Take into consideration that there might be newly advertised AOWSDL in that the requester can use:

```
fact { all myAOWSDL : AOWSDL | (lone aowsRequester :
AOWebServiceRequester | myAOWSDL in
aowsRequester.newlyAdvertisedAOWSDL) }
```

-- All requests made by the AOWebServiceRequester are stored in the Requester's set of requests:

```
fact { all myRequest : Request | (one aowsRequester : AOWebServiceRequester |
myRequest in aowsRequester.request) }
```

-- If the AOConnector does not have knowledge of any newlyAdvertisedAOWSDL, then the requester shouldn't have them as well because all AOWSDLs go to the connector first before being consumed:

```
fact { all myAOWSDL : AOWSDL | (all myRequester:AOWebServiceRequester
| (myAOWSDL !in myRequester.aoconnector.newlyAdvertisedAOWSDL) =>
(myAOWSDL !in myRequester.newlyAdvertisedAOWSDL) )}
```

-- If aoconnector does not have newlyAdvertisedAOWSDL, aocomposite shouldn't have them as well because all AOWSDLs go to the connector first before being sent to the composite object to form the aggregate object of service providers:

```
fact { all myAOWSDL : AOWSDL|  (all myAOConnector : AOConnector |
(myAOWSDL !in myAOConnector.newlyAdvertisedAOWSDL) =>
(myAOWSDL !in myAOConnector.aocomposite.newAOWSDL))}
```

-- NewlyAdvertisedAOWSDL in the aoconnector must be same with newlyAdvertisedAOWSDL in the requester:

```
fact { all aowsRequester : AOWebServiceRequester |
(aowsRequester.aoconnector.newlyAdvertisedAOWSDL =
aowsRequester.newlyAdvertisedAOWSDL) }
```

-- The AOConnector might have AOComposite (as in the case of using an aggregate complex of service providers to carry out a variety of tasks):

```
fact { all myAOComposite : AOComposite | (one myAOConnector :
AOConnector | myAOComposite in myAOConnector.aocomposite) }
```

-- AOConnector might have new request from the requester that is yet to be processed:

```
fact { all myRequest : Request | (one myAOConnector : AOConnector |
myRequest in myAOConnector.request) }
```

-- AOConnector might have newly advertised AOWSDL:

```
fact { all myAOWSDL : AOWSDL | (lone myAOConnector : AOConnector |
myAOWSDL in myAOConnector.newlyAdvertisedAOWSDL) }
```

-- AOConnector might have result :

```
fact { all myResult : Result | (lone myAOConnector : AOConnector | myResult
in myAOConnector.result) }
```

-- AOConnector might have chosen one AOWSDL from the result that it has received:

```
fact { all myAOWSDL : AOWSDL| (lone myAOConnector : AOConnector |
(myAOWSDL in myAOConnector.chosenAOWSDL && myAOWSDL in
myAOConnector.result.result)) }
```

-- AOConnector might still be linked to one AOWSDL that has become useless and needs to be disconnected (e.g. when it has discovered and integrated with a better provider):

```
fact { all myAOWSDL : AOWSDL | (lone myAOConnector : AOConnector |
myAOWSDL in myAOConnector.oldAOWSDL) }
```

-- AOConnector has direct connection (i.e. not through AOComposite block) to some AOWebServiceProviders:

```
fact { some myAOWSDL : AOWSDL | (some myAOConnector : AOConnector |
myAOWSDL in myAOConnector.aowsdl) }
```

-- All AOConnectors has to be connected to AOUDDI to enable discovery of web service providers:

```
fact { all myAOConnector : AOConnector | (one myAOUDDI : AOUDDI |
myAOConnector in myAOUDDI.aoconnectors && myAOUDDI in
myAOConnector.aouddi) }
```

-- If aowsRequester made a request, the AOConnector must have that request as well because the connector object is the only link to the requester to communicate with the rest of AOWS subsystems:

```
fact { all myRequest : Request | (one aowsRequester : AOWebServiceRequester |
myRequest in aowsRequester.request   => myRequest in
aowsRequester.aoconnector.request  ) }
```

-- If the AOWSDL of a web service provider is in the connector and if it is already directly connected, then the AOComposite shouldn't have the AOWSDL:

```
fact { all myAOWSDL : AOWSDL |  (one myAOConnector : AOConnector |
myAOWSDL in myAOConnector.aowsdl => myAOWSDL !in
myAOConnector.aocomposite.aowsdl) }
```

-- The AOComposite should be integrated with a few AOWebServiceProviders so that a greater number of different tasks may be performed:

```
fact { some myAOWSDL : AOWSDL | (some myAOComposite : AOComposite
| myAOWSDL in myAOComposite.aowsdl) }
```

-- The AOComposite should have AOWSDLs that contains information about web services needed by the client:

```
fact { all myAOWSDL : AOWSDL | (lone myAOComposite : AOComposite |
myAOWSDL in myAOComposite.newAOWSDL) }
```

-- The AOComposite should have the AOWSDL of the web service that needs to be removed before we can remove it (and connect to a better service provider):

```
fact { all  myAOWSDL : AOWSDL | (lone myAOComposite : AOComposite |
myAOWSDL in myAOComposite.oldAOWSDL) }
```

Part 3: Predicates used to model and analyze AOWS

Predicates:

-- A new AOWebServiceRequester needs to connect to the AOConnector first before it can access any of the other subsystems:

```
pred RequesterConnectToAOConnector ( aowsRequester :
AOWebServiceRequester, myAOConnector : AOConnector ) {
  aowsRequester.aoconnector = myAOConnector
}
```

-- After an AOWebServiceRequester connects to the AOConnector object, the connector will then connect to the AOUDDI for discovery purposes

```
pred AOConnectorConnectToAOUDDI ( myAOUDDI, myAOUDDI' :
AOUDDI, myAOConnector : AOConnector) {
  -- the precondition:
  myAOConnector !in myAOUDDI.aoconnectors
  -- update the AOUDDI and connector (post-condition):
  myAOUDDI'.aoconnectors = myAOUDDI.aoconnectors + myAOConnector
  myAOConnector.aouddi = myAOUDDI'
}
```

-- Every time a new AOWebServiceProvider is introduced into the AOWS domain, the new provider has to register itself to the AOUDDI by sending its

AOWSDL file to the AOUDDI to be published. Any new service provider (for it to become discoverable by the clients in the system) must register itself and also deposit its AOWSDL in the AOUDDI. This entry of the AOWebServiceProvider will be registered into the AOUDDI Registry (myAOUDDI.aowsdls = myAOUDDI.aowsdls + aowsProvider.aowsdl). The AOUDDI will update its records and save the AOWSDL (myAOUDDI.newAOWSDL = aowsProvider.aowsdl).

```
pred RegisterNewAOWS ( myAOUDDI, myAOUDDI' : AOUDDI, aowsProvider
: AOWebServiceProvider) {
    -- precondition:
    aowsProvider.aowsdl !in myAOUDDI.aowsdls
    aowsProvider.aowsdl !in myAOUDDI.newAOWSDL
    -- add the new provider's AOWSDL to the AOUDDI and update the
AOUDDI:
    myAOUDDI'.aowsdls = myAOUDDI.aowsdls + aowsProvider.aowsdl
    myAOUDDI'.newAOWSDL = aowsProvider.aowsdl
}
```

-- The AOUDDI will then forward the information about new AOWebServiceProvider to the AOConnector through the notification process shown below:

-- Every time a new AOWebServiceProvider gets registered into the AOUDDI, the AOUDDI will notify the AOConnector about its existence. The AOUDDI will

send the AOWSDL, which will be used by the AOConnector to find out what aspects are provided by AOWebServiceProvider.

```
pred NotifyAOConnectorAboutNewAOWS ( myAOUDDI : AOUDDI, aoConnector : AOConnector ) {
    -- precondition – the aoconnector has to be connected to the AOUDDI for the connector to be able to receive notifications:
    aoConnector in myAOUDDI.aoconnectors
    -- post condition – the AOUDDI successfully transmits the new AOWSDL file to the connector:
    aoConnector.newlyAdvertisedAOWSDL = aoConnector.aouddi.newAOWSDL
}
```

-- Each time the AOConnector gets notified of a new AOWebServiceProvider, it must also notify the AOWebServiceRequester that is dependent on it. The AOConnector will forward the AOWSDL to the AOWebServiceRequester, so that the client knows the types of services that are provided by new AOWebServiceProvider:

```
pred NotifyRequesterAboutNewAOWS ( myAOConnector : AOConnector, aowsRequester : AOWebServiceRequester ) {
    --precondition:
    myAOConnector in aowsRequester.aoconnector
    --postcondition:
```

```
    aowsRequester.newlyAdvertisedAOWSDL                                    =
myAOConnector.newlyAdvertisedAOWSDL
}
```

-- The AOConnector will connect to new AOWS provider directly (i.e. without the need of an AOComposite) if the single service provider itself can satisfy all the client's needs:

```
pred DirectConnectionToNewAOWS ( myAOConnector, myAOConnector' :
AOConnector ) {
   --precondition:
   myAOConnector.newlyAdvertisedAOWSDL !in myAOConnector.aowsdl
   -- update:
   myAOConnector'.aowsdl = myAOConnector.aowsdl +
myAOConnector.newlyAdvertisedAOWSDL
}
```

-- After connecting to the new service provider, the AOConnector will select the redundant service provider to be disconnected from it:

```
pred SelectAOWSToDisconnect ( myAOWSDL : AOWSDL, myAOConnector :
AOConnector ) {
   --precondition
   myAOWSDL in myAOConnector.aowsdl


   --postcondition
   myAOConnector.oldAOWSDL = myAOWSDL
```

```
}
```

-- The selected AOWSDL of the redundant provider from the above process will then be removed from the AOConnector:

```
pred DisconnectDirectConnection ( myAOConnector, myAOConnector' :
AOConnector) {
   --precondition:
   myAOConnector.oldAOWSDL in     myAOConnector.aowsdl


   --update:
   myAOConnector'.aowsdl = myAOConnector.aowsdl -
myAOConnector.oldAOWSDL
}
```

-- The AOConnector will get the AOComposite to combine several AOWS providers into an aggregate object:

```
pred GetAOComposite ( myAOConnector : AOConnector, myAOComposite :
AOComposite ) {
    myAOConnector.aocomposite = myAOComposite
}
```

-- To replace an existing provider in the composite object with a better service provider, send the two AOWSDLs of the providers to the AOComposite, (one is AOWSDL of the provider that will be connected and the other is AOWSDL of the provider that will be disconnected):

```
pred SendAOWSDLToAOComposite ( myAOConnector : AOConnector,
```

```
myAOComposite : AOComposite ) {
    myAOComposite.newAOWSDL =
myAOConnector.newlyAdvertisedAOWSDL
    myAOComposite.oldAOWSDL = myAOConnector.oldAOWSDL
}
```

-- The AOComposite will be connected to the new service provider:

```
pred IndirectConnectionToNewAOWS( myAOComposite, myAOComposite' :
AOComposite ) {
    -- precondition:
    myAOComposite.newAOWSDL !in myAOComposite.aowsdl
    -- update:
    myAOComposite'.aowsdl = myAOComposite.aowsdl +
myAOComposite.newAOWSDL
}
```

-- The AOComposite object will then disconnect the “old” unneeded AOWS provider:

```
pred DisconnectIndirectConnection ( myAOComposite, myAOComposite' :
AOComposite ) {
    -- precondition:
    myAOComposite.oldAOWSDL in myAOComposite.aowsdl

    -- update:
    myAOComposite'.aowsdl = myAOComposite.aowsdl -
myAOComposite.oldAOWSDL
```

```
}
```

-- The AOWebServiceRequester must be able to create new requests:

```
pred CreateRequest ( myRequest : Request, aowsRequester :
AOWebServiceRequester ) {
    aowsRequester.request = myRequest
}
```

-- The AOWebServiceRequester should be able to send this new request to AOConnector:

```
pred SendRequestToAOConnector ( aowsRequester : AOWebServiceRequester,
myAOConnector : AOConnector ) {
    myAOConnector.request = aowsRequester.request
}
```

-- The AOConnector should be able to then pass the request to the AOUDDI if it is to query for service providers:

```
pred SendRequestToAOUDDI ( myAOUDDI, myAOUDDI' : AOUDDI,
myAOConnector : AOConnector) {
    --precondition:
    myAOConnector.request !in myAOUDDI.request
    --update:
    myAOUDDI'.request = myAOUDDI.request  + myAOConnector.request
}
```

-- The AOUDDI will process the request to compute the result and transmit it

back to the aoconnector:

```
pred ComputeResultAndTransmit( myResult : Result, myAOUDDI : AOUDDI,
myAOConnector : AOConnector ) {
   myAOUDDI.result = myResult
   myAOConnector.result = myAOUDDI.result
}
```

-- The connector can select the best AOWS provider:

```
pred SelectBestAOWS ( myAOConnector : AOConnector, myAOWSDL :
AOWSDL ) {
   -- precondition:
   myAOWSDL in myAOConnector.result.result
   -- postcondition:
   myAOConnector.chosenAOWSDL = myAOWSDL
}
```

-- The AOConnector will connect to selected AOWS provider directly (i.e. it does not need the help of the AOComposite complex as the connector is integrating directly to a single provider):

```
pred DirectConnectionToRequestedAOWS ( myAOConnector,
myAOConnector' : AOConnector ) {
   --precondition:
   myAOConnector.chosenAOWSDL !in myAOConnector.aowsdl
   --update the aowsdl field of the AOConnector:
   myAOConnector'.aowsdl = myAOConnector.aowsdl +
myAOConnector.chosenAOWSDL
```

```
}
```

-- The AOConnector will pass the information about the selected AOWSDL to the AOComposite object to use:

```
pred TransmitAOWSDLToAOComposite ( myAOConnector : AOConnector,
myAOComposite : AOComposite ) {
   myAOComposite.newAOWSDL = myAOConnector.chosenAOWSDL
}
```

-- The AOComposite will connect to the selected provider:

```
pred IndirectConnectionToRequestedAOWS( myAOComposite,
myAOComposite' : AOComposite ) {
   -- precondition:
   myAOComposite.newAOWSDL !in myAOComposite.aowsdl
   -- update:
   myAOComposite'.aowsdl = myAOComposite.aowsdl +
myAOComposite.newAOWSDL
}
```

Part 4: Predicates used to model and analyze an example of using Aspects to carry out dynamic search operations.

Given below are samples of the signatures and predicates used to model and analyse the entities and the relationships involved between them and the various aspects and aspect details involved in searching for a hotel room using our Travel Planner application.

Signatures:

The additional signatures required here are mentioned below:

```
sig SearchForHotel extends FunctionalAspect { }

sig SearchForHotelRoom extends FunctionalAspect { }

sig SearchForHotelDetail extends FunctionalAspectDetail { }

sig SearchForHotelRoomDetail extends FunctionalAspectDetail { }

sig DataRetrieval extends String { }

sig SearchForHotelDataRetrieval extends DataRetrieval { }

sig SearchForHotelRoomDataRetrieval extends DataRetrieval { }

sig Select extends String { }

sig SelectHotel extends Select { }

sig SelectHotelRoom extends Select { }

sig Persistency extends String { }

sig DataSet extends String { }

sig  HotelName extends String { }

sig HotelRoomInfo extends String { }
```

Predicates:

-- Initialize the aowsentrypoint and the standalone fields:

```
pred Initialize ( funcAspect : FunctionalAspect, bool : Boolean ) {
    funcAspect.aoWSEntryPoint = bool
    funcAspect.standalone = bool
}
```

-- Set the details about the aspect:

```
pred SetAspectDetailDetail ( funcAspect : FunctionalAspect, string : String ) {
    funcAspect.aspectDetail.detail = string
}
```

-- Change aowsentrypoint to true if aspect does not require any other aspect to execute before using it, i.e. if this can be used as the starting operation when consuming the web service:

```
pred SetAOWSEntryPoint ( funcAspect : FunctionalAspect, bool : Boolean) {
    funcAspect.aoWSEntryPoint = bool
}
```

-- Change the standalone to true if the aspect does not require other operations to be executed after this:

```
pred SetStandalone ( funcAspect : FunctionalAspect, bool : Boolean ) {
    funcAspect.standalone = bool
```

```
}
```

-- Set the provided as true or false as it is of a boolean type:

```
pred SetProvided( funcAspect : FunctionalAspect, bool : Boolean ) {
    funcAspect.aspectDetail.provided = bool
}
```

-- Set the user of the aspect:

```
pred SetUserOperation(funcAspect1 : FunctionalAspect, funcAspect2 :
FunctionalAspect){
    funcAspect1.userOperation = funcAspect2.aspectName
}
```

-- Set the Type in the Aspect Detail to a String object:

```
pred SetAspectDetailType( funcAspect : FunctionalAspect, string : String ) {
    funcAspect.aspectDetail.type = string
}
```

-- Set the Type in the Aspect to a String object:

```
pred SetType ( funcAspect : FunctionalAspect, string : String ) {
    funcAspect.type = string
}
```

-- Set the ReturnType to a String object:

```
pred SetReturnType ( funcAspect : FunctionalAspect, string : String ) {
```

```
    funcAspect.returnType = string
}
```

-- Set the parameter name to a String object:

```
pred SetParameterName ( funcAspect : FunctionalAspect, string : String ) {
    funcAspect.parameter.parameterName = string
}
```

-- Set the parameter type of the Functional Aspect to a String object:

```
pred SetParameterType ( funcAspect : FunctionalAspect, string : String ) {
    funcAspect.parameter.parameterType = string
}
```

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

Reference Number*.....2005/433.....
This number will be assigned when the application is accepted for the UAHPEC agenda. You will receive an acceptance letter with the number included. Quote this reference number on all documentation to the Committee and Participants.

University of Auckland Human Participants Ethics Committee

RESEARCH PROJECT APPLICATION FORM (2005)

Applications will only be accepted on forms dated for the current year. Please complete this form in reference to the *UAHPEC Guidelines 2003* available on the University of Auckland website under Research and Research Ethics and Biological Safety Administration. Submit one unstapled, single sided copy of the form and all accompanying documentation to the Research Ethics and Biological Safety Administration, the Secretariat, Room 016 Alfred Nathan House, 24 Princes Street. For Yes or No answers delete whichever does not apply. Use language that is free from jargon and comprehensible to lay people.

GENERAL INFORMATION / COVERSHEET

1. **PROJECT TITLE:** Supporting Web Services Systems Specification using Aspect-Oriented Component Engineering

2. **APPLICANT/PRINCIPAL INVESTIGATOR (P.I.)** (Ph D student)
Name: Santokh Singh
Address: Department of Computer Science
Email address: santokh@cs.auckland.ac.nz
Phone number: 09 373-7599 ext 82283

Supervisor:
Name: Prof John Hosking
Address: Department of Computer Science
Email address: john@cs.auckland.ac.nz
Phone number: 09 373-7599 ext 88297
If Doctoral student, name of degree, Department and Supervisor.

3. **NAME OF STUDENT:** (If applicable) Santokh Singh
Address: Department of Computer Science
Email address: santokh@cs.auckland.ac.nz
Phone number: 09 373 7599 ext 82283
Name of degree and Department: PhD, Dept of Computer Science, Faculty of Science

4. **OTHER INVESTIGATORS:**
Names:
Organisation:
Is ethical approval being applied for from another institution? **NO**

(If YES, indicate name of the institution and attach evidence.) .....

5. **AUTHORISING SIGNATURES:**

HEAD OF DEPARTMENT: .....Date:.....

HOD name printed: .....Department.....

6. APPLICANT'S DECLARATION

The information supplied is, to the best of my knowledge and belief, accurate. I have read the current University of Auckland Human Participants Ethics Committee Guidelines. I clearly understand my obligations and the rights of the participants, particularly in regard to obtaining freely given informed consent.

Signature of P.I. /Supervisor…..……. Date: ……..……

Signature of Student:……..…… Date: ……..……

If a student project, including doctorate, signatures of both the Supervisor and the student are required.

SECTION A: PROJECT

Use as much space as is necessary to complete your answers.

Type your answers in 12pt Times New Roman, beginning on the line below the question.

1. AIM OF PROJECT:

a) What is the hypothesis / research question(s)? (State briefly)

Web services (discussed below) are a new breed of distributed systems and are still a maturing technology. Many questions, especially pertaining to their performance, security and interoperability, are yet not answered. In addition, most web service-based systems are currently designed using conventional object-oriented analysis and design approaches. During the development of a number of distributed systems, we have found that such design approaches do not adequately help developers to capture, reason about and encode higher level component capabilities and are especially poor with respect to addressing issues relating to cross-cutting component services. As such we wish to answer the following research questions:

1. Can we make better characterized and categorized novel aspect-oriented web services (AOWS) systems?
2. Are these systems better at description, discovery and integration as compared to existing web services systems?
3. Can these systems be made more autonomous as regards discovery and integration?

.

b) What are the specific aims of the project?

The primary objective of this thesis is to research and propose a new breed of novel dynamic aspect-oriented web services system that is better characterized and categorized, and then design, develop and provide support for such systems by extending and applying the Aspect-Oriented Component Engineering, (AOCE) methodology. There is a usability testing component part to the project which requires human participants, for which ethical approval is sought.

2. RESEARCH BACKGROUND

Provide sufficient information to place the project in perspective and to allow the significance of the project to be assessed.

Most new distributed systems now use internet technologies as a fundamental part of their remoting architecture. However most lack the ability to work over a wide variety of internet services with security constraints, lack adequate dynamic queryable descriptions and binding services, use proprietary solutions, or have limited cross-platform or cross-language support features, together with complex data structure representations that are specific to the language used.

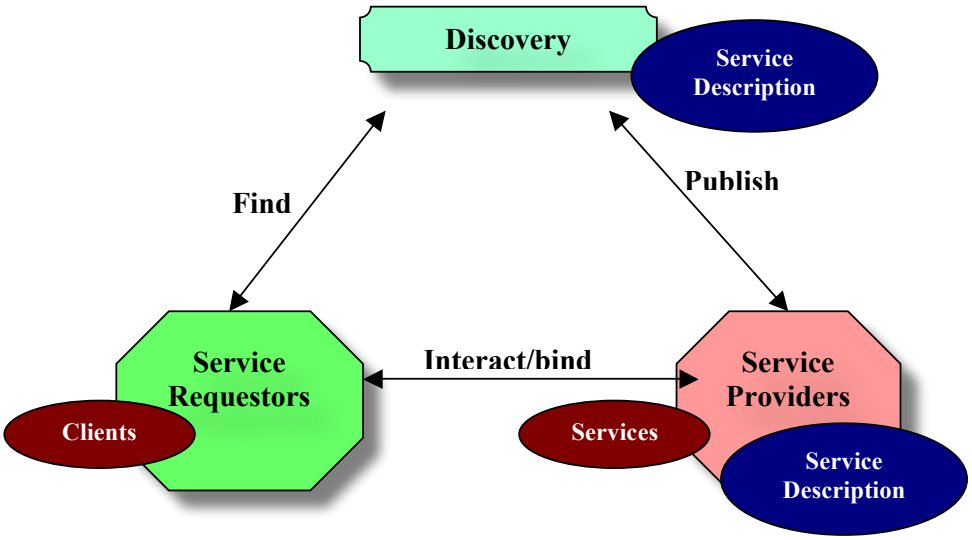


Figure 1: Generic web services architecture

One solution to overcome these problems has been the development of web services, Figure 1 shows its generic architecture. These are basically remote component services described, located and accessed using a set of open standards from the World Wide Web Consortium, (W3C). Using web services gives rise to the very promising possibility of allowing heterogeneous application integration over the internet. Web services have quickly become popular in large part because they build on a well known and widely accepted meta language, called the eXtensible Mark-up Language, or XML. They provide a basic communication infrastructure on which existing remote object systems can operate, by using HTTP as a de facto Web Service message carrier.

Web services provide a simple, standardised mechanism for describing services within service documents, and allow for locating these web services by indexing discovery agencies, and further allow for the co-ordination of cross-system processes. A prediction is that web services will be the next wave in business process automation (BPA). This is because from a business perspective, web services provide a newer and better way to enhance, extend, and even reengineer the capabilities of current strategic business applications for BPA, including software systems that deal with Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Supply Chain Management (SCM) systems.

However, web services are still a maturing technology. It has been pointed out that many questions, especially pertaining to their performance, security and interoperability, are yet not answered. In addition, most web service-based systems are currently designed using conventional object-oriented analysis and design approaches. During the development of a number of distributed systems, we have found that such design approaches do not adequately help developers to capture, reason about and encode higher level component capabilities and are especially poor with respect to addressing issues relating to cross-cutting component services.

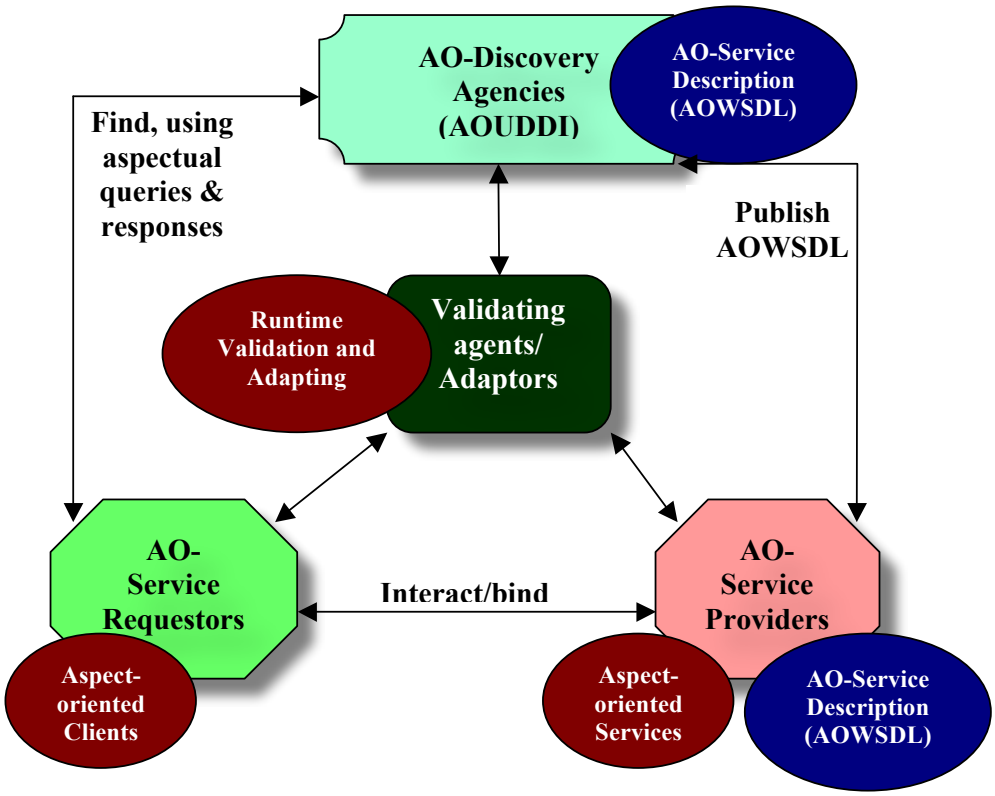


Figure 2: Aspect-oriented web services architecture (AO denotes Aspect-oriented)

The primary objective of this thesis is to research and propose a new breed of novel dynamic aspect-oriented web services (AOWS) system, as shown in Figure 2, that is better characterised and categorised using aspect-oriented components, and then design, develop and provide support for such systems by extending and applying the Aspect-Oriented Component Engineering, (AOCE) methodology. We also carried out research to provide ways to support better and more efficient description, dynamic discovery and integration in our novel web services systems specification approach using AOCE. These features are either lacking or cannot be supported in existing web services technologies.

The reason we chose the AOCE development methodology is that, currently used component-based systems engineering approaches for web services development are inadequate and tend to focus more on low-level software component interface design and implementation. This has the problem of the techniques being both cumbersome and difficult to comprehend. This also limits, and in worst case prevents the reusability of software components produced. This results in unnecessary wastage in terms of time, effort and resources. We propose the use of AOCE to address these shortcoming and need some feedback from our graduating students who have used it and volunteer to answer the questionnaire.

- Users' prior experience of using AOCE and web services will help answer the questions in this evaluation and these answers will be recorded in the questionnaire form itself.
- All participants will be normal adults who are computer literate.

2. **Describe and discuss the ethical issue(s) arising from this project.** (Be sure to address these in the body of the application.)

We do not foresee any potential ethnical issues that could arise from the usability testing. Both the applicant and applicant's student are members of an academic department (Computer Science). They

have no influence on potential participants' academic outcomes in statistical survey related courses as no such courses are run in the Computer Science Department. Also there will be no tangible financial rewards for participants as they will be take part in the usability testing on a strictly voluntary basis.

SECTION B: PARTICIPANTS

The term 'participants' is taken to mean subjects, clients, informants and patients as well as persons subjected to experimental procedures.

1. **What types of people are participating in the research?** (Delete those who do not apply).
   Normal Adults

2. **Explain how many organisations, departments within the organisations, and individuals you wish to recruit.** (Attach any letter of support you may have had from an organisation)

   About 8 participants. To test the AOCE methodology, during the summer school at the beginning of this year, 8 final year software engineering students had volunteered to do development of web service based-systems (for mobile applications) using the AOCE (eXtreme AOCE) development methodology. Their project was not part of my thesis but the techniques and technology used is related to the work that I had already done and published prior to their project. Their project was for mobile devises. As such the questions answered by this group of graduating students would be very relevant and necessary for evaluation purposes.

   The user evaluation will be conducted by a single person (the applicant's student, Santokh Singh) over a week with each participant taking about one hour to complete.

3. **How will you identify your potential participants?** (If by advertisement / notice, attach a copy)
   From a pool of Software Engineering students who did summer projects with us on related topics, i.e. mobile applications consuming aspect-oriented web services.

4. **How and where will potential participants be approached? Explain how you will obtain the names and contacts of participants.** (e.g. by email, by advertisement, through an agency holding these details.)
   Emails

5. **Who will make the initial approach to potential participants?** (e.g. will the owner of the database send out letters?)
   By the applicant's research student, Santokh Singh

6. **Is there any special relationship between participants and researchers?** (e.g. student / teacher. If YES, explain.) **YES**

   They are our former software engineering students who are finishing their studies this year. Some of them are our project students but their reports etc. have already been marked and assessed, with the examiners meeting finished. As such they are not in any way prejudiced or coerced, nor would they feel obliged to provide answers against their wishes. This is made clear to them in our information sheet (attached with this application) to them.

7. **Are there any potential participants who will be excluded?** **YES**

(If YES, explain, and state the criteria for excluding participants).

As the students are graduating this year, some of them might leave for overseas for work or go back to their home country and as such may not be available for the survey if approval is not obtained early.

SECTION C: RESEARCH PROCEDURES

There is a need here to fully inform the Committee about all factors relating to the research, including where appropriate, the researchers' qualifications to conduct this work (Investigation).

1. **PROJECT DURATION** (approximate dates): **From…1/March/2003 to 28/Feb/2006**

2. **Describe the study design.** (E.g. longitudinal study)

Questionnaire

3. **List all the methods used for obtaining information.** (Attach questionnaires / research instruments / interview guidelines to this application).
Anonymous questionnaires

4. **Who will carry out the research procedures?**
By the applicant and the research student, Santokh Singh

5. a) **Where will the research procedures take place?** (Physical location / setting).
Anywhere convenient to them or the Computer Science department, The University of Auckland

b) **If the study is based overseas, which countries are involved?** (Provide local contact information on the Participant Information Sheet(s).) NA

c) **If the study is based overseas, explain what special circumstances arise and how they will be dealt with? Explain any special requirements of the country and / or the community with which the research will be carried out.** NA

6. **How much time will participants need to give to the research?** (Indicate this in the Participant Information Sheet(s).)
Participants just need to answer the questions in a questionnaire form that will be given to them. The whole procedure will take no more than one hour. The questionnaire form is attached to this application.

7. **Does this research include the use of a questionnaire / email?** (If YES, attach a copy to this application.)
YES

8. **Are you intending to conduct the research in (University) class time?** (If YES, include advice from the course Coordinator giving approval for this to occur.) **NO**

9. **Is deception involved at any stage of the research?** (If YES, justify its use, and describe the debriefing procedure.) **NO**

10. **Will information on the participants be obtained from third parties?** (e.g. from participant's employer, teacher, doctor etc. If YES, explain, and indicate in the Participant Information Sheet(s).) **NO**

11. **Will any identifiable information on the participants be given to third parties?** (If YES, explain, and indicate in the Participant Information Sheet(s).) **NO**

12. **Provide details on any compensation or reimbursement of expenses, and where applicable, level of payment to be made to participants.** (If payment / koha is offered, explain in the Participant Information Sheet(s).)
All participants are volunteers and they will not be rewarded financially for their participation.

13. a) **Does the research involve the administration of any substance** (e.g. eye-drops / food) **to participants?** **NO**

b) **Does this research involve potentially hazardous substances,** (e.g. radioactive materials)**?** **NO**

SECTION D: INFORMATION & CONSENT

1. **By whom and how, will information about the research be given to participants?** (e.g. in writing, verbally – a copy of the information given to prospective participants in the form of Participant Information Sheet(s) must be attached to this application.)
Verbally and in writing, the latter as part of the attached.

2. a) **Will the participants have difficulty giving informed consent on their own behalf?** (Consider physical or mental condition, age, language, legal status, or other barriers.) **NO**

b) **If participants are not competent to give fully informed consent, who will consent on their behalf?** (e.g. parents / guardians)

3. **Consent should be obtained in writing. Explain and justify any alternative to written consent.**

Consent will be obtained verbally and in writing, the latter using the attached consent form.

4. **It is expected that access to the Consent Forms be restricted to the researcher and/or the Principal Investigator. If you intend otherwise, please explain.** N/A

5. **Will Consent Forms be stored by the Principal Investigator, in a locked cabinet, on University premises?** YES

6. **It is required that Consent Forms be stored separately from data and kept for six years. If a different procedure is to be followed, describe and justify.** N/A

SECTION E: STORAGE & USE OF RESULTS

1. **Will the participants be audio-taped or video-taped, or recorded by any other electronic means?** (If YES, explain in the Participant Information Sheet(s) and the Consent Form. Consider whether recording is an optional or necessary part of the research design, and reflect this in the Consent Form.) NO

2. a) **How will data, including audio and videotapes and electronic data be handled and stored to protect against unauthorised access?** (Explain this in the Participant Information Sheet(s) with details of storage, possible future use and eventual destruction.)

All resulting observational data (questionnaire feedbacks) will be compiled and will be kept in a secure locked cabinet within the Department of Computer Science, the University of Auckland.

b) **If the tapes are being transcribed / translated by someone other than the researcher, explain what arrangements are in place to protect the confidentiality of participants.** (Attach any confidentiality agreements to this application.) N/A

c) **If recordings are made, will participants be offered the opportunity to edit the transcripts of the recordings?** (In either case, the Participant Information Sheet must inform the participants. Where participants are asked to make a choice, this should be shown on the Consent Form.)

No recordings will be made N/A

d) **Will participants be offered their tapes (or a copy thereof)?** (In either case, the Participant Information Sheet must inform the participants. Where participants are asked to make a choice, this should be shown on the Consent Form.) No recordings will be made N/A

e) **Will data or other information be stored for later use?** NO

i) **If YES, explain how long the data will be stored and how it will be used.** (Indicate this in the Participant Information Sheet(s). The period data is to be kept will be commensurate to the scale of its research. For peer reviewed publication or research that might be further developed, the University expects six years.)

ii) **If NO, describe how and when the data will be destroyed.** (Indicate this in the Participant Information Sheet(s).)

The questionnaires will be destroyed by shredding after 6 years.

f) **Describe any arrangements to make results available to participants, including whether they will be offered their tapes.** (Explain this in the Participant Information Sheet(s). Where participants are asked to make a choice, this should be shown on the Consent Form.)

A summary of the survey results and resulting publications will be made available to participants on request.

3. a) **Are you going to use the names of the research participants in any publication or report about the research?** (The Participant Information Sheet(s) must inform the participants, and be part of the consent obtained in the Consent Form(s). This is a problem either when you are dealing with a small group of participants known to a wider public or when there is to be a report back to participants likely to know each other.) NO

b) If you don't use their names, is there any possibility that individuals or groups could be identified in the final publication or report? (If YES, explain, and describe in the Participant Information Sheet(s). **YES**

Given the specialized nature of this field and the limited number of people who have used this development methodology, there may be a possibility that someone may identify the group involved as a whole but they may not be able to identify individuals involved.

SECTION F: TREATY OF WAITANGI

1. **Does the proposed research impact on Maori persons as Maori? If YES, complete all questions in this section and attach evidence of consultation from the nominated Maori Advisor within your Faculty**. (If NO, go to Section G.) **NO**

2. **Explain how the intended research process is consistent with the provisions of the Treaty of Waitangi.** (Refer to the Guidelines for further information)

3. **Identify the group(s) with whom consultation has taken place, describe the consultation process, and attach evidence of the support of the group(s).**

4. **Describe any on-going involvement the group(s) consulted has / have in the project.**

5. **Describe how information will be disseminated to participants and the group(s) consulted at the end of the project.**

SECTION G: OTHER CULTURAL ISSUES

1. **Are there any aspects of the research that might raise any specific cultural issues, other than those covered in Section F?** (If YES, explain. Otherwise go to Section H) **NO**

2. **What ethnic or cultural group(s) does the research involve?**

3. **Identify the group(s) with whom consultation has taken place, describe the consultation process, and attach evidence of the support of the group(s).**

4. **Describe any on-going involvement the group(s) consulted has / have in the project.**

5. **Describe how information will be disseminated to participants and the group(s) consulted at the end of the project.**

SECTION H: CLINICAL TRIALS

1. **Is this project a Clinical Trial?** (If YES, complete section, otherwise go to Section K. If YES, attach ACC Form A or B – see Guidelines **NO**
2. **Is this project initiated by a Pharmaceutical Company?** **NO**
3. **Are there other NZ or International Centres involved?** **NO**
4. **Is there a clear statement about indemnity?** **NA**
5. **Is Standing Committee on Therapeutic Trials (SCOTT) approval required?** **NO**
6. **Is National Radiation Laboratory approval required?** (Attach) **NO**
7. **Is Gene Therapy Advisory Committee on Assisted Human Reproduction (NACHDSE) approval required?** **NO**

SECTION I: RISKS AND BENEFITS

1. **What are the possible benefits to research participants of taking part in the research?**
   i) All the participants are have experience using AOCE in design and development of software. Participating in this study will encourage them to think critically and reflect upon the work that they have done. They will also be able to make better judgement as to what to look for when designing and developing software.
   ii) The prototype testing software is based on the same programming environment as is used in many statistical organisations. Participants may be motivated to see research in a closely associated field.

2. **What are the possible risks to research participants of taking part in the research?** (Make sure that you have clearly identified /explained these risks in the Participant Information Sheet(s). Nil

3. a) **Are the participants likely to experience discomfort (physical, psychological, social) or incapacity as a result of the procedures?** (If YES, describe, and explain them clearly in the Participant Information Sheet(s) **NO**

   b) **What other risks are there?** Nil

   c) **What qualified personnel will be available to deal with adverse consequences or physical or psychological risks?** (Explain in the Participant Information Sheet(s). NA

SECTION J: FUNDING

It is expected that all funding will be mentioned in the Participant Information Sheets.

1. **Do you have or intend to apply for funding for this project?** (If YES, complete this section and acknowledge it in the Participant Information Sheet(s)), otherwise proceed to Section J) **NO**

2. **From which funding bodies?** NA

3. **Is this a UniServices project?** (If YES, what is the project reference number?) **NO**

4. **Explain investigator's and /or supervisor's financial interest, if any, in the outcome of the project.** Nil

5. **Do you see any conflict of interest between the interests of the researcher(s), the participants or the funding body?** (If YES, describe them.) **NO**

SECTION K: HUMAN REMAINS, TISSUE & BODY FLUIDS

(SECTION K: Not Applicable)

1. **Are human remains, tissue, or body fluids being used in this research?** (If YES, complete this section otherwise go to Section L) **NO**
2. **How will the material be taken?** (e.g. operation, urine samples, archaeological digs)

3. **Will specimens be retained for possible future use?** (If YES, explain and state this in the Participant Information Sheet(s) **YES / NO**

4. **Is material derived or recovered from archeological excavation?** (If YES, explain how the wishes of Iwi and Hapu (descent groups), or similar interested persons, or groups, have been respected?) **YES / NO**

5. **Where will the material be stored, and how long will it be stored for?**

6. a) **How will the material be disposed of?** (If applicable)

   b) **Will material be disposed of in consultation with relevant cultural groups?** **YES / NO**

7. Is the material being taken at autopsy? **YES / NO**

If YES, provide a copy of the information to be given to the Transplant Coordinator, and state the information that the Transplant Coordinator will provide to those giving consent. Indicate how the material will be stored / disposed of, and explain how the wishes with regard to the disposal of human remains of the whanau (extended family) or similar interested persons will be respected.

8. Is blood being collected? **YES / NO**

(If YES, what volume at each collection, how frequent are the collections, and who is collecting it?)

a) Explain how long it will be kept and how it will be stored.

b) Explain how it will be disposed of.

SECTION L: OTHER MATTERS

1. The Committee treats all applications independently. If there is relevant information from past applications or interaction with the Committee, please indicate and append.

2. Have you made any other related applications? (If YES, supply approval reference number(s). **NO**

3. Are there any other matters you would like to raise that will help the Committee review your application?

We need to evaluate a development methodology. No one will volunteer to develop whole systems just for evaluating this methodology because too much time and effort is involved.
These are students who have prior experience using this development methodology (AOCE) and have some limited experience with similar systems (though they were mobile applications). As such they are the ideal candidates for this evaluation. Also we have to approach them fast before some of them leave the university/country.

----END OF APPLICATION FORM----

Department of Computer Science
Level 3, Science Centre
Building 303, 38 Wellesley St
Auckland
Phone 3737599 ext 88297

PARTICIPANT INFORMATION SHEET – PARTICIPANT RECRUITMENT

Project title: **Supporting Web Services Systems Specification using Aspect-Oriented Component Engineering**
Researcher name: Santokh Singh

To: Students

My name is Santokh Singh and I am a Ph D (Computer Science) student at The University of Auckland conducting research into supporting web services systems specification using Aspect-Oriented Component Engineering (AOCE). I am conducting this research to explore the usability of the AOCE methodology to develop novel aspect-oriented web services systems. As a student who has developed software using AOCE you have been approached to participate in this study and I would appreciate any assistance you can offer me. Your participation is voluntary and not part of your assigned tasks as a student.

As a participant in this evaluation your feedback will be recorded and analyzed in order for us to achieve the best possible survey design experience. The goal of the thesis is to research and propose a new breed of novel dynamic aspect-oriented web services system that is better characterized and categorized, and then design, develop and provide support for such systems by extending and applying the Aspect-Oriented Component Engineering, (AOCE) methodology. There is a usability testing component part to the project which requires human participants.

You may, voluntarily, answer the questionnaire in your own spare time or come to the computer science department so that we can arrange a suitable place and time for you to answer the questions. You will not be paid for answering the questionnaire and any additional expense for travel and food will not be reimbursed.

Participants just need to answer the questions in a questionnaire form that will be given to them. The whole procedure will take no more than one hour.

Choosing either to participate, or not to participate in this study will not influence any of your academic evaluation at the University of Auckland.

A questionnaire will be provided for you to fill in as part of the usability evaluation. The completed questionnaire will be held in secure storage within the Department of Computer Science at the University of Auckland for six years and then destroyed by shredding. The individual questionnaire responses will be summarized and analyzed and this summary information may be used both to improve our research outcomes that we are developing and report on the findings of the study. A summary of the results of the survey and any resulting publications will be made available to you on request

Given the specialized nature of this field and the limited number of people who have used the AOCE methodology, there may be a possibility that someone may identify the group of students involved (in answering the questionnaire) as a whole but they may not be able to identify individuals.

All personal information will remain strictly confidential and no material that could personally identify you will be used in any report on this study

Researcher name and contacts	**Supervisor name and contacts**	**HOD name and contacts**
Santokh Singh santokh@cs.auckland.ac.nz 09 373-7599 ext 82283 Address: Department of Computer Science Level 3, Science Centre Building 303, 38 Wellesley St, Auckland	John Hosking john@cs.auckland.ac.nz 09 373-7599 ext 88297 Address: Department of Computer Science Level 3, Science Centre Building 303, 38 Wellesley St, Auckland	John Hosking john@cs.auckland.ac.nz 09 373-7599 ext 88297 Address: Department of Computer Science Level 3, Science Centre Building 303, 38 Wellesley St, Auckland

For ethical concerns contact: The Chair, The University of Auckland Human Participants Ethics Committee, Office of the Vice Chancellor, Research Office, Level 2, 76 Symonds Street, Auckland. Tel: 373-7599 extn. 87830.
APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE ON …(date)... TO …(date)…FOR ……(3) YEARS REFERENCE NUMBER 200../…