Developing .NET Web Service-based Applications with Aspect-Oriented Component Engineering

Santokh Singh[1], John Grundy[1, 2] and John Hosking[1]
Department of Computer Science[1] and Department of Electrical and Computer Engineering[2]
University of Auckland
Private Bag 92019, Auckland, New Zealand
{santokh|john-g|john }@cs.auckland.ac.nz

Abstract

Current approaches to engineering web service-based software systems are limited by lack of comprehensive web service component characterisations. In this paper, we describe our recent work applying Aspect-Oriented Component Engineering (AOCE) to develop more adaptable, higher quality and more reusable software components for web services. We give examples on how AOCE can be used to design and implement Web Service-based systems, including aspect-oriented enhancements to the Web Services Description Language and the Universal Discovery, Description and Integration standards. We describe a prototype web services system we have implemented using AOCE to demonstrate and evaluate our technique.

1. Introduction

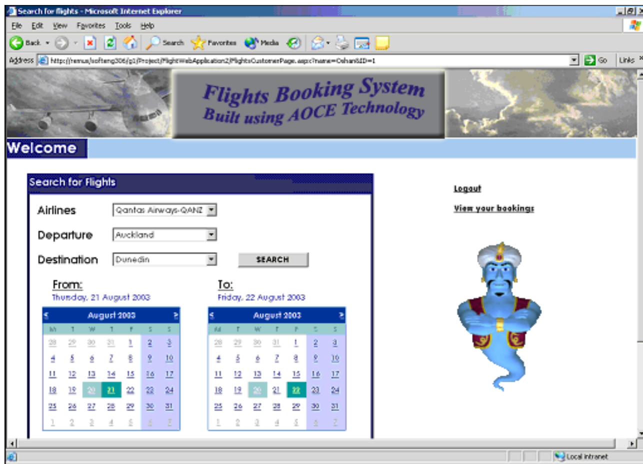
Web services have quickly become a very important enterprise system development technology. Numerous commercial and non-commercial organizations want to use them to enhance their IT systems' ability to be extended and integrated with internal and external systems [1, 7]. Web services promise to enable business to business integration seamlessly and dynamically irrespective of platform, language or culture [4, 17]. The number of organizations making use of web services is predicted to increase tremendously in the next few years**.** Key features of web service-based systems are the self-description of web service components via Web Services Description Language (WSDL) and the aim of dynamically discovering new web service components to integrate at run time via Universal Discovery, Description and Integration (UDDI) [7, 19].

Presently component-based systems engineering, including that for web service-based systems, tends 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 difficult to both understand and combine [9, 10]. Most current development approaches also make many assumptions about other components related to a web service, constraining its reuse. Furthermore the component documentation is too low level, focusing on component interface and message types, which again is hard to understand at higher levels. As web services are a relatively new and still a growing technology, there are still a lot of unanswered issues about web services design and implementation, including those relating to security, performance, collaboration and interoperability [1, 4]. Aspect-oriented component engineering for web services may provide an answer to overcome such limitations.

We have been carrying out extensive research into developing Aspect-Oriented Component Engineering [9, 10, 11] for software component design and development. This approach uses aspects to better-describe functional and non-functional characteristics of software components than conventional techniques. In this paper we describe our work using AOCE to design and implement Microsoft™ .NET-based web services. We firstly motivate this research with a web service-based example application, and survey recent work in the area. We describe how we use aspects to design and characterise web service components, and to provide improved support for discovering, validating and integrating web service components at run-time. We give examples of applying this approach to our example application, briefly outline its implementation using aspects, and evaluate our approach's strengths and weaknesses, identifying areas for future research.

2. Motivation

In this paper we describe our investigation into applying a new methodology, Aspect-Oriented Component Engineering [10, 11], to web service-based component development. The motivation behind this is that current approaches to web service component-based systems engineering focus more on low-level software component interface design and implementation. This has the problem of being both cumbersome and difficult to comprehend [12, 3]. This problem is even more glaring and evident when large scale developments or maintenance of web service-based systems are undertaken.



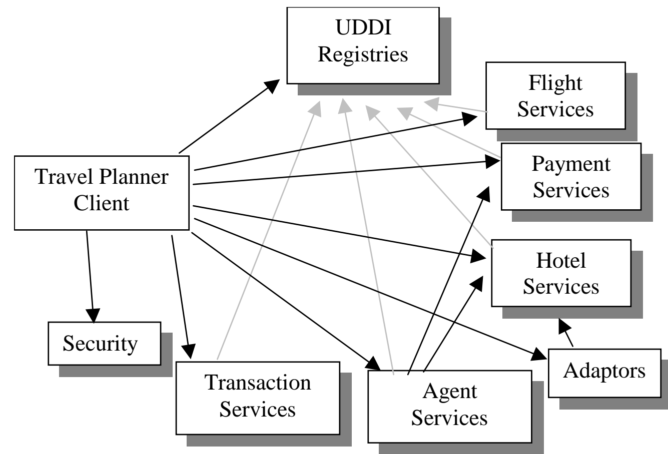


Figure 1. An example web services-based application.

Consider a commonly-used example of a web service-based application, a travel planner [8]. Figure 1 (a) shows a Travel Planner client user interface, which provides an interface to remote Flights Booking web services. The client has been implemented using C#.NET, and includes a range of interfaces for managing travel itineraries, finding and booking travel-related items (flights, hotel rooms, rental cars etc), and active help via “agents” (such as the “blue genie” on the right).

The architecture of the travel planner is illustrated in Figure 1 (b). The travel planner is composed of multiple web service components rather than a single monolithic application. The travel planner client needs to dynamically discover available flights, hotels, payment, agency, transaction co-ordination and other remote services at run-time. These are integrated with the client to provide the overall application functionality. Some components may require adaptors to allow the travel planner to communicate with them with its known protocols e.g. some hotel services. Others may require multiple composite components to provide required functionality, such as the agency using other payment and car booking remote web services. Some components provide “horizontal services”, such as transaction co-ordination and security. The client discovers the available services via one or more UDDI registries, and may be able to select between multiple available services providing the same functionality [17, 20].

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 them?
- How can web services be advertised to other components with enough information that useful discovery algorithms can be provided?
- How can it canbe validated that discovered components meet advertised characteristics at run-time?
- How can adaptors to components be discovered or synthesised and composite component aggregates be found and initialised?

We have been conducting research on existing methodologies for developing complex web services systems and found shortcomings with them. OMG’s Model Driven Architecture (MDA) [18] defines software at the model level, expressed in OMG’s standard Unified Modeling Language (UML). The MDA application's base model specifies details of its business functionality and behavior in a technology-neutral way called the application’s Platform-Independent Model. An intermediate product called a Platform-Specific Model (PSM) and it reflects non-business, computing-related details e.g. affecting performance and resource utilization that are added to the Platform-Independent Model by the web services’ architects. Though it’s a commendable development methodology that can be applied to large scale development of web service-based systems, it has a number of disadvantages. All of the techniques used in MDA tend to focus on lower-level features of the system. Other Component-Based Development methodologies include the COMO approach [14], The Select Perspective™ [2] and The Catalysis™ approach [6]. These also focus on the provided features of components and not component requirements and inter-component relationships. This can make designs hard to understand at abstract levels or during reengineering and refactoring processes. Higher level systemic component descriptions such as persistency, user interfaces, security, transaction processing, performance etc. are all missing from this approach. These high-level features are very important for understanding and using systemic components and their functionalities, especially in the context of large and complex systems.

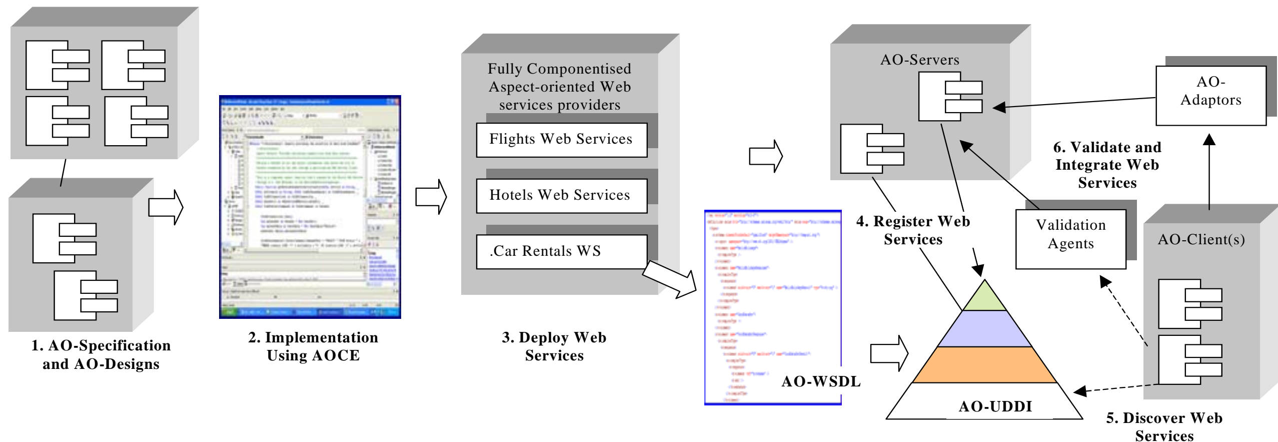


Figure 2. An overview of how we use aspects to develop .NET web service-based systems.

3. Our Approach: AOCE.NET

Aspect-Oriented Component Engineering (AOCE) is a component development methodology that uses aspects to characterise and categorize different system cross-cutting capabilities and to reason about inter-component services [10, 11]. These aspect-characterised services, such as data persistency, message distribution, transaction co-ordination, security, user interfaces and system resource utilisation [15, 13], can either be provided by a software component or required by it in order to operate. If a service is required, it can be acquired from other components that provide such a service in compatible ways. AOCE supports the identification, description and reasoning about high-level component functional and non-functional requirements grouped by such systemic aspect categories. We have applied this in past work to CORBA and RMI-based remote object systems, and done preliminary investigation of its applicability to web service-based components [11, 8, 9].

In Aspect-oriented Component Engineering a software component is characterised by the systemic aspects it provides services to support or requires services from another component in order to operate [11]. In previous work we have used this approach to effectively engineer components for a range of domains, including adaptive user interfaces, plug-and-play collaborative work components, and various enterprise systems [9, 11, 8]. We have used aspect characterisations to describe all components in an application.

In a web services-based architecture, such as provided by Microsoft™'s .NET technology, a system is made up of multiple components. Some of these are remote functions ("web services") accessed typically by standardised communications technologies, such as XML-based Simple Object Access Protocol (SOAP) across HTTP or similar transport technologies [4]. When applying the aspect-oriented component engineering concepts to web services we are able to characterise web services and their clients by the aspect-categorised services they provide and/or require. We also wanted to experiment with enhancing web service component descriptions to enhance their discovery and dynamic integration within an architecture at run-time.

Figure 2 shows the main steps in applying the AOCE methodology to .NET-based web service-based system development. Aspect-oriented specifications and designs capture the functionality of desired web service clients and service components. Aspects are used to identify functional and non-functional characteristics that a web service or client provides or requires for operation. Our .NET web services are then implemented with the need for them to be run-time discovered, validated and integrated, and to be characterised by aspect-enhanced service descriptions. Clients are implemented with the need for them to run-time discover their required services using aspect-enhanced web service characterisations as well as conventional web service category and provider descriptions. Each web service in our approach has an extended, Aspect-Oriented Web Service Description Language (AO-WSDL) description characterising the service's functional and non-functional properties. Web services are deployed and their descriptions registered with an extended registry mechanism, Aspect-Oriented Universal Discover, Description and Integration service (AO-UDDI).

At run-time, a web service client issues one or more aspect-enhanced queries to the AO-UDDI tool. This locates matching web services, using the standard UDDI information to locate category matches and the aspect enhancements to locate and rank functional and non-functional matches. In order to communicate with a discovered service, the client may recognise its protocol, but sometimes may need to locate or even synthesise an appropriate adaptor to access it. Aspect enhancements are

used to assist in discovering or building appropriate service adaptors. At times multiple components may need to be located and used to match a single client query, and these composite web service component aspects must be checked. When located, remote services may need to be dynamically tested to ensure they meet advertised characteristics e.g. performance, security protocols and transactional behaviour. We use aspect characterisations to support dynamic testing agents running validation checks on discovered web services.

4. AOCE.NET Travel Planner Example

We have applied AOCE to the development of our example travel planner application design, implementation with .NET web services, and dynamic discovery and integration of web services the client application uses.

4.1. Web Service Design and Implementation

Figure 3 shows an example of a high-level AOCE web service component diagram for part of the travel planner system. Each component is represented as a traditional UML class, and in this example the focus is on the Booking client and related components. This component design diagram shows both software components making up the application and various systemic aspects that are cross-cutting the different components. The search and booking interfaces use middleware (web services) to access a remote travel itinerary manager service, along with a remote payment service payment and distributed transaction co-ordinator. A diamond sign shaped aspect annotation indicates that the aspect provides crosscutting information for another component while a square one requires it from another component. In this format, an indication of the functional and non-functional aspectual properties is attached to each visual component. More detailed views and associated component properties specify provided and required aspect details of each web client and service. Our approach is different from traditional Aspect-Oriented Programming systems [15, 13] in that the aspects themselves are not separated out into their own modules. Our aim is not necessarily to inject aspect-implementing code into modules but to better-characterise cross-cutting features of our web services-based software components.

Each aspect-characterised feature of these web service components has additional constraint specifications ("aspect detail properties"), such as the required persistency mechanism, data size limits, event-based or data-based distribution mechanisms, caching and synchronisation mechanisms, required data transmission performance, transaction co-ordination demarcation, and so on. For example, in Figure 3, the MakeBooking component requires security, data persistency and transaction processing aspects. As shown by the diamond symbols and matching patterns, the TransactionCo-ord component provides the security and transaction co-ordination MakeBooking requires. The PaymentManager component provides distribution and security support and the TravelItemsManager data persistency. These aspects are required by the MakeBooking component to enable it to function in this architecture.

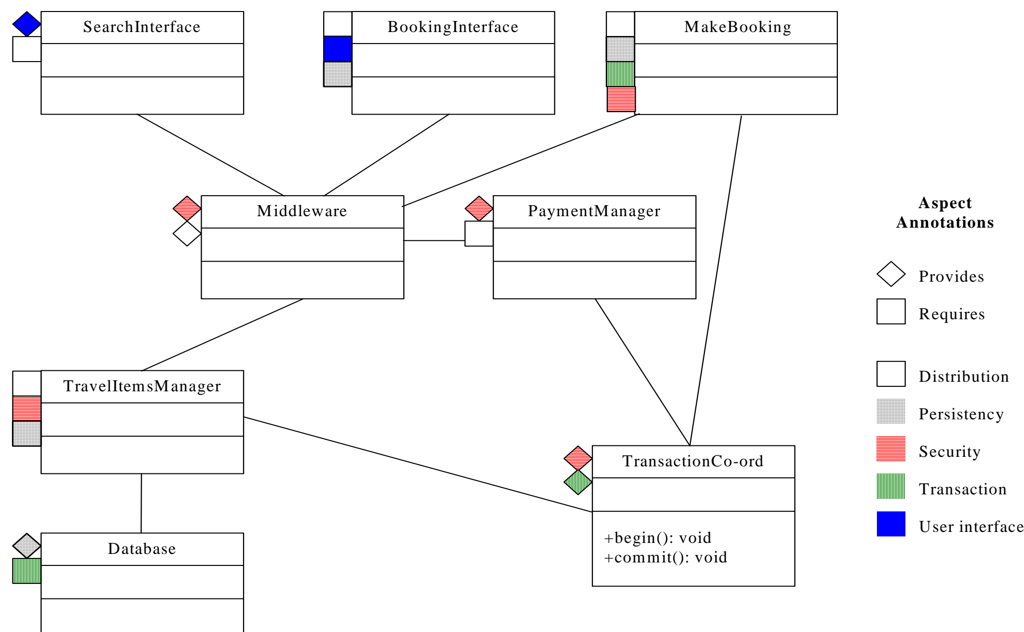


Figure 3. An AOCE design for travel planner components.

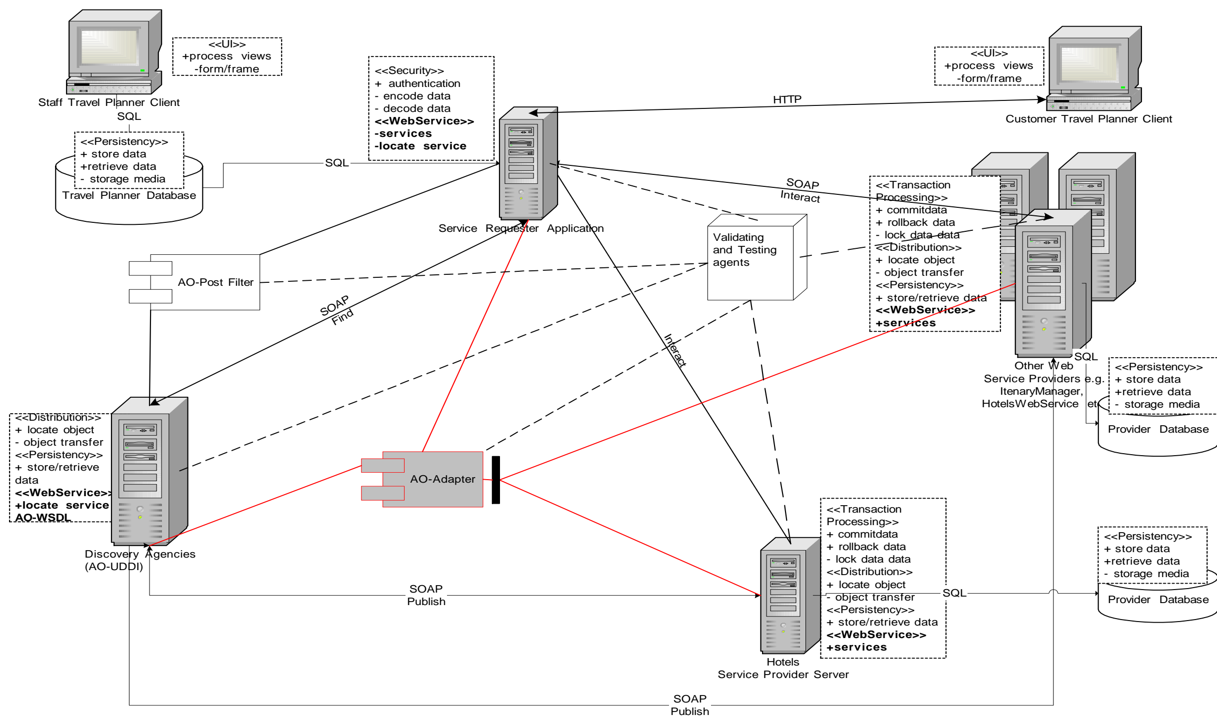


Figure 4. The travel planner's AO-web service-based architecture.

The software architecture of our collaborative Travel Planner is shown in Figure 4. This example uses a more detailed textual representation of cross-cutting aspects to annotate the architecture structure. In this figure, aspects are placed between double angled brackets, e.g. <<Persistency>>, <<Security>> and <<Distribution>> shown in the figure stand for persistency, security and distribution aspects respectively. All aspect-details pertaining to a particular aspect are listed immediately below the aspect itself. Also aspect-details have a "+" or "-" symbol preceding them. A "+" sign means that the aspect details are provided by the component while a "-" signifies that it is required. The software components that make up each subsystem/application expose interfaces that relay information about these aspect-oriented functions. These interfaces are implemented within the component that contains them. They are used by other aspect-oriented components that are assembled together to build the software application.

Using such aspect-oriented component designs, we implement .NET web service components that:

- are self-describing not only in terms of their interfaces but include aspect characterisations for richer run-time understanding and configuration;
- at run-time are able to locate web service components that provide their required functional services as specified by their aspect characterisations;
- and may make use of "standardised" aspect-based component adaptors to interact with discovered web services in a de-coupled manner, without needing any hard-coded type or behavioural information about the component.

4.2. AO-WSDL

Standard web service components are characterised by the Web Service Description Language, an extensible XML-based mechanism for describing the low-level interface features of the provided web service [4]. Web service requestors will only access, consume or integrate with the web service if it is understandable and exposes services that satisfy the requirements of the clients. We have incorporated aspect-oriented elements into the WSDL document to increase understandability of the web service at run-time. These aspectual information enhancements allow for easier, more dynamic and automated systems integration by allowing queries for web services to filter inappropriate services out using their aspect information. They also allow for more precise run-time integration to remote services and support automated testing of

discovered web services to validate they meet aspect-specifiedc constraints.

WSDL itself has 6 major elements that are extensible i.e. for definitions, types, message, portType, binding and service. It also has other optional utility elements, e.g. for documentation and import purposes. We have added a number of aspectual extensions, all neatly bundled into a major “aocomponents” element, to the standard WSDL format as shown in figure 5. This 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. Figure 5 further gives a summarised description and overview of the 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. AO-WSDL also allows for more dynamic and automatic searches for any given aspect, aspect details and properties of the services advertised because our AO-WSDL specifications for web service components follow consistent, formal and clearly defined semantics and syntax.

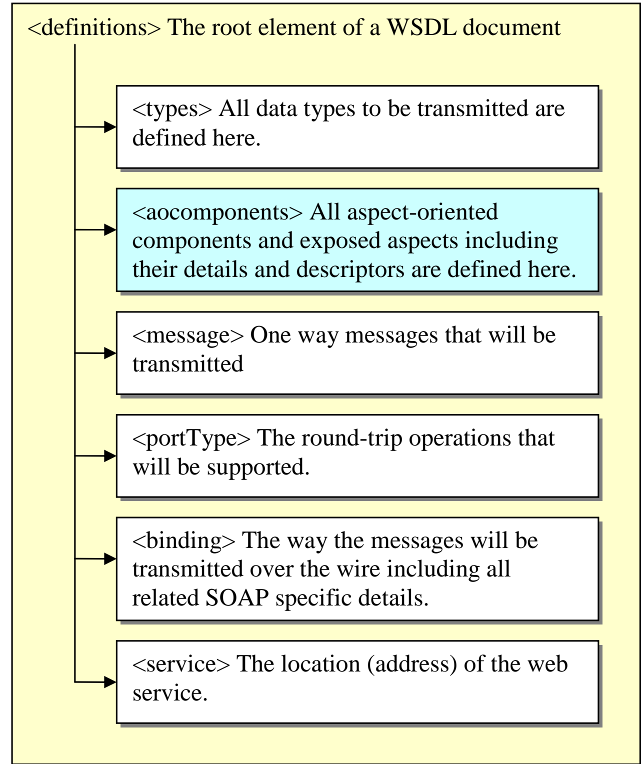
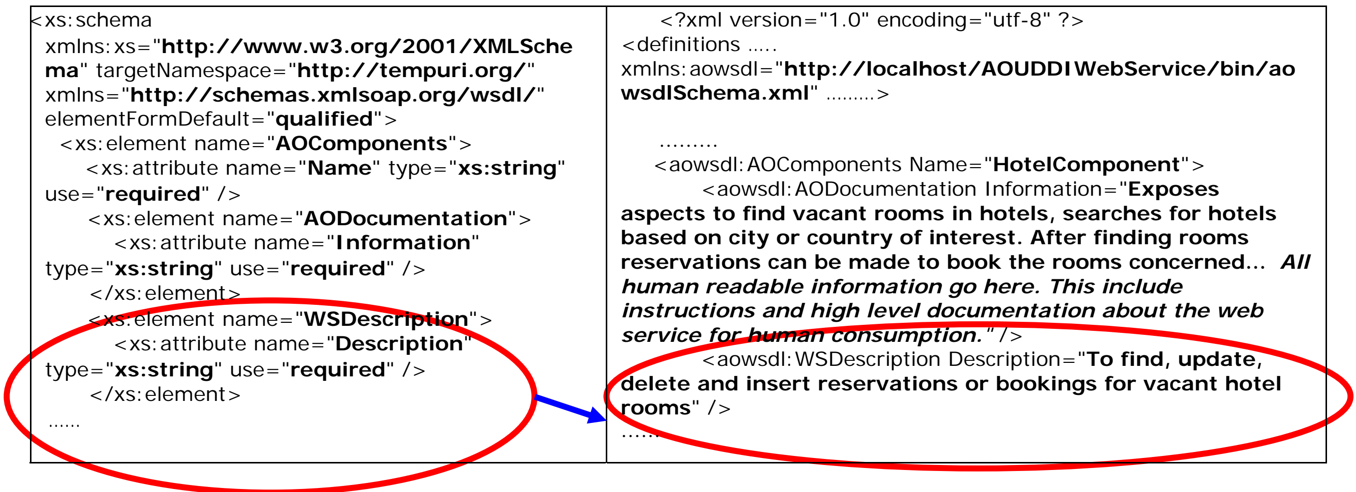


Figure 5: AOWSDL document showing the hierarchy of its elements.



```
<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>
 ......
```

```
   <?xml version="1.0" encoding="utf-8" ?>
<definitions .....
xmlns:aowsdl="http://localhost/AOUDDIWebService/bin/aowsdlSchema.xml" .........>

   .........
   <aowsdl:AOComponents Name="HotelComponent">
       <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" />
......
```

Figure 6. (a) the initial section of the AO-WSDL schema; and (b) implemented part of this section in the AO-WSDL from the travel planner.

Figure 6 shows the initial section of the AO-WSDL schema and an example of its use in the AO-WSDL document for the HotelComponent. All our 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”, gives instructions used for automatic discovery and integration of the web service.

```
  ...........
    <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>
   ...........
```

```
.......
    <aowsdl:Component ComponentName=
 "HotelComponent ">
      <aowsdl:ComponentDescription Description=
 "Component to find hotels in various cities
 and countries including rooms availability" />
            <aowsdl:Aspects>
              <aowsdl:FunctionalAspects>
                <aowsdl:Aspect Type="Persistency"
                    AspectName= "StoreData "
                    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" />
```

Figure 7. (a) Components and aspects from the AO-WSDL schema; and (b) corresponding elements in AO-WSDL.

Each component of the web service provider is nested within the “AOComponents” element as shown in Figure 7. 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 7, and the clear and concise language used allows automatic querying to be possible.

Each component exposes one or more aspects. Each aspect element describes details about a particular cross-cutting features impacting the component e.g. persistency, transaction processing, security, resource utilisation, etc. An aspect describes a functional or non-functional type of cross-cutting concern. Each aspect has an aspect type associated with it, e.g. the aspect type could be Persistency, Distribution, Transaction, Security etc, categorising the cross-cutting concern.

```
<xs:element name="Aspect">
...........
    <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: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" />
............
```

```
 <aowsdl:Aspect Type="Persistency"
AspectName="PersistenceHotelsDataSetfromCityCountry"
WSEntryPoint="true" Standalone="true">
..............
     <aowsdl:AspectDetail Type="data retrieval"
Detail="select" Provided="true" />
       <aowsdl:AspectUserOperations
UsedBy="Persistence_HoteFinder|
            TransactionProcessing_ItenaryManager" />
       <aowsdl:UsesOperations
Uses="Persistence_roomsByHotelID|Persistence_OnSiteF
acilities|Persistence_OffSiteFacilities|Persistence_places
OfInterest" />
   </aowsdl:Aspect>
......
```

Figure 8. (a) Aspect details in the AO-WSDL schema; and (b) corresponding elements from the AO-WSDL document in the collaborative travel planner.

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 8, 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.

4.3. AO-UDDI

A key feature of web services-based architectures is the dynamic discovery, adaptation and integration of distributed web service components. The Universal Description, Discovery and Integration (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 [1, 17]. Unfortunately current web services technologies such as .NET provide only limited, low-level support for dynamic integration, with much current research attempting to improve the state-of-the-art [16]. We have used our aspect-oriented descriptions of web service components to provide improved support, developing an Aspect-oriented UDDI (AO-UDDI) registry tool. Each web service is assigned a unique identifier when it is registered with our AO-UDDI. All aspect-oriented features and related information about the web services together with this identifier is stored in databases with local caches of it available for speedier searches and discoveries.

In a conventional UDDI-based system a web service client requests matching web services based usually on standard, function-oriented categories [20]. In our AO-UDDI registry, aspect characterisations can be used both to inform category choices and also 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 9 illustrates these features with an example from our .NET travel planner application. The travel planner client issues a query to an AO-UDDI 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 AO-UDDI registry locates registered web services matching the category query (flight booking service), filters and/or 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 AO-UDDI 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).

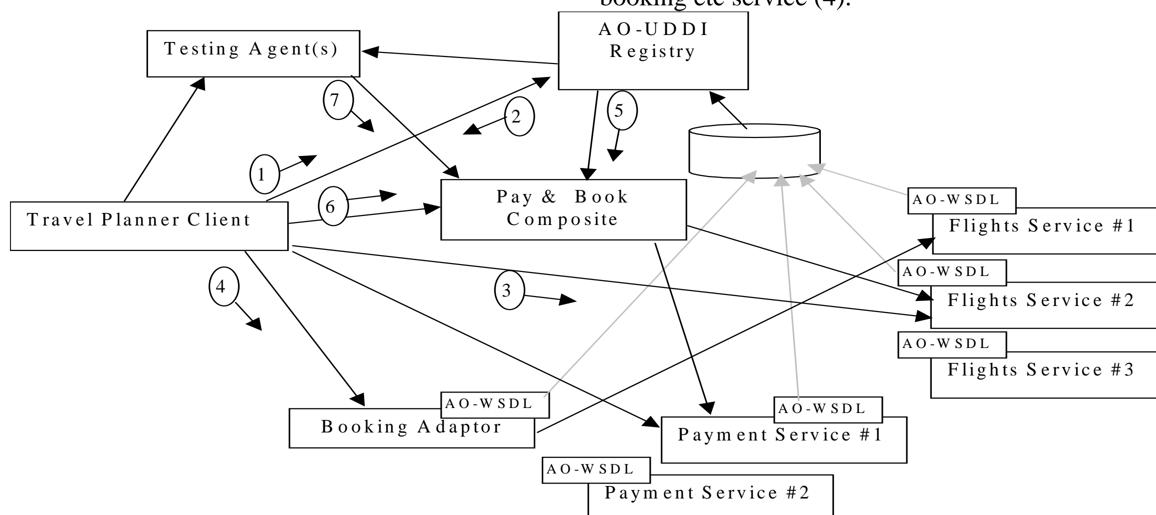


Figure 9. An aspect-enhanced UDDI query example for the travel planner.

A more complex query to the AO-UDDI registry by the client might ask for a combined flight booking and payment service. Such a service doesn't exist, but the AO-UDDI 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) are used to verify the accuracy and performance of the web service component responses to generated queries. These agents use the aspect descriptions in AO-WSDL documents to synthesise requests to the discovered services and check that they meet the client-specified criteria e.g. performance of remote service, message size used by remote service, security approach of remote server and so on.

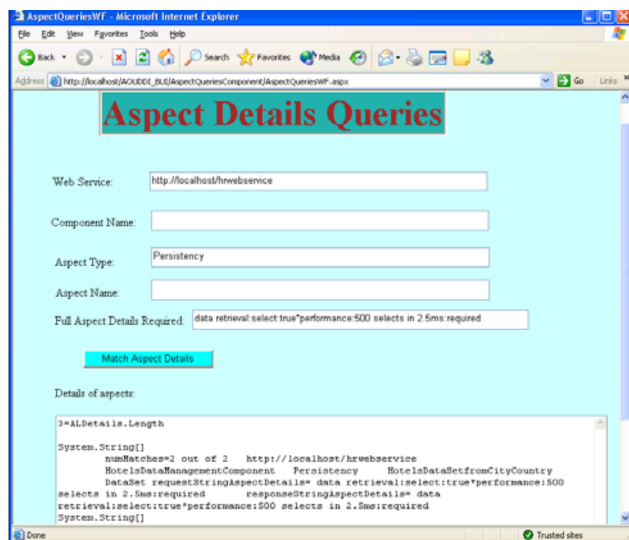


Figure 10. Example of using the AO-UDDI registry.

The AO-UDDI registry can be accessed by client components as shown in Figure 9. It can also be used to manually select and choose between available services by an application user. The user interface of this AO-UDDI registry client is shown in Figure 10. Here the user is performing an aspect-based query, seeking data management components that provide a specified performance criteria.

5. Design and Implementation

We implemented our travel planner prototype system using standard C#.NET web service components. Web service clients were implemented so that they do not hard-code any remote service information but instead use our extended AO-UDDI mechanism to locate one (or more) components that satisfy their required services. Web services were implemented so that they are dynamically located and integrated by clients. Web service components can be run-time tested by dynamic validation agents to ensure that they meet their aspect characterised performance and other constraints in their actual deployment situation. A number of adaptor components were implemented that allow a web service client to interact with discovered web services without direct knowledge of their SOAP protocols and behaviour, instead using standardised, aspect-categorised adaptor messages to interact indirectly with it.

We extended the WSDL grammar by enriching it with aspect-oriented features so that it becomes better characterised and categorised. All these additional features were extracted directly from our web service source files using .NET's Reflection classes and parsing the web services files with .asmx (i.e. .NET files supporting web services and capable of providing aspects over the wire) extensions. The parsing ensures that all the documentation and descriptors can be extracted from comments in the source files and inserted into the AOWSDL file. We had resorted to consistent and coherent naming conventions throughout our development process and this made it easier to comprehend and extract aspectual information from our source code. AOWSDL as such could be automatically generated from our web services that were implemented by using AOCE techniques. Furthermore, clients not supporting the aspectual features inherent in AOWSDL can still use it as a standard WSDL file by parsing it and ignoring all the aspect-oriented elements.

We designed and implemented an AOUDDI that could execute dynamic discoveries of .NET web services through the use of XML queries. An AOUDDI user interface was also constructed that could be used manually to discover the web services. Further queries specifying detailed aspectual characteristics that are required can be made to the AOUDDI. The AOUDDI will discover and return all matching web service providers located.

Tests were performed by validating agents to gauge how accurately the web services returned by the AOUDDI satisfied and matched the queries. The best matching web service provider was found to have more functionalities than requested for by the web service clients. These web services can be integrated either automatically or manually with the web service requestors. Automatic integration can be achieved by using the AOUDDI to integrate the best matched web service provider found with the web service requestor. Otherwise the user has the option of choosing from the list of all matching web service providers returned by the AOUDDI and manually integrating them.

6. Discussion

Designing and developing the Travel Planner application gave us extensive hands-on experience in applying and further refining the Aspect-Oriented Component Engineering methodology. In earlier versions

we had experimented with the Sun Microsystems' Java programming language to build simple prototypes but without AOUDDI and AOWSDL functionalities. We also developed a version of the travel planner with standard C#.NET without aspect-oriented design or WSDL and UDDI enhancements.

In this paper we described a revision of the travel planner design and implementation, all developed the using Microsoft's .NET technology and our aspect-based enhancements to component design and web services technologies. The Visual Studio IDE that we used allowed us to do rapid prototyping with AOCE for our collaborative web services based Travel Planner. This IDE has a comprehensive library of APIs and easy to use advance features, eg. property boxes with extensive and clearly defined fields to manipulate UI components; drag and drop features for connecting to databases etc.

We achieved a higher level of characterisation and modularisation in our travel planner system designs and implementations by using the AOCE methodology to build our web services-based travel planner system. We found that this also brought about an increased understanding of the interrelationships between the various components concerned. Compared with previous prototypes of our travel planner design, this aspect-enhanced design we found to be much more easily understood and extended than our previous non-aspect-based component designs [9]. In our earlier prototype not using AOCE techniques the travel planner web service components lacked any reflection of their inter-relationships with other components and the cross-cutting concerns that impact them. When showing the designs and web service interface descriptions to other developers we had to explain this information using annotated UML diagrams, free text or verbal explanations. By capturing the cross-cutting concerns using our AOCE component aspects for the travel planner web services we found that these additional techniques were no longer required or could be substantially simplified. Feedback from other developers on our extended designs and web service characterisations have indicated that they substantially enrich the knowledge of a web service over conventional UML diagrams and WSDL and UDDI technologies.

Our AOUDDI extended web service registry was better equipped to handle queries pertaining to aspects and their details than the standard UDDI registry and previous aspect-enhanced registries of our own [1, 10]. The AOWSDL also provides a richer characterisation of web service components than its WSDL counterpart. The extensions in our AOWSDL documents allow for better dynamic discovery of web services at run-time as they incorporate both non-functional characteristics and use a standardised functional categorisation of component services. The major overhead here is that there are a lot of descriptors used in AO-WSDL to define the aspects, their details and the services that they provide. But this is easily offset by the advantages gained by using the aspect-oriented features in AOCE.

On the downside, as there are no universally accepted standards in the terminology and notations used in AOCE by the various interested parties trying to use it, the result could be an extra initial cost in terms of developers' time and effort to understand other peoples' designs and code. We are working on creating some universally accepted standards for AOCE in these respects so as to streamline our efforts with others who are using this methodology. We are also very interested in developing tools that can do more comprehensive modelling and designs to support development using AOCE techniques. The existing modelling tools, e.g. Rational Rose and Microsoft's Visio, do not support visual representations of aspects in their models. Also, a lot of textual descriptions need to be inserted into the existing design diagrams to describe aspectual features and we hope to overcome this limitation in the tools we are building. Currently we have to build our AO-WSDL descriptions by hand, and this is very time-consuming and error-prone. Generation of AO-WSDL descriptions from a design tool would greatly enhance this.

One issue that can cause problems with our approach is if a web service lacks specification of certain characteristics using aspects in its AO-WSDL description, or a third-party web service only has a conventional WSDL specification. In these circumstances we allow the web service to be integrated back can not do some of the things supported by our approach e.g. run-time validation of discovered service; filtering of service using aspects using AO-UDDI etc. However, our approach does conceptually allow a third-party or partially aspect-enhanced service to be further characterised by additional AO-WSDL information. This aspect-enhancement of existing 3rd party web services is an interesting future research direction we wish to pursue.

Our future research plans also include building better tools to generate AO-WSDL. The AO-WSDL will also be upgraded so that it can support and describe more aspectual features. Our aspectual features are very important and critical in providing accurate and appropriate responses, especially to AO-UDDI queries by web service requestors. These additional aspectual features would also be used in our post-filter devices. We are also looking at expanding our automated validation tools. We are also working with intelligent agents to help us with our automatic dynamic lookup, interpretation, translation and integration tasks involving aspect-oriented web services. Another area of interest to us is extending AO-UDDI to understand queries in natural language. .NET provides a variety of extended features that may be used to help support aspect-oriented web service component development. These include class attributes, a powerful reflection mechanism supporting run-time code injection

into class implementations, and a class meta-data management facility. We would like to explore how these could be used to provide other aspect-based support features e.g. run-time code injection for web service implementations. We see this as being complementary to the web service interface characterisations that we have extended with our AOCE for web services. Finally it would be advantageous to integrate all these additional features and tools into the Visual Studio .NET IDE. This will save much time when developing web service systems using AOCE techniques because developers would be able to create aspect-oriented components and sub-systems faster.

7. Summary

AOCE provides a new framework for describing and reasoning about component capabilities from multiple aspect-oriented perspectives. Aspect information in component implementations allows developers, end users and other components of web services to access high level knowledge about component capabilities. We have extended and enriched the web service description language into AO-WSDL documents that contain aspect-enriched descriptions. These extended descriptions can be indexed by AO-UDDI registries and the aspect information used to assist better description, discovery, testing and integration of web service components.

We successfully applied this methodology to develop a web service based collaborative Travel Planner system composed of a number of web service-based software components. AOCE for web services brought about an increase in reusability, reconfigurability and understandability during the development process. We were also able to consume web services exposing aspects in WSDL files. Our AO-UDDI was used by client components to dynamically locate available services or to find adaptors or composite services. The combination of AO-WSDL and AO-UDDI used in conjunction with web services supports more powerful dynamic service discovery, validation and integration compared to those using conventional WSDL and UDDI techniques. Though we have 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 concepts can be applied equally well to the development of any type of web service-based system.

References

1. Adams, C., Boeyen, S. UDDI and WSDL extensions for Web service: a security framework, In *Proceedings of the 2002 ACM workshop on XML security*, Fairfax, VA , 2002, ACM.
2. Allen, P., and Frost, S. Component-Based Development for Enterprise Systems: Applying the Select Perspective, Addison-Wesley, 1998.
3. Brown, A.W., and Wallnau., K.C. The Current State of CBSE, In *IEEE Software*, Volume 155, Sept-Oct 1998.
4. Cerami, E. Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, Feb 2002, O'Reilly.
5. Chappell, D. and Jewell, T., Java Web Services, March 2002, O'Reilly.
6. D'Souza, D.F. and Wills, A.C. Objects, Components and Frameworks with UML, The Catalysis™ Appproach, Addison-Wesley, 1999.
7. Ferrara, A. and MacDonald, M. Programming .NET Web Services, September 2002, O'Reillly
8. Grundy, J.C., Hosking, J.G., Mugridge, W.B., and Apperley, M.D. Tool integration, collaboration and user interaction issues in component-based software architectures, In *Proceedings of TOOLS Pacific '98*, Melbourne, Australia, Nov 24-27 1998, IEEE CS Press, pp. 299-311.
9. Grundy, J.C. and Hosking, J.G., In Engineering plug-in software components to support collaborative work, *Software – Practice and Experience, 2002*; vol. 32, pp. 983-1013.
10. Grundy, J. Multi Perspective Specification, Design and Implementation of Software Components using Aspects, In *International Journal of Software Engineering and Knowledge Engineering*. Vol. 10, No. 6 (2000) 713-734, World Scientific Publishing Company
11. Grundy, J. and Ding, G. Automatic Validation of Deployed J2EE Components Using Aspects, In Proceedings of 2002 IEEE International Conference on Automated Software Engineering, Edinburgh, UK, IEEE CS Press.
12. Heisel1, M., Santen, T., and Souqui`eres, J., Towards a Formal Model of Software Components, In *Formal Methods and Software Engineering - Proceedings of the 4th International Conference on Formal Engineering Methods.*
13. Kiczales et al, Aspect-oriented Programming, In *Proc. of the 1997 European Conf. on Object-Oriented Programming*, Finland (June 1997), Springer-Verlag, LNCS 124.
14. Lee, S.D., Yang, Y.J., Cho, E.S., Kim, S.D., Rhew, S.Y. COMO: A UML-Based Component Development Methodology, In *Proceedings of the 1999 Asia-Pacific Software Engineering Conference,* 1999, pp 54 – 61.
15. Lieberherr, K. Connections between Demeter/Adaptive Programming and Aspect-Oriented Programming (AOP), http://www.ccs.neu.edu/home/lieber/, 1999.
16. Microsoft, Visual Studio and .NET, http://www.microsoft.com/net/, 2003 Microsoft Corporation.
17. Ran, S. A model for web services discovery with QoS, In *ACM SIGecom Exchanges* , Volume 4 , No 1, 2003.
18. Siegel, J. Using OMG's Model Driven Architecture (MDA) to Integrate Web Services, http://www.omg.org/.
19. Torkelson, L., Petersen, C. and Torkelson, Z. Programming the Web with Visual Basic .NET, SoftMedia Inc 2002.
20. Zhang, L.J. Li, H., Chang, H., Chao, T. XML-based advanced UDDI search mechanism for B2B integration, In *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.