An Architecture for Developing Aspect-Oriented Web Services

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

Abstract

Current web services approaches have many limitations, especially with description, discovery and integration mechanisms. In this paper we present a novel software architecture called aspect-oriented web services (AOWS) which addresses these problems. AOWS uses descriptions of cross-cutting concerns between web services to give more complete descriptions of services, supporting richer dynamic discovery and seamless integration. We describe our architecture, a formal specification of it and an implementation using .NET web services technology.

1. Introduction

Web services are software systems identified by URIs, whose public interfaces and bindings are defined in Web Services Description Language (WSDL) documents [1], [3], [7]. A web service's interface and bindings can be discovered by other systems using Universal Description, Discovery and Integration (UDDI) registries. These systems may then interact with the web service in a manner prescribed by its definition, using XML based messages. Conventional web service design techniques focus on low level component interface design and implementation. This can lead to development of components whose services are difficult to understand and combine [4], [8].

Aspect Oriented Component Engineering (AOCE) is a methodology we have developed that uses aspects to characterize and categorize different systemic cross-cutting capabilities of components and to reason about inter-component relationships [21]. Based on this, we have developed a new approach for describing, discovering and integrating web services-based components. This involves extending the WSDL and UDDI mechanisms to include specifications of web service "aspects", characterizing systemic, cross-cutting concerns impacting a web service. These aspects provide an enriched description mechanism improving discovery and dynamic integration [11] [16]. Our approach contrasts to other approaches to describing cross-cutting aspects for web services [1], [5], [4], which focus on supporting interface extension and weaving or before/after processing code insertion for invoked services.

We present a motivation for our work and describe our new aspect-oriented web services (AOWS) architecture. We then present a formal specification of AOWS using Alloy [13], and describe our experiences implementing .NET-based web services with our approach. We summarize the current strengths and limitations of our AOWS approach and present some directions for future research.

2. Motivation

Consider a travel planning application built from dynamically discovered web services providing travel item search (flights, cars, hotel rooms etc), booking, payment, event scheduling and itinerary management [21]. The application developer wants to allow dynamic discovery of appropriate services providing these functions. Multiple, alternative service providers may be discovered. Services may provide limited or comprehensive functionality. Some may be free, others require payment. 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 validation of a web service may need to be performed to ensure it actually meets its advertised characteristics.

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:

- 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 such web services be appropriately described and advertised so that clients can discover and integrate with them? This includes describing extra behavioural characteristics to select between web services with compatible interfaces.
- 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.
- How can adaptors to components be discovered or synthesized and initialized, including supporting composite component aggregation?

Current approaches, such as TopCoder^R [23], OMG's Model Driven Architecture (MDA) [20], Select Perspective^{TM} [16] and Architecture Based Component Composition Approach (ABC) [19], try to combine the best of traditional software development methodologies with the power of community-based development, but have not addressed the above problems. They focus on low level features of components rather than component requirements and inter-component relationships, making the components hard to understand and combine. One of the most popular is OMG's MDA, which defines software using UML models, including base model specifying business functionality and behaviour in a technology-neutral way (Platform-Independent Model and an intermediate model (Platform-Specific Model) reflecting non-business, computing-related details, e.g. affecting performance and resource utilization, added to the Platform-Independent Model by the web services' architects. Though it is commendable and can be applied to large scale web service-based system development, it has several disadvantages, including exhaustive and sometimes frustrating editing of complex designs and implementations for large systems, and a focus on lower-level system features.

TopCoder^R, the most comprehensive and practical, has four stages to a release of each component: specification, architecture/design, development/testing, and certification. If any phase fails an acceptance test, the phase is restarted. This methodology is tedious and also focuses on lower-level features of the component/system. This can make designs hard to understand at abstract levels or during 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.

Various web services-oriented compositional methods and techniques have been developed in recent times [7], [3], [22], [23]. Most of these focus on discovering or defining component interfaces and some focus on supporting specification of specific kinds of cross-cutting concerns. However few as yet have comprehensive architectural and specification support for a wide range of aspects impacting web services. Several aspect-oriented development approaches for large component-based systems have been developed [4], [14], but most have not as yet been applied to web services.

3. Aspect-oriented Web Services

To solve these challenges and overcome current limitations we have been developing Aspect-Oriented Web Services (AOWS). This extends our AOCE work which developed extensions to the object component model to support component design, de-coupled implementation and run-time discovery and integration using component aspects [1], [8], [9], [10]. Component aspects are cross-cutting concerns impacting on components, including persistency management, distribution, security, transaction processing and resource use. Instead of merely weaving or morphing [11] aspects into code using joint points, point cuts and advice mechanisms, AOCE also uses our more efficient and effective concept of developing highly characterized and categorized reusable software components that are enriched with clearly identified and defined aspects from the beginning of the development process itself. These 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.

AOWS also uses the concept of aspects [15], in this case aspects that impact on different parts of web services. Figure 1 shows an example from the travel planner system. The client discovers various services from a registry (1). Flight searches are performed via various providers (2), and bookings made directly or via agents (3), possibly using a payment system (such as credit card authorization) (4). Two examples of web

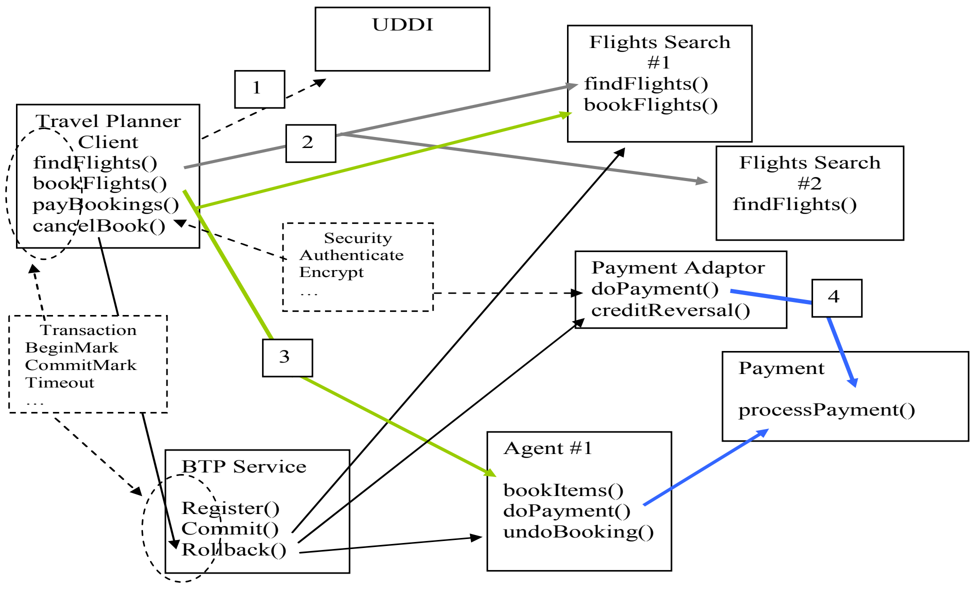


Figure 1. Examples of web service aspects.

service aspects (dotted boxes) and their impact are illustrated. Security issues 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 [21]. We aim to better support this range of activities when designing, implementing and deploying web services using AOWS. We have developed a model of AOWS-based systems, formal specification of this model, and proof-of-concept implementation of the model with .NET web services.

Figure 2 shows key architectural abstractions of our approach. A web service client, AOWebServiceRequester, uses an AOConnector (1) to communicate with service providers and services. These include an AOUDDI repository, a set of AOWebService Providers, AOComposites, Runtime Testing Agents and a set of AOAdaptors.

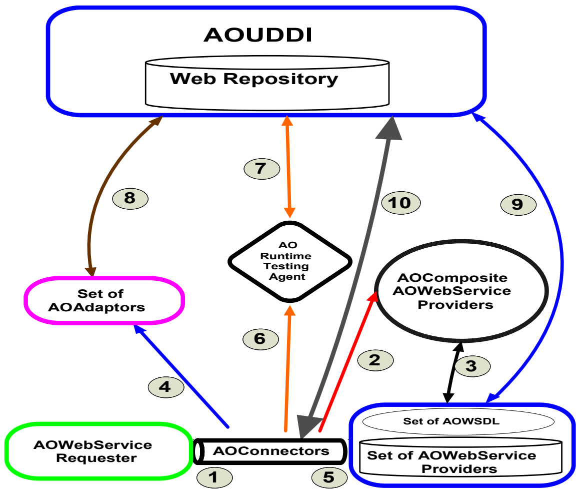


Figure 2. An overview of our Aspect-oriented Web Services architecture.

If an AOWebServiceRequester needs a set of services that are only available by composing multiple providers, an AOComposite is used (2). This selects from the set of the relevant providers and present itself as a provider of those services (3). For example, a user may wish to search for holiday resorts, inter-connecting flights, other transport arrangements and make a single payment for the whole trip. The client needs multiple providers to achieve this myriad of tasks. The relevant providers are selected and bundled by an AOComposite and made available as a composite service to the requester's AOConnector.

AOWebServiceProviders must publish their services by registering and depositing their unique AOWSDL [21] documents with the AOUDDI (9). AOWSDL interfaces include aspect specifications of service capabilities and requirements for operation, as used in AOCE. Our AOWSDL document is very different from other aspect-oriented service documents like AO4BPEL [4] which employs more traditional point-cut specification mechanisms over services.. Using our AOWSDL is richer and more structured, providing more details about both functional and non-functional properties of web service interfaces via aspect details and detail property constraints. This allows us to configure and adapt to whole components which cannot be done using AO4BPEL. We use two sets of descriptors in our AOWSDL, one more verbose for humans to read and another crisp one for robots to decipher and dynamically locate, integrate and consume useful discovered web services. These enable rich queries of the repository for services and deployment-time testing of discovered services to ensure they meet requester requirements and advertised capabilities.

AOConnectors communicate with the services (5) to service AOWebService Requester requests. For example, the travel planner client (AOWebServiceRequester) may request Flight Booking services that, in addition to standard UDDI query terms, provide BTP long transactions, and require pre-authentication of the client's user and a Travel Agent payment service. An AOConnector queries the AOUDDI for all such services and then services client requests via their AOWebServiceProviders.

AOConnectors may use adaptors to communicate with a web service (4), obtained via the AOUDDI registry. Extended UDDI queries are used to locate appropriate adaptors (8). The AO Adapter checks if the AOWSDL documents are in the proper protocol, and if not, the adaptor converts it to the proper format. AORunTimeTestingAgents can validate a discovered web service (6), from its AOWSDL description by mimicking the requester and using AOConnectors as conduits to test the providers. This ensures that the responses and results received are in conformity with the services promised in the AOWSDL.

4. Specification of AOWS using Alloy

We have formally modelled, analyzed and verified the various AOWS subsystems and their relationships in Alloy [13]. In this section we overview Alloy and show how we used it to formally specify the structure and behaviour of AOWS-based systems.

4.1. Alloy Overview

Alloy is a first-order logic based structural modelling language for expressing complex structural constraints and behaviors. It treats relations as first class objects and uses relational composition operators to combine structured entities. Essential constructs are:

- 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 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 Boolean behavior constraints in AOWS and evaluates them. It is a parameterized formula that can be further applied in other constraints.
- A fact (fact) imposes global constraints on relations and objects. A fact is a formula that takes no arguments and need not be invoked explicitly. It acts as a model axiom.
- An assertion (assert) specifies an intended property in the AOWS system. It is a formula which needs to be checked for correctness, assuming the facts in the model.

The Alloy Analyzer is a tool for analyzing models written in Alloy [6]. Given a finite scope for a specification, Alloy Analyzer translates it into a propositional formula and uses a solver to generate instances that can satisfy the facts and properties expressed in the specification. I.e., given a formula and a scope and 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 Analyzer provides two kinds of risk analysis for our AOWS specification. The first is to check whether constraints given are too weak. Flaws of this kind are found by 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 kind are found by simulation where consistency of a fact or function is demonstrated by generating a snapshot showing its invocation.

4.2. Alloy Specification of AOWS

Using the constructs in Alloy we formally modelled AOWS based on the component inter-relationships outlined earlier. Here we define some of the signatures used to construct our Alloy model. The full list and description of all the signatures used for our model is too large to include completely in this paper. Figure 3 shows the signature of the AOConnector, central to our architecture, which acts as a conduit between the various other 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 know about any updated, useful and current information about all the relevant and available AOWeb services through their respective AOWSDLs, the composite and any web service that is consumed by the client, as shown in the fields of the connector.

Figure 3 also specifies signatures for AOWS requesters and providers. Requesters connect to all other AOWS objects through a connector making requests and getting responses through it. The most important feature of a provider for any client is the AOWSDL document exposing its services. We model each AOWSDL as a set of AO Components modeling the collection of aspect-oriented components the service provides. Each component contains a unique name for its provider, a set of AOComponent(s), an AODocumentation, and AOWSDescription. AOComponents contain sets of Functional Aspects and NonFunctionalAspects. Each component also has a name, used as an identity, and an AOComponentDescription. FunctionalAspects is a collection of aspects related to specific aspectual functions, for instance persistency aspects having aspect details catering for search, update, delete and insert operations, whereas the set of NonFunctionalAspects are not specific to core functionality, e.g. performance aspects.

```
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 3. AOConnector, AOWebServiceRequester, AOWebServiceProvider, AOWSDL and related aspectual signatures used to model AOWS.

```
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 4. Facts and predicates, relating providers, requesters and aoconnectors.

```
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 5. Alloy code snippet from a formal model of the Travel Planner application.

The AODocumentation is human readable and contains summarized information about 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.

The signatures for all the elements of an AOWS system together specify all the parts of AOWS in our Alloy model. In Figure 4 are 2 of the many facts that define the structure of an aspect oriented web service provider. Together they state that an AOWSDL is unique to a particular web service provider, as no two AOWSDL can be the same as they have at least a different URL, and each AOWSDL must originate from an AOWebService Provider, so requesters can integrate and consume the services. The predicate captures the behavior of the AOCconnector as it dynamically integrates with a service provider found to be useful to the requester by making a direct connection (not via an AOComposite).

We also simulated an AO 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 5 is 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.

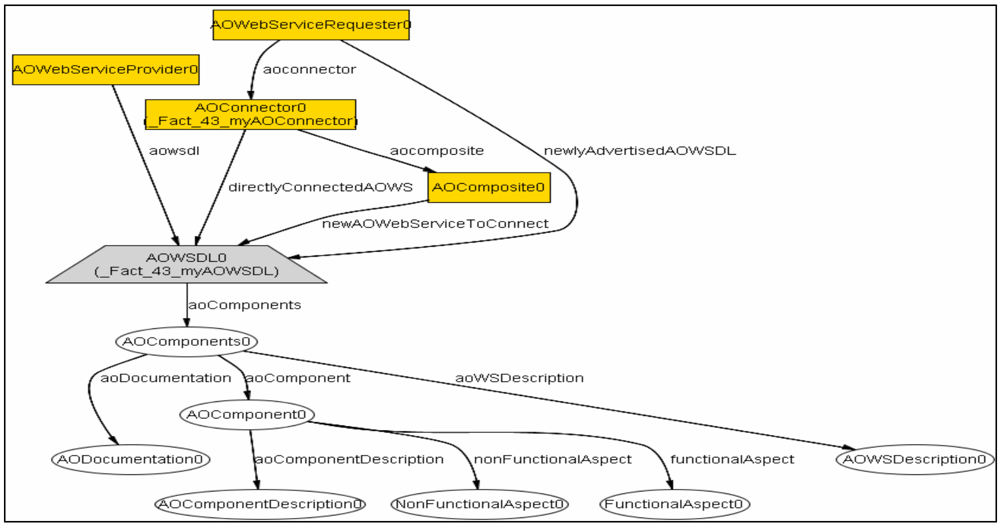


Figure 6. Model generated from Alloy showing the relationship between the aspect-oriented web service provider and requester through an AOConnector.

5. AOWS Dynamic Behaviour

Alloy Analyzer allows us to generate simulations of relationships between objects under consideration in the AOWS system. Simulations were created by first generating the main objects in the system based on their signatures. Each signature is also associated with facts, assertions and predicates permitting simulation of relationships or functions within the system described and discussed in-depth here. We simulated the behavior of various sub-systems of AOWS based on relationships and functions of its subparts. The model illustrated in Figure 6 shows the main objects involved in the dynamic interaction between a requester and provider. These are service provider, requester, AOWSDL, AOComponents, AOComponent, AOConnector, AOComposite, NonFunctionalAspects and FunctionalAspects. Correctness was tested by checking for errors or counter-examples. We then verified the model and its constraints were feasible and viable by testing assertions for scenarios relevant to AOWS behavior, for example a requester dynamically discovering and connecting to a provider through an AOConnector. When a provider with required services is found, the AOConnector is simulated to directly connect to the provider (without need for an AOComposite). The sequence of events as shown in Figure 5 is: (1) requester creates a new request (usingCreate Request()) relaying it to the AOConnector (Send RequestToAOConnector()); (2) the connector passes this request to the AOUDDI (Send RequestToAOUDDI()) which processes the request and transmit result(s) to the connector (ComputeResultAndTransmit()); (3) the AOConnector selects the best AO web service provider (SelectBestAOWS()) based on matching AOComponents, aspects, aspect details and properties required; (4) it then dynamically connects and integrates the requester with selected provider through the connector object (DirectConnectionToRequestedAOWS ()).

```
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',
          myAOConnector, myAOUDDI )
      ComputeResultAndTransmit ( myResult,
          myAOUDDI, myAOConnector )
      SelectBestAOWS ( myAOConnector,
          myAOWSDL )
      DirectConnectionToRequestedAOWS(
         myAOConnector', myAOConnector )
      }
} check
   TestDirectConnectionToRequestedAOWS for 2
```

Figure 7. Alloy assertion for dynamic service discovery via an AOConnector.

Figure 8 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. A succession of other scenario-based assertions was then applied. No counter-examples were found by the Analyzer in its check runs for any scenario we tested, giving us confidence that our AOWS approach is formally feasible and logically correct [13].

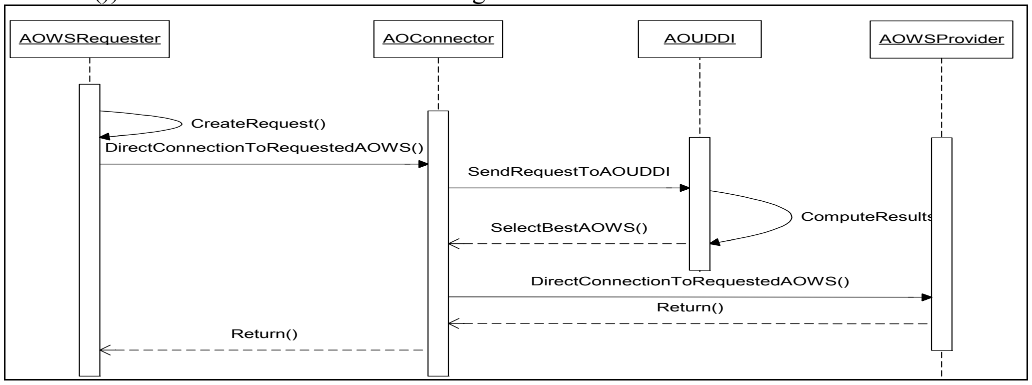


Figure 8. Sequence diagram depicting the dynamic service discovery via an AOConnector that was simulated using Alloy assertions.

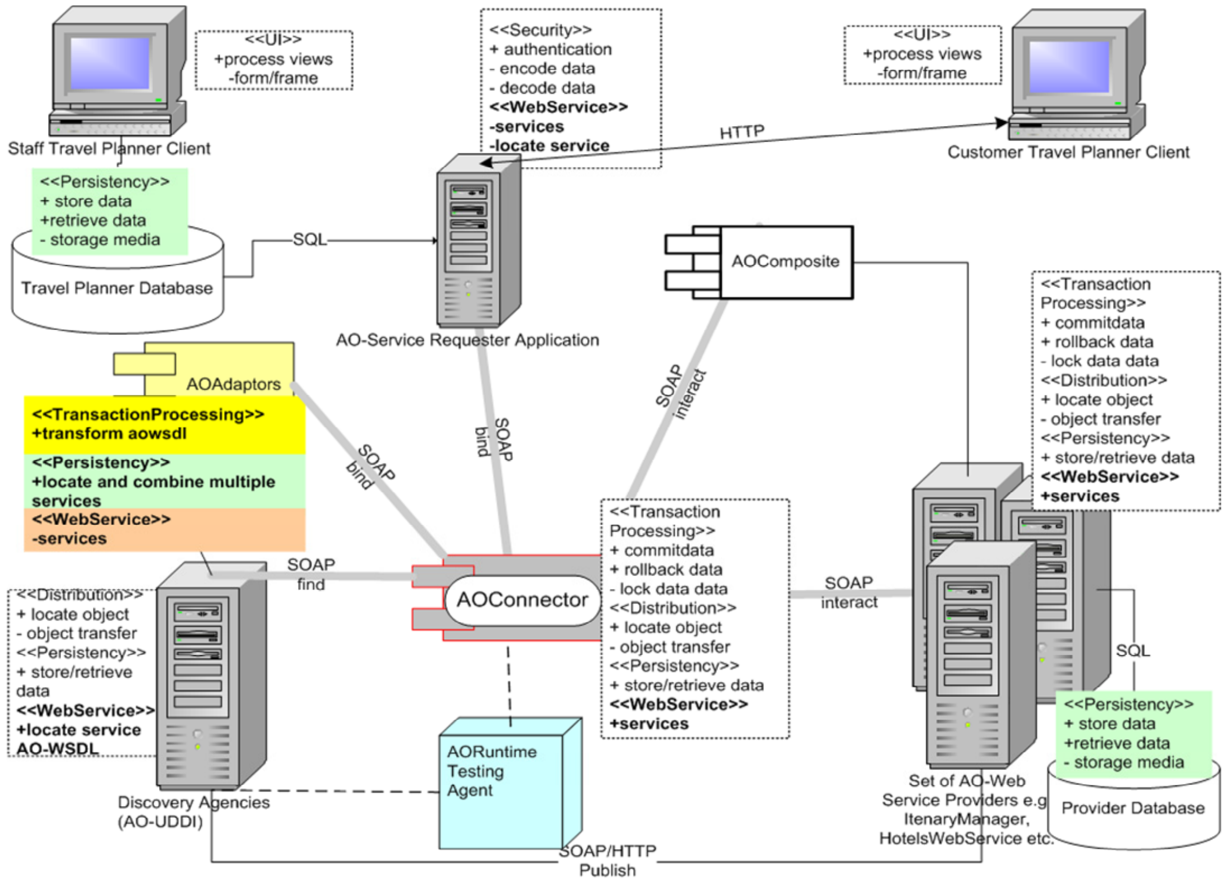


Figure 9. The travel planner's AOWS-based architecture.

6. Implementation

To demonstrate our Alloy-modelled approach is realizable, we designed and developed a prototype collaborative Travel Planner that can dynamically discover and locate relevant aspect-oriented web service providers through our AOConnector mechanism so that users can use it to plan and make bookings for various itinerary items of their travel. Figure 9 shows the architecture of the collaborative Travel Planner using abstractions from our Alloy AOWS model. The AOUDDI, AOWeb Service requesters (clients), AOWeb Service providers, AOAdaptors, AOComposite and AORuntimeTestingAgents are inter-connected through the AOConnector sub-system. The AOWS system is completely componentized and aspectized. The XML messages flowing through the connector are enriched with aspects and aspectual properties. These aspectual features together with the connector provide more efficient and effective dynamic description, discovery and integration of web services, irrespective of language/platform.

Figure 9 shows some aspects of components in the Travel Planner subsystems (in format <<Aspect name>>). 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.

We then implemented the .NET [2], [18] aspect-oriented web services system based on the Alloy model 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;

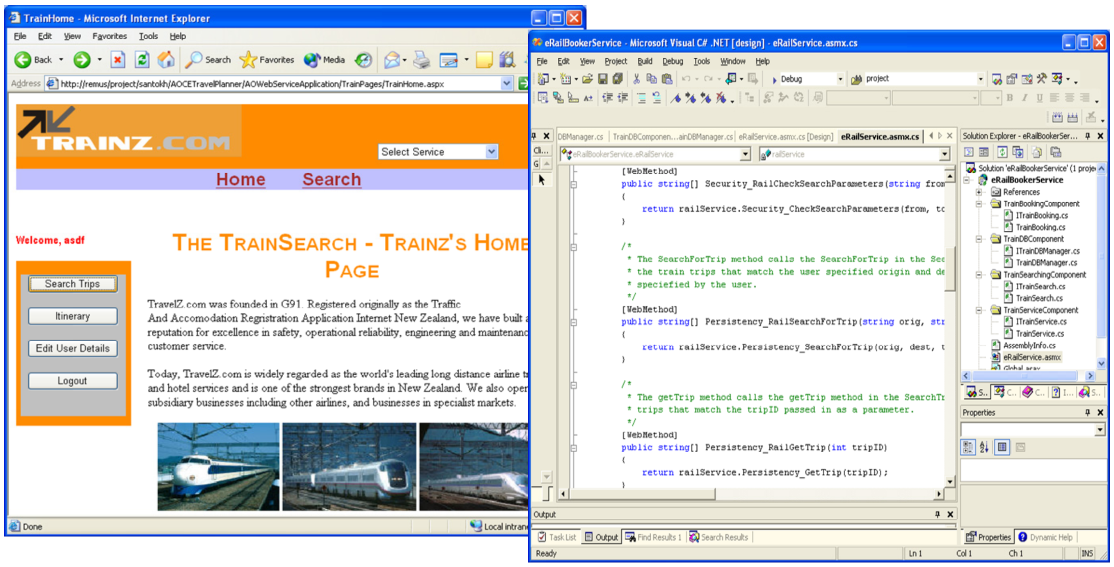
- may make use of “standardised” aspect-based adaptors to interact with discovered web services in a de-coupled manner without hard-coding type or behavioural information about the component.

Figure 10 shows a section of the GUI of the Travel Planner 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 to show their interfaces and implementing classes.

7. Discussion

The use of Alloy to formally model, analyse and verify the AOWS architecture allowed us invaluable insight into the various subsystems, AO components and objects that constitute the web-based system and their detailed aspect-oriented operations that make it work. We were able to identify possible problems in our models, isolate and rectify them in our designs before any implementations were done. This was particularly useful in the connector object as initially we were not exactly sure how many operations it should support to be optimal. Through the use of Alloy we were able to analyze, modify and mould it into its present state as described in this paper.

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, and processing information e.g. choosing the best web service provider. The connector is not modelled along existing models e.g. the façade patterns etc., because besides the above inclusion of functionalities into it, our connector object needs to possess, albeit a minimal knowledge of the particular client using it so that it can have a one to one relationship with the client but a one to many relationship with the web services. When a connection is terminated with the client, the connector be reused with other clients by resetting it and clearing its memory first. 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.


TrainHome - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://remus/project/santokh/AOCETravelPlanner/AOWebServiceApplication/TrainPages/TrainHome.aspx
TRAINZ.COM
Select Service
Home
Search
Welcome, asdf
Search Trips
Itinerary
Edit User Details
Logout
Done
Local intranet
eRailBookerService - Microsoft Visual C# .NET [design] - eRailService.asmx.cs
File Edit View Project Build Debug Tools Window Help
Debug
project
DBManager.cs
TrainDBComponen...ainDBManager.cs
eRailService.asmx.cs [Design]
eRailService.asmx.cs
eRailBookerService.eRailService
railService
Output
Task List
Output
Find Results 1
Search Results
Ready
Ln 1
Col 1
Ch 1
INS
Properties
Dynamic Help

THE TRAINSEARCH - TRAINZ'S HOME PAGE

TravelZ.com was founded in G91. Registered originally as the Traffic
And Accomodation Regristration Application Internet New Zealand, we have built a
reputation for excellence in safety, operational reliability, engineering and maintenanc
customer service.

Today, TravelZ.com is widely regarded as the world's leading long distance airline t
and hotel services and is one of the strongest brands in New Zealand. We also oper
subsidiary businesses including other airlines, and businesses in specialist markets.

```
[WebMethod]
public string[] Security_RailCheckSearchParameters(string from
{
    return railService.Security_CheckSearchParameters(from, tc
}

/*
 * The SearchForTrip method calls the SearchForTrip in the See
 * the train trips that match the user specified origin and de
 * speciefied by the user.
 */
[WebMethod]
public string[] Persistency_RailSearchForTrip(string orig, str
{
    return railService.Persistency_SearchForTrip(orig, dest, t
}

/*
 * The getTrip method calls the getTrip method in the SearchTr
 * trips that match the tripID passed in as a parameter.
 */
[WebMethod]
public string[] Persistency_RailGetTrip(int tripID)
{
    return railService.Persistency_GetTrip(tripID);
}
```

```
Solution Explorer - eRailBookerSer...
Solution 'eRailBookerService' (1 proje
  eRailBookerService
    References
    TrainBookingComponent
      ITrainBooking.cs
      TrainBooking.cs
    TrainDBComponent
      ITrainDBManager.cs
      TrainDBManager.cs
    TrainSearchingComponent
      ITrainSearch.cs
      TrainSearch.cs
    TrainServiceComponent
      ITrainService.cs
      TrainService.cs
    AssemblyInfo.cs
    eRailService.asmx
    Global.asax
```

Figure 10. (a) Travel planner GUI and (b) example C# code implementing aspects.

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, AO-UDDI 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 than other conventional approaches [22], [23]. 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 [4], [10], [22]. 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 with aspect-oriented development and components have found similar results [4], [14].

8. Summary

We have presented a novel methodology and software architecture called aspect-oriented web services which addresses identified problems with current web services approaches, notably in the areas of description, dynamic discovery and integration mechanisms. We presented a formal specification, analysis and verification of AOWS using Alloy, a formal modelling language. Aspect-oriented Component Engineering or AOCE was used to provide the new development framework for describing and reasoning about the AOWS component/systemic capabilities from multiple aspect-oriented perspectives. We further used .NET web services technology and successfully implemented a prototype of the formalized AOWS in the form of a collaborative Travel Planner application to serve as proof that AOWS is also practical and realizable.

Acknowledgements

The many helpful comments of the anonymous referees on an earlier draft of this paper are gratefully acknowledged. This work has been supported in part by the New Zealand Foundation for Research, Science and Technology and the University of Auckland Research Committee.

References

[1] Adams, C., Boeyen, S. UDDI and WSDL extensions for Web service: a security framework, In Proc. 2002 ACM workshop on XML security, Fairfax, VA , 2002.

[2] Ballinger, K., .NET Web Services: Architecture and Implementation, Addison-Wesley, 2003.

[3] Cerami, E. Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, Feb 2002, O'Reilly.

[4] Charfi A., Mezini M., Aspect-oriented Web service composition with AO4BPEL, ECOWS 2004, Springer-Verlag. 2004, Berlin, Germany.

[5] Colyer A., Clement, A., Large-scale AOSD for Middleware, AOSD 04, ACM.

[6] Dong, J.S., Sun, J., and Wang, H., Checking and Reasoning about Semantic Web through Alloy, FME 2003, LNCS 2805.

[7] Gannod, C., Bhatia, S. Facilitating Automated Search for Web Services, In Proc. IEEE International Conference on Web Services, ICWS'04, IEEE.

[8] Grundy, J.C. and Hosking, J.G., In Engineering plug-in software components to support collaborative work, S-P&E, 2002; vol. 32, pp. 983-1013.

[9] Grundy, J. Multi Perspective Specification, Design and Implementation of Software Components using Aspects, In Int. J. Soft. Eng. and Knowledge Eng. Vol. 10, No. 6 (2000), pp. 713-734, World Scientific.

[10] Grundy, J. and Ding, G. Automatic Validation of Deployed J2EE Components Using Aspects, In Proc. 2002 IEEE International Conference on Automated Software Engineering, Edinburgh, UK, IEEE CS Press.

[11] Hanenberg, S., Hirschfeld, R., Unland, R.. Morphing Aspects: Incompletely Woven Aspects and Continuous Weaving, AOSD 04, Lancaster, UK, ACM 2004

[12] Heisel, M., Santen, T., and Souqui`eres, J., Towards a Formal Model of Software Components, In Formal Methods and Software Engineering - Proc. 4th Int. Conf. on Formal Engineering Methods.

[13] Jackson, D. Alloy: a lightweight object modeling notation. ACM Trans. on Software Engineering and Methodology, 2002.

[14] Katara, M., Katz, S., Architectural Views of Aspects, In Proc. AOSD 2003, Boston, MA USA, ACM 2003.

[15] Kiczales et al, Aspect-oriented Programming, In Proc. the 1997 European Conf. on Object-Oriented Programming, Finland (June 1997), Springer-Verlag, LNCS 124.

[16] Latchem S. Patterns for Internet development. Providing a component based technical architecture for Internet solutions. SIGS 2000, Newdigate, UK.

[17] Lieberherr, K. Connections between Demeter/Adaptive Programming and Aspect-Oriented Programming (AOP), http://www.ccs.neu.edu/home/lieber/, 1999.

[18] Microsoft, Visual Studio and .NET, http://www.microsoft.com/net/, 2003, Microsoft..

[19] 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.

[20] Siegel, J. Using OMG's Model Driven Architecture (MDA) to Integrate Web Services, http://www.omg.org/.

[21] Singh, S., Grundy, J., Hosking, J.,. Developing .NET Web Service-based Applications with Aspect-Oriented Component Engineering, AWSA'04, Australia.

[22] Stearns, M., Piccinelli, G., Managing Interaction Concerns in Web-Service Systems, Proc. 22nd Int. Conf. on Distributed Computing Systems Workshops, pp. 424

[23] TopCoder^{R}, 2005, http://www.topcoder.com/.

[24] Vitharana, P., Mariam, F., and Jain, H., Design, Retrieval, And Assembly in Component-based Software Development, CACM, vol. 46, no. 11, Nov. 2003.