# Automated responsive web service evolution through generative aspect-oriented component adaptation

Xiaodong Liu*, Yankui Feng and Jon Kerridge

School of Computing,
Napier University,
Edinburgh EH10 5DT, UK
E-mail: x.liu@napier.ac.uk E-mail: y.feng@napier.ac.uk
E-mail: j.kerridge@napier.ac.uk
*Corresponding author

**Abstract:** When building service oriented systems, it is often the case that existing web services do not perfectly match user requirements in target systems. To achieve smooth integration and high reusability of web services, mechanisms to support automated evolution of web services are highly in demand. This paper advocates achieving the above evolution by applying a highly automated aspect-oriented adaptation approach to the underlying components of web services by generating and then applying the adaptation aspects under designed weaving process according to specific adaptation requirements. An expandable library of reusable adaptation aspects at multiple abstraction levels has been developed. A prototype tool is developed to scale up the approach.

**Keywords:** web service evolution; web service integration; aspect-oriented programming; aspect reuse and generative component adaptation.

**Reference** to this paper should be made as follows: Liu, X., Feng, Y. and Kerridge, J. (2008) ‘Automated responsive web service evolution through generative aspect-oriented component adaptation’, *Int. J. Computer Applications in Technology*, Vol. 31, Nos. 1/2, pp.83–93.

**Biographical notes:** X. Liu received his PhD in Computer Science from De Montfort University, UK. He is an academic member of the School of Computing, Napier University. As an active researcher, his current research focuses on software evolution, software reuse, component-based development and formal methods.

Y. Feng is a PhD researcher in software reuse. He received his MSc in Software Engineering from the University of Edinburgh. He had seven years experience as a System Developer of information systems, web-based applications and control systems prior to joining Napier.

J. Kerridge received his PhD in Computing from the University of Manchester, UK. He is an active researcher in database systems, modelling, and software engineering. He has been the Leader of more than 20 research projects.

## 1 Introduction

Currently, web services provide a model for design and implementation of distributed applications by loosely and dynamically linking a number of smaller web services that can inter-operate regardless of how they are implemented and where they are hosted. Based on web services, Service Oriented Architecture (SOA) (Ferris and Farrell, 2003; Kleijnen and Raju, 2003; Kreger, 2003) is the latest evolution in distributed computing.

However, the increasing popularity of web services and SOA imposes new challenge on the evolution of web services due to their unique characteristics (Arsanjani, 2002; Baldwin et al., 2002; Bennett and Xu, 2003; Litoiu, 2004; Kajko-Mattsson and Tepczynski, 2005). The users of a web service may distribute globally and are very diverse in their detailed requirements. Web services are normally linked into applications loosely and the linking is very changeable. In reality, it is often the case that an existing web service does not perfectly match user requirements in a target system although it is generally qualified. All the above factors incur frequent and imminent evolution of web services. Multiple versions of a web services need co-exist. To assure seamless composition and wide reusability of web services, mechanisms to support rapid automated evolution of web services in either functional or non-functional aspects are highly in demand.

This paper presents an approach to achieve web service evolution at the above level. The approach, namely a Generative Aspect-oriented component adaptatIoN (GAIN), is based on the successful points in a few technologies, i.e., Aspect Oriented Programming (Mezini and Ostermann, 2005; Kiczales et al., 2001; Sullivan, 2001; Viega and Voas, 2000), Software Product Line (Batory et al., 2002; Diaz-Herrera et al., 2000) and Generative Component

Adaptation (Batory et al., 2000; Cleaveland, 1998). In GAIN approach, web service evolution is carried out by adapting the underlying components of web services within an aspect-oriented component adaptation framework by generating and then applying the adaptation aspects under designed weaving process according to specific adaptation requirements. The generation absorbs the variation concept of software product line and assures the perfect suitability of adaptation aspects for the specific adaptation requirements of aimed reuse context. Compared with traditional AOP, the weaving process of aspects in GAIN supports more complicated control flow, i.e., not only sequence, but also switches, synchronisation and multiple threads, to make the adaptation more accurate and efficient for components reused in more complicated environments such as concurrent dynamic applications. To facilitate the reusability of adaptation knowledge, an expandable library of reusable adaptation aspects at multiple abstraction levels has been developed. A prototype tool is developed to scale up the approach.

The remainder of the paper is organised as follows: Section 2 discusses the related work. Section 3 describes the approach framework. Section 4 presents how to generate and apply reusable adaptation aspects under the designed weaving process. Section 5 introduces the prototype tool, and Section 6 presents an example to demonstrate the approach. Finally, Section 7 presents the conclusion.

## 2 Related work

### *2.1 Achieving extensibility through product-lines and Domain-Specific Languages (DSL)*

In Batory's et al. (2002) approach, system extensibility and understandability can be achieved through an integration of Product-Line Architectures (PLA) and DSL technologies. GenVoca PLA is developed to express the building blocks as layers or refinements, whose addition or removal simultaneously impacts the source code of multiple, distributed programs. They extended the Java language with a domain-specific language, to express state machines and their refinements, and wrote their components in this extended language.

The above work represents a working approach to software system evolution with generative programming and product line technology; however, the approach is very domain specific, which may limit its applicability.

### *2.2 Feature analysis for service-oriented reengineering*

Chen et al. (2005) introduced an approach to supporting service-oriented reengineering from non-service-oriented software systems by using feature analysis. With feature-based information, service identification and packing processes are performed and result into a service delegation, which integrates reusable software components into service construction.

Instead of direct evolution of web services, the work focused on reengineering legacy software into service-oriented systems.

### *2.3 Semantically extensible schemas for web service evolution*

Wilde (2004) presented a framework for semantically extensible schemas for Web service evolution. The basic idea is to construct a framework which augments the almost non-existent support for versioning of web services in the Simple Object Access Protocol (SOAP) and the Web Service Definition Language (WSDL). Versioning not only covers controlled ways to deal with different version of a service's vocabulary, but also means to semantically describe extensions, so that older software versions can 'understand' newer versions of the service vocabulary.

### *2.4 Superimposition*

Superimposition (Bosch, 1999) is a novel black-box adaptation technique proposed by Bosch at University of Karlskrona/Ronneby. Software developers are able to impose a number of predefined, but configurable types of functionality on reusable components. The notion of superimposition has been implemented in the Layered Object Model (LayOM), an extensible component object language model. The advantage of layers over traditional wrappers is that layers are transparent and provide reuse and customisability of adaptation behaviour.

Superimposition uses nested component adaptation types to compose multiple adaptation behaviours for a single component. However, due to lack of component information, modification is limited at simple level, such as conversion of parameters, and refinement of operations. Moreover, with more layers of code imposed on original code, the overhead of the adapted component increases heavily, which degrades system efficiency.

### *2.5 SAGA project*

Scenario-based dynamic component Adaptation and GenerAtion (SAGA) (Liu et al., 2005; Wang et al., 2004) at Napier University developed a deep level component adaptation approach with little code overhead through XML-based component specification, interrelated adaptation scenarios and corresponding component adaptation and generation.

SAGA project focused on mainly generative component adaptation at binary code level, i.e., the adapted part of the component will be generated as new blocks of binary code and these blocks will then be composed with other unchanged blocks of code to form a new adapted component. However, automation is a challenge in SAGA approach because it is always complex to generate blocks of code according to scenarios and original component code.

To reach high automation, a large set of adaptation rules and domain knowledge has to be developed to support the process, and probably the application domains has to be restricted as well.

### *2.6 Aspectual component*

To achieve reusable aspects, Lieberherr et al. (1999) introduced the concept of Aspectual Components. Aspects are specified independently as a set of abstract join points. Using this model, an aspect is described as a set of abstract join points which are used when an aspect is combined with the base-modules of a software system. In this way, the aspect-behaviour is kept separate from the core components, even at run-time.

It distinguishes between components that enhance and cross-cut other components and components that only provide new behaviour. An aspectual component has a provided and a required interface. Connectors connect the provided and required interfaces of other components. The connection process starts with a level-zero components consisting of very simple class definition.

### *2.7 JAsCo*

JAsCo (Suvee et al., 2003; Vanderperren et al., 2005) is an aspect based research project for component based development, in particular, the Java Beans component model. JAsCo combines the expressive power of AspectJ with the aspect independency idea of Aspectual Component. The JAsCo language introduces two concepts: aspect beans and connectors. An aspect bean is used to define aspects independently from a specific context, which interferes with the execution of a component by using a special kind of inner class, called a hook. Aspect beans can be reused and applied upon a variety of components. A connector allows specifying precedence and combination strategies between the aspects and components.

However, JAsCo is not suitable for specific modification requirements since it does not provide a mechanism for conducting users' requirements. In addition, the way to apply aspects on target components/systems is based on traditional AOP process, and therefore, may result in lower readability, maintainability and performance. Moreover, the current implementation of JAsCo has been bounded to Java, which limits its usability.

### *2.8 Summary*

Although web service evolution is highly in demand, this issue has not been addressed enough by research communities, which are mainly focusing on direct web service composition. Probably this situation is due to the short history of service-oriented technology. One solution is to apply adaptation to the underlying components of web services with appropriate component adaptation techniques, for example, generative aspect-oriented component adaptation.

Some AOP based frameworks have been developed to achieve reusable aspects. However, an AOP platform independent framework is still desired in a heterogeneous distributed environment to solve crosscutting problem since a programming language independent AOP framework is still missing. Furthermore, current AOP techniques only support weaving aspects sequentially. To cope with complex adaptation, it often requires weaving aspects in more sophisticated control flow, e.g., dynamically deciding whether to invoke a particular aspect, and synchronising in multi-thread applications.

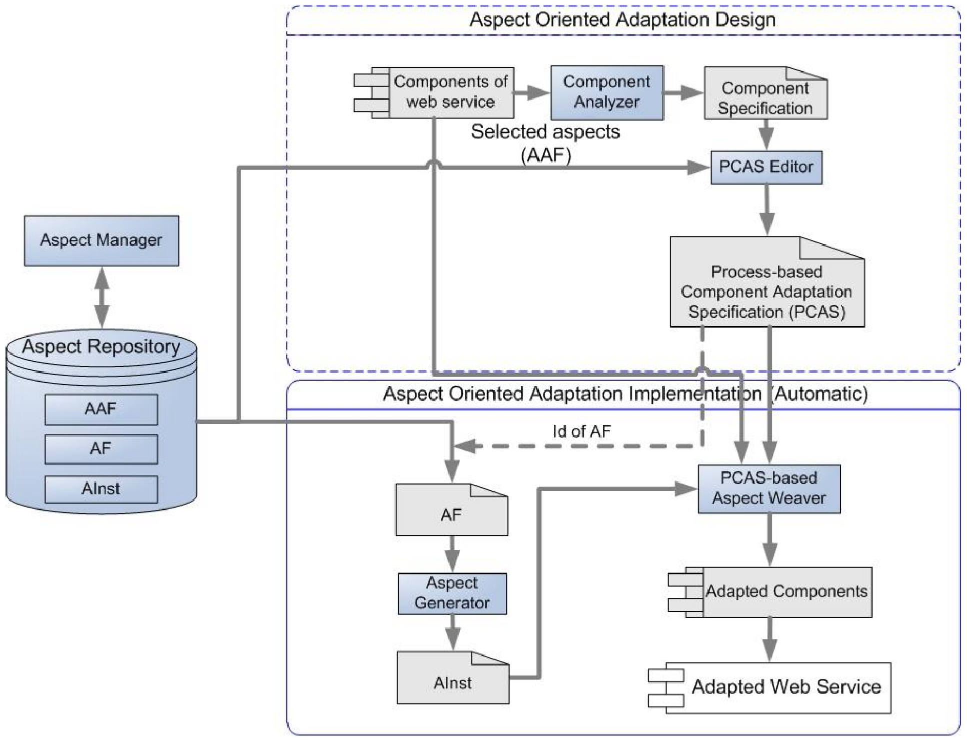
## 3 The approach framework

The general process of the proposed approach is given in Figure 1. We presume that a web service has been found potential suitable to be used in a service oriented application, however, the consumer indicated some mismatches between the web service and the requirements of the target application, and wishes to have a new adapted version of the web service.

The mismatches will be eliminated by applying aspect-oriented adaptation to the underlying components that implement the web service because the woven aspects will change the interface and behaviour of the components and hence change the behaviours and attributes of web services. At start, the components relevant to the possible adaptation are analysed with the component analyser, which analyses the source or binary code of the component and extracts component specification information, e.g., class names and method signatures. The component specification will be used to guide component adaptation. If the components already have well defined specification, this step can be skipped.

Then based on the adaptation requirements, a Process-based Component Adaptation Specification (PCAS) will be composed by software engineers, who select aspects defined at the abstraction level of Abstract Aspect Frames (AAF). The selection of aspects is actually the process to determine functional variation of a specific adaptation. An AAF is considered as a template to coin out specific aspects. The composition of PCAS is supported by an interactive IDE called PCAS Editor, which supports both graphical and XML source view of the PCAS.

**Figure 1** The Generative Aspect-oriented component adaptatIoN (GAIN)



A PCAS is an XML formatted document, which includes the details of component adaptation, such as the target component, the weaving process, and the abstract aspects to be applied. In a PCAS, sequence and switch structure are supported to achieve flexible adaptation on components. In PCAS, the adaptation process is depicted with only the ID of the selected aspects. Full details of the aspects are still kept in Aspect Repository.

Based on PCAS and the lower level aspect definition, namely Aspect Frame (AF) in the aspect repository, executable Aspects Instances (AInsts) are generated by the aspect generator according to different AOP implementation specifications. As result, platform variation is achieved during aspect generation. The input for the aspect generator is AF, which is determined at adaptation design stage, and the output is AInsts.

The aspect repository supports highly and incrementally reusable aspects. Reusable aspects are defined at different abstraction levels and kept in the repository as AAF, AF, and AInst. The reusable assets in the repository include both primitive and composite aspect types. A composite aspect type is a composition of primitive or composite aspects under an adaptation process defined in PCAS.

The aspect manager is a tool to manage reusable aspects in the aspect repository, and to present graphical views of aspects at various abstraction levels.

The generated executable aspects are applied to the component by the aspect weaver. A new adapted version of the component is then created through the aspect weaving. Since current AOP platform like AspectJ does not support complicated flow control such as switch in weaving process, post-processing is applied to enable process-based weaving in our framework.

Finally, by re-deploying the adapted components a new adapted version of the web service is created with the mismatches eliminated.

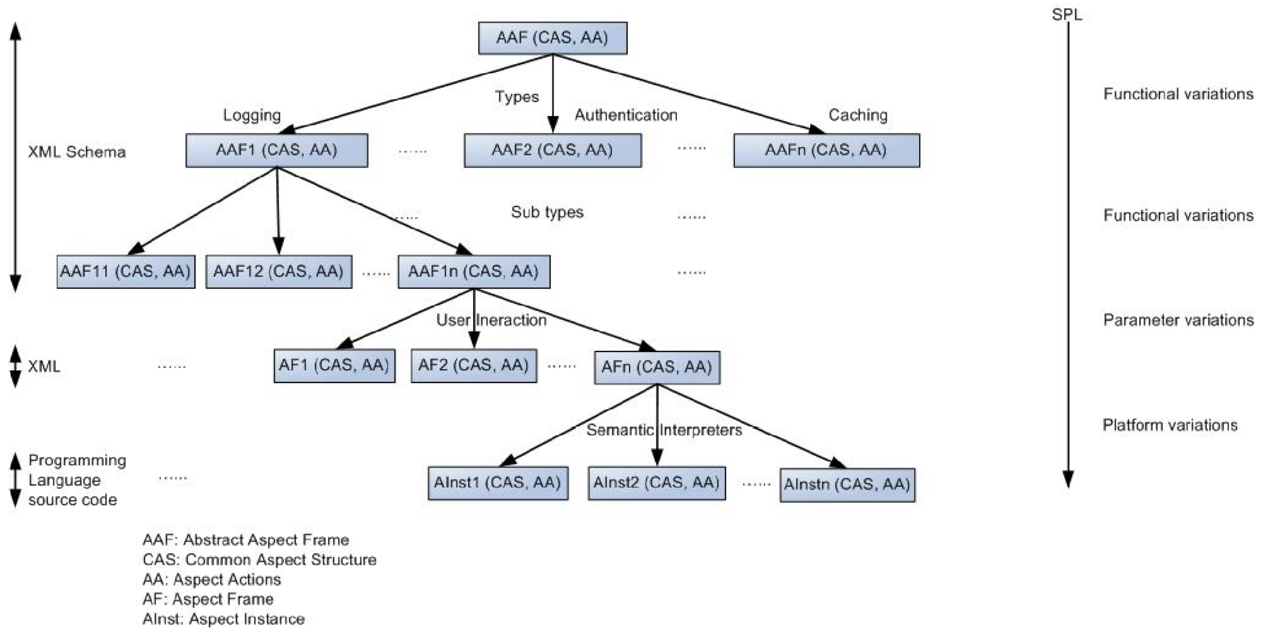
## 4 Aspect-oriented generative adaptation

### 4.1 *Capturing adaptation knowledge in aspects*

A challenge with software product line approach is to model the variability between the core assets and the applications (Batory et al., 2002, Diana and Webber, 2004; Diaz-Herrera et al., 2000). Parameterisation is a variability mechanism that allows a software engineer to change the values of the attributes in a core asset component. Modelling variability with parameterisation allows a software engineer to populate attributes. Modelling variability with variation points is where the core asset components consist of variation points and the software engineer may build target system components using unique variants built from the variation points. This approach provides software engineers with the most flexibility as it allows them to create and maintain their unique variants.

As shown in Figure 2, parameterisation and variation points have been applied in our approach to achieve highly reusable aspects. The adaptation knowledge is captured in aspects and aims to be reusable in various adaptation situations. As shown in Figure 2, to achieve automated and precise adaptation, these aspects are defined at three abstraction levels, i.e., AAF, AF and AInsts.

**Figure 2** Multiple abstraction levels of reusable aspects in Software Product Line view



AAF are the fundamental and the most abstract level of the Aspect Repository. As XML schema files, AAFs are used to define the structure of different aspects. According to the functionality, the AAFs form a hierarchical structure that reflects functional variations of different adaptations. Adaptation aspects are modelled into different types, for example, logging, caching, authentications, etc. Each aspect type is then refined into a group of sub-types. For example, aspects about authentication may consist of operating-system-based authentication and database-based authentication.

AAFs are a hierarchical aspect type system defined in XML schema format. This type hierarchy includes many levels of aspect types and sub-types, which capture various functionalities of the adaptation aspects.

Each AAF may have many AFs. AFs are the second abstraction layer in aspect definition. AFs are the instances of related AAFs. Compared with its AAF, an AF has the details of a concrete aspect populated into it by assigning a value to the parameters. User interaction is required to create an AF from an AAF. Defined in XML format, AFs are independent from concrete AOP platforms such as AspectJ.

An AF is not executable until it is mapped onto a concrete AOP platform. The result of this mapping is a family of Aspect Instances based on various AOP platforms. An Aspect Instance is executable and specific to a concrete AOP platform, and it reflects platform variations of an aspect on different AOP platforms. The agent to generate Aspect Instances from their AF is called Semantic Interpreter. The generation process is fully automatic.

The three abstraction levels of aspects facilitate the reusability of adaptation aspects as they realise different variations of these aspects, including functional variations, parameter variations and platform variations. At each level, a pair, namely (CAS, AA) is used to describe Common Aspect Structure (CAS) and Aspect Actions (AA). Common core assets are defined in Common Aspect Structures and variations are defined in Aspect Actions.

CAS provides the basic information of an aspect, e.g., which component to be adapted (target component), pointcut name, etc. All aspects have the same CAS at AAF level no matter how different these aspects are in functionality and implementation platform.

On the other hand, Aspect Actions provides the information of the variations of different aspects of the same or different aspect types. For instance, for an aspect of logging type, an output file name must be provided; similarly a database connection pool aspect must be supplied with a capacity.

## *4.2 Process based Component Adaptation Specification (PCAS)*

To satisfy the adaptation requirements for a particular reuse context, it often requires performing complex adaptation to multiple components with a set of generated aspects applied to these components under a specially designed process containing conditions, synchronisation and other flow controls. PCAS is developed to describe the above complicated adaptation details.

The elements in a PCAS include target component(s) (‘class’ and ‘method’), information of aspect(s) to be applied such as aspect id, type, and level (‘aspect_level’), and process control information, such as execution mode (‘Sequence’, ‘Switch’ and ‘Case’), conditions, and synchronisation support (‘synchronised’). Flow control elements are used to provide advanced weaving process, and synchronisation support enables multiple accesses to the same resource such as a file or a database from different aspects.

A sample of PCAS structure is given in Figure 3 with the data detail omitted, and a full example of the definition is given in Section 6.

If a PCAS is found common and reusable in the future, its process control part can be regarded as a composite aspect type. Composite aspects are supported in AAF level to achieve advanced reuse in typical aspect using cases.

**Figure 3** A sample of Process-based Component Adaptation Specification structure

```
<AOP-Process name='xxxx'
xmlns="http://www.dcs.napier.ac.uk/2005/PCAS">
  <Container name='xxxx'>
    <Sequence>
        <Apply-aspect  aspect_id='xxxx'
          class='xxxx'
          method='xxxx'
          aspect_level='xxxx'
          aspect_type='xxxx'
          af_id='xxxx'
          af_name='xxxx'
          synchronised='xxxx'
          comment='xxxx'/>
        <Switch expr='xxxx'>
          <case value='xxxx'>
            <Apply-aspect  aspect_id='xxxx'
             class='xxxx'
             method='xxxx'
             aspect_level='xxxx'
             aspect_type='xxxx'
             af_id='xxxx'
             af_name='xxxx'
             synchronised='xxxx'
             comment='xxxx'/>
          </case>
          <case value='xxxx'>
            <Apply-aspect  aspect_id='xxxx'
             class='xxxx'
             method='xxxx'
             aspect_level='xxxx'
             aspect_type='xxxx'
             af_id='xxxx'
             af_name='xxxx'
             synchronised='xxxx'
             comment='xxxx'/>
          </case>
        </Switch>
    </Sequence>
  </Container>
 </AOP-Process>
```

To implement PCAS in weaving process, a post-weaving technique is developed. The post-weaving tool gets class files for aspects generated by AOP platform such as AspectJ as input, and then modifies those class files to generate new class files that support complicated flow control and synchronisation according to PCAS.

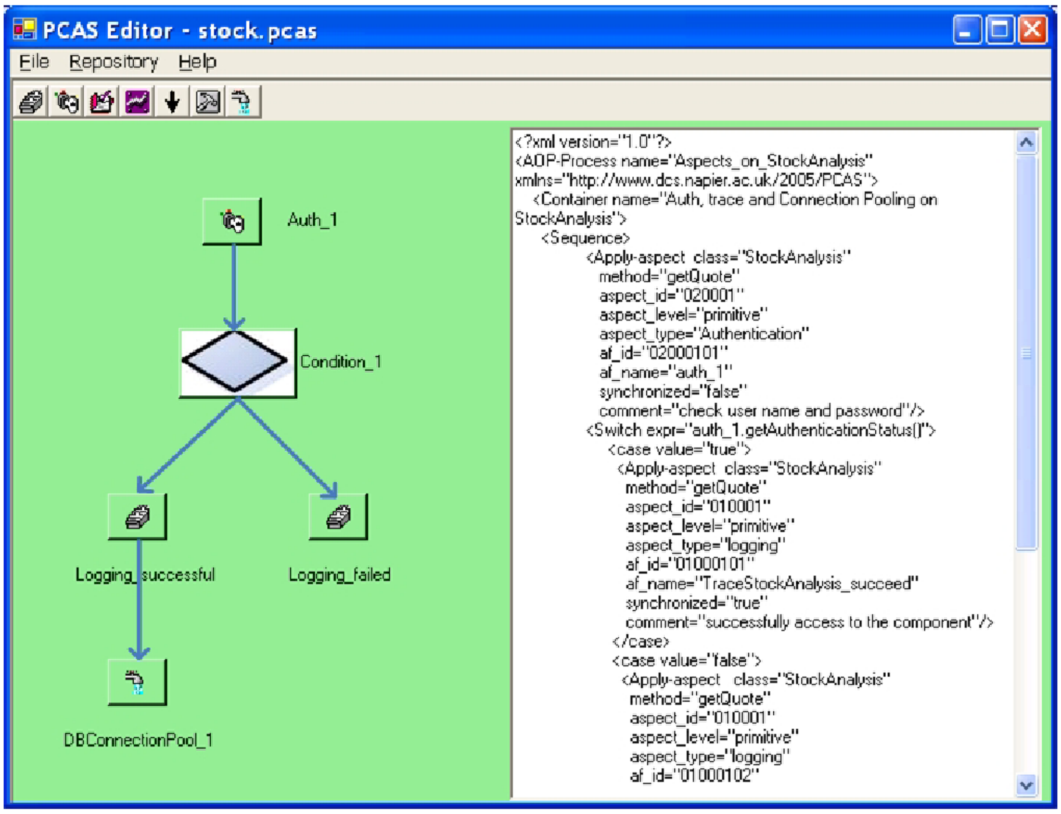
## 5 The prototype tool

A CASE tool has been developed to scale up the proposed approach. With this tool, service developers define aspect weaving process by drag-and-drop in a graphical interface; they select candidate aspects and fill in necessary details of CAS and AA. The semantic interpreter will generate AInsts automatically. According to the defined PCAS, Aspect Weaver will complete the aspect weaving and generate adapted components.

The tool includes the following parts:

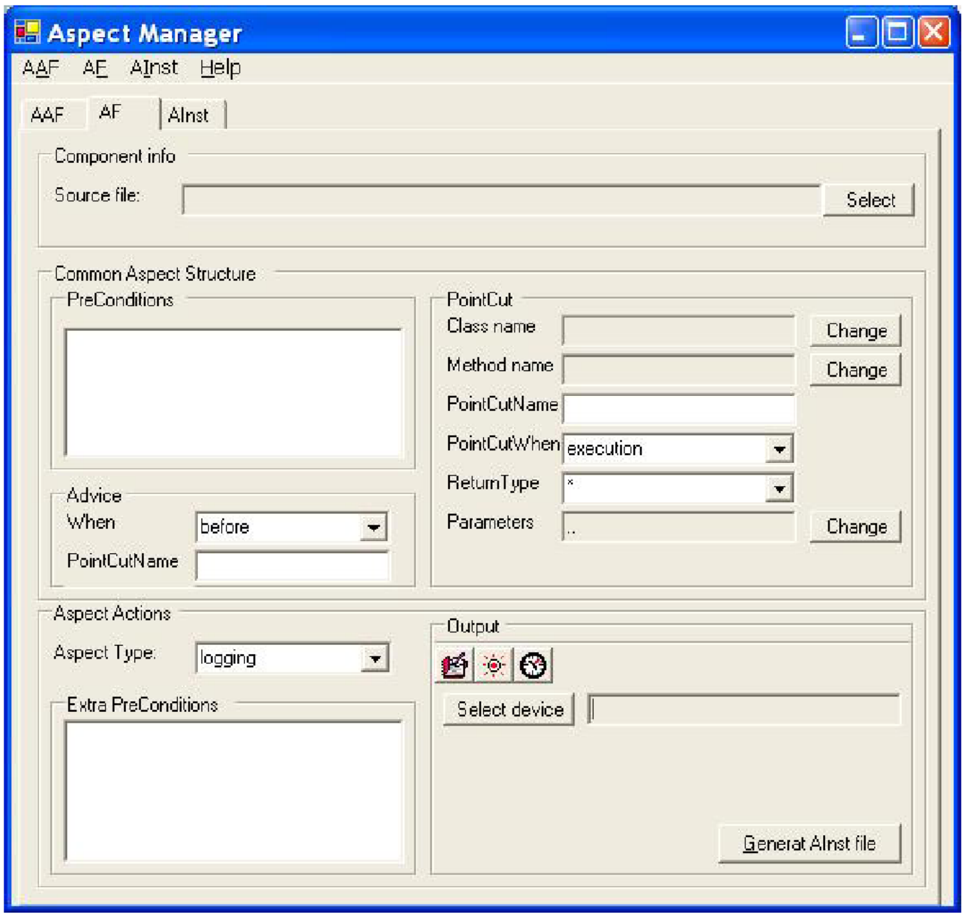
- *PCAS editor*, which provides an edit environment for PCAS both in graphical interface and at XML level. A screen dump is shown in Figure 4.
- *Aspect manager*, which supports the management and retrieval of reusable aspects in Aspect Repository and the graphical view of different levels of aspects. Aspects at different levels can be created, removed, and edited in Aspect Manager, either in the graphical user interface, or at XML level. Aspect Manager provides two retrieval modes of AAFs/AFs, that is, search by keywords and search by classification category. A screen dump of Aspect Manager is shown in Figure 5.
- *Semantic interpreters*, which translate AFs to AInsts based on selected specific AOP platform and aspects. If there are m different AOP platforms and $n$ different aspects in the tool, there will be $m \times n$ different interpreters.
- *Component analyser*, which analyses component and gets necessary information such as the class names and method names, for component adaptation.
- *Aspect generator*, based on AFs and corresponding Semantic Interpreters, executable aspect instances will be generated by Aspect Generator. The result executable aspects will be saved into aspect repository as AInsts.
- *Aspect weaver*, which is used to generate new components by weaving generated AInsts into original components.

**Figure 4** A screen dump of PCAS editor (for colours see online version)



```
<?xml version="1.0"?>
<AOP-Process name="Aspects_on_StockAnalysis"
xmlns="http://www.dcs.napier.ac.uk/2005/PCAS">
   <Container name="Auth, trace and Connection Pooling on
StockAnalysis">
      <Sequence>
            <Apply-aspect  class="StockAnalysis"
              method="getQuote"
              aspect_id="020001"
              aspect_level="primitive"
              aspect_type="Authentication"
              af_id="02000101"
              af_name="auth_1"
              synchronized="false"
              comment="check user name and password"/>
            <Switch expr="auth_1.getAuthenticationStatus()">
               <case value="true">
                <Apply-aspect  class="StockAnalysis"
                  method="getQuote"
                  aspect_id="010001"
                  aspect_level="primitive"
                  aspect_type="logging"
                  af_id="01000101"
                  af_name="TraceStockAnalysis_succeed"
                  synchronized="true"
                  comment="successfully access to the component"/>
               </case>
               <case value="false">
                <Apply-aspect   class="StockAnalysis"
                  method="getQuote"
                  aspect_id="010001"
                  aspect_level="primitive"
                  aspect_type="logging"
                  af_id="01000102"
```

**Figure 5** A screen dump of aspect manager



Source file:

Class name

Method name

PointCutName

PointCutWhen: execution

ReturnType: *

Parameters: ..

When: before

PointCutName

Aspect Type: logging

## 6 An example

Two case studies have been done to evaluate the proposed approach. In this section, based on one case study – a stock trading system, an intuitive example is given to demonstrate the proposed approach in web service evolution.

Prior to doing the case study, a share analysis component has been published as a web service to provide share information such as company name, share quote and analysis charts.

In the case study, we have found the web service is potentially suitable for our new share trading system and wish to integrate the web service into our new share trading system, which will be heavily accessed by many users simultaneously. It is identified that this web service may be less dependable because the heavy access load often causes access failure due to the connection time-out problem. In addition, to increase the system security, access to share information also needs to be restricted to approved users and monitored by logging the access time.

To meet the above evolution requirements, the service developer plans to introduce the following aspect-oriented adaptation actions to the underlying component: firstly, to add authentication to the component prior to using it; secondly, to set up a database connection pool to relieve the access load; finally, to monitor the access detail with logging. According to the result of monitoring, the capacity of and the life span of individual connections in the connection pool are subject to constant adjustment to reach the best correction to the access failure problem.

Four aspects are applied to the component to implement the above adaptation actions, namely authentication, logging if authenticated successfully, logging if authenticated unsuccessfully, and database connection pool. These adaptation actions are then described in a PCAS shown in Figure 6. As shown in Figure 4, the specification is created with the PCAS Editor by finding appropriate AAFs, i.e., either primitive types or composite types of aspects, and putting these AAFs into an adaptation process. Functional variation of adaptation is implemented through the selection of appropriate AAFs and the composition of PCAS.

The specification in PCAS is at a rather overview level and does not contain the details of individual aspects. Developers need to provide parameter value for each aspect. Common AFs can be saved into Aspect Repository for further reuse. In this example, four AFs will be generated for each of the above aspects accordingly. Due to the structural similarity of AFs of different aspects, we only give the AF for database connection pool in Figure 7 as an example.

**Figure 6** The Process-based Component Adaptation Specification

```
<?xml version='1.0'?>
<AOP-Process name='Aspects_on_StockAnalysis'
xmlns="http://www.dcs.napier.ac.uk/2005/PCAS">
<Container name="Auth, trace and Connection Pooling
                    on StockAnalysis">
<Sequence>
 <Apply-aspect  class='StockAnalysis'
   method='getQuote'
   aspect_id='020001'
   aspect_level='primitive'
   aspect_type='Authentication'
   af_id='02000101'
   af_name='auth_1'
   synchronised='false'
   comment="check user name and password"/>
 <Switch expr="auth_1.getAuthenticationStatus()">
   <case value='true'>
    <Apply-aspect  class='StockAnalysis'
      method='getQuote'
      aspect_id='010001'
      aspect_level='primitive'
      aspect_type='logging'
      af_id='01000101'
      af_name='TraceStockAnalysis_succeed"
      synchronised='true'
      comment="successfully access to the component"/>
  </case>
  <case value='false'>
   <Apply-aspect  class='StockAnalysis'
      method='getQuote'
      aspect_id='010001'
      aspect_level='primitive'
      aspect_type='logging'
      af_id='01000102'
      af_name='TraceStockAnalysis_failed"
      synchronised='true'
      comment="failed access to the component"/>
  </case>
 </Switch>
 <Apply-aspect
     class="java.sql.DriverManager,java.sql.Connection"
     method='getConnection,close'
     aspect_id='030001'
     aspect_level='primitive'
     aspect_type='DBConnectionPool'
     af_id='03000101'
     af_name="StockDBConnectionPoolAspect"
     synchronised='false'
     comment="Add all DB connections into the pool"/>
  </Sequence>
 </Container>
 </AOP-Process>
```

**Figure 7** Aspect frame of DB connection pool

```
<?xml version='1.0' ?>
<Aspect name="StockDBConnectionPoolAspect">
<!-- Core asset -->
<CommonCoreAsset>
<PreConditions>
<Precondition>java.sql.*;</Precondition>
</PreConditions>
<PointCuts>
<PointCut>
<Name>connectionOpen</Name>
<When>execution</When>
<ReturnType>java.sql.Connection</ReturnType>
<ClassName>java.sql.DriverManager
</ClassName>
<MethodName>getConnection</MethodName>
<Parameters>String,String,String</Parameters>
<Advice>                                              CAS
<When>around</When>
</Advice>
</PointCut>
<PointCut>
<Name>connectionClose</Name>
<When>execution</When>
<ReturnType></ReturnType>
<ClassName>java.sql.Connection</ClassName>
<MethodName>close</MethodName>
<Parameters></Parameters>
<Advice>
<When>around</When>
</Advice>
</PointCut>
</PointCuts>
</CommonCoreAsset>

<!-- Variations -->
<Variation type='DBConnectionPool'>
<ExtraPreConditions>
<ExtraPrecondition/>
</ExtraPreConditions>
<DBConnectionPool>
<Capacity>50</Capacity>
<ExpireTime>                                          AA
<!--check connection pool at 2AM everyday,
    and release connections, which exceed the
    maximal spare time;  in this example, the
    maximal spare time is set to 24 hours-->
<CheckPoint>02:00:00</CheckPoint>
<MaxSpareTime>86400</MaxSpareTime>
</ExpireTime>
</DBConnectionPool>
</Variation>
</Aspect>
```

From AFs, Aspect Generator generates aspect instances (AInsts) that are specific to a selected AOP platform. The generated AInst of the AF of DB Connection Pool is given in Figure 8.

**Figure 8** Aspect instance of DB connection pool

```
import java.sql.*;

public aspect StockDBConnectionPoolAspect  {
 // StockDBConnectionPool(int capacity, String
 //                   checkpointtime, long maxsparetime);
 StockDBConnectionPool stockDBConnPool =
       new StockDBConnectionPool(50, "02:00:00",
24*60*60);

pointcut connectionOpen(String url, String username, String
password) : call(public static Connection
          DriverManager.getConnection(String,String,String))
         && args(url, username, password)
         && !within(StockDBConnectionPoolAspect);

Connection around(String url, String userName, String
password)  throws SQLException : connectionOpen(url,
userName, password) {
 Connection connection =
   stockDBConnPool.getConnection(url, userName,
password);

   if(connection == null) {
     connection = proceed(url, userName, password);
     stockDBConnPool.registerConnection(connection, url,
                          userName, password);
   }
   return connection;
 }

 pointcut connectionClose(Connection connection)
         : call(public void Connection.close())
         && target(connection)
         && !within(StockDBConnectionPoolAspect);

 void around(Connection connection) :
        connectionClose(connection) {
   if(!stockDBConnPool.putConnection(connection)) {
     proceed(connection);
   }
 }
}
```

The Aspect Weaver weaves the generated aspect instances into the underlying component of the original web service according to the PCAS. The final adapted component source code is invisible to the developer. By deploying the adapted component, the new version of the web service is built and published to the service consumer.

Compared with other approaches, the benefits of GAIN are summarised below:

- the automated adaptation process will decrease the workload on software engineers and scale up the capacity of the approach
- since aspects are generated with SPL technique in response to specific adaptation requirements, these aspects will be very suitable for the particular target application.

## 7 Conclusions

Despite the increasing popularity of web services, the current state of art is that the service consumer has to use the web services as what they are because of the lack of adequate web service evolution mechanisms. In most cases, web services are subject to frequent evolution requirements and multiple versions need to co-exist due to the large diversity of potential users. The evolution may occur to both functional and non-functional aspects of a web service and requires to be done rapidly at low cost.

The proposed approach applies aspect-oriented generative adaptation to the underlying components of a web service to meet the evolution requirements of the web service so that the web service can be integrated into the target application smoothly. Automation and aspect-oriented deep level adaptation are the benefits of the approach. This is achieved with the following key techniques in an aspect-oriented component adaptation framework:

- the generation of adaptation aspects based on specific evolution requirements and selected abstract aspects as templates
- the advanced aspect weaving process definition mechanism that supports switch and synchronisation
- an expandable library of reusable adaptation aspects at multiple abstraction levels.

The approach enables web service developers to adapt their published web services to meet the integration requirement of specific web service applications. The benefits of the approach include deeper adaptability, higher automation and therefore smooth web service composition and wider reusability. As consequence, the target web service oriented systems will have better quality and more suitable functionality. Our case studies have shown that the approach and tool are promising in their ability and capability to meet the evolution requirements of web services.

## References

Arsanjani, A. (2002) 'Developing and integrating: enterprise components and services', *Communications of the ACM*, Vol. 45, No. 10, pp.31–34.

Baldwin, A., Shiu, S. and Mont, M.C. (2002) 'Trust services: a framework for service-based solutions', *Proceedings of 26th Annual International Computer Software and Applications Conference*, Oxford, England, August, pp.507–513.

Batory, D., Chen, G., Robertson, E. and Wang, T. (2000) 'Design wizards and visual programming environments for GenVoca generators', *IEEE Transactions on Software Engineering,* Vol. 26, No. 5, May, pp.441–452.

Batory, D., Johnson, C., MacDonald, B. and Heeder, D.V. (2002) 'Achieving extensibility through product-lines and domain-specific languages: a case study', *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 11, No. 2, April, pp.191–214.

Bennett, K.H. and Xu, J. (2003) 'Software services and software maintenance', *Proceedings of 7th European Conference on Software Maintenance and Re-engineering (CSMR'03)*, Benevento, Italy, March, pp.3–12.

Bosch, J. (1999) 'Superimposition: a component adaptation technique', *Information and Software Technology*, Vol. 41, No. 5, pp.257–273.

Chen, F., Li, S., Yang, H., Wang, C. and Chu, W. (2005) 'Feature analysis for service-oriented reengineering', *Proceedings of IEEE 12th ASIA-PACIFIC Software Engineering Conference*, December, Taipei.

Cleaveland, J.C. (1998) 'Building application generators', *IEEE Software*, Vol. 5, No. 4, July, pp.25–33.

Diana, L. and Webber, H.G. (2004) 'Modeling variability in software product lines with the variation point model', *ELSEVIER, Science of Computer Programming,* Vol. 53, pp.305–331.

Diaz-Herrera, J.L., Knauber, P. and Succi, G. (2000) 'Issues and models in software product lines', *International Journal on Software Engineering and Knowledge Engineering*, Vol. 10, No. 4, pp.527–539.

Ferris, C. and Farrell, J. (2003) 'What are web services?', *Communications of the ACM*, Vol. 46, No. 6, June, p.31.

Kajko-Mattsson, M. and Tepczynski, M. (2005) 'A framework for the evolution and maintenance of Web services', *Proceedings of the 21st IEEE International Conference on Software Maintenance*, Budapest, Hungary, September, pp.665–668.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W. (2001) 'Getting started with AspectJ', *Communications of the ACM*, Vol. 44, No. 10, October, pp.59–65.

Kleijnen, S. and Raju, S. (2003) 'An open web services architecture', *ACM Queue*, Vol. 1, No. 1, March, pp.38–46.

Kreger, H. (2003) 'Fulfilling the web services promises', *Communications of the ACM*, Vol. 46, No. 6, June, pp.29–34.

Lieberherr, K., Lorenz, D. and Mezini, M. (1999) *Programming with Aspectual Components*, March, Technical Report, College of Computer and Information Science, Northeastern University, NU-CCS-99-01.

Litoiu, M. (2004) 'Migrating to web services: a performance engineering approach', *Journal of Software Maintenance and Evolution Research and Practice*, Vol. 16, Nos. 1–2, pp.51–70.

Liu, X., Wang, B. and Kerridge, J. (2005) 'Achieving seamless component composition through scenario-based deep adaptation and generation', *Journal of Science of Computer Programming (Elsevier),* Special Issue on *New Software Composition Concepts*, Vol. 56, Nos. 1–2, pp.156–170.

Mezini, M. and Ostermann, K. (2005) 'A comparison of program generation with aspect-oriented programming', *Lecture Notes in Computer Science*, Vol. 3566, pp.342–354.

Sullivan, G.T. (2001) 'Aspect-oriented programming using reflection and meta object protocols – providing programmers with the capability to modify the default behaviour of a programming language', *Communications of the ACM*, Vol. 44, No. 10, October, pp.95–97.

Suvee, D., Vanderperren, W. and Jonckers, V. (2003) 'JAsCo: an aspect-oriented approach tailored for component based software development', *Proceedings of the 2nd International Conference on Aspect-oriented Software Development*, Boston, USA, pp.21–29.

Vanderperren, W., Suvée, D., Verheecke, B., Cibrán, M.A. and Jonckers, V. (2005) 'Adaptive programming in JAsCo', *Proceedings of the 4th International Conference on Aspect-oriented Software Development*, Chicago, USA, March, pp.75–86.

Viega, J. and Voas, J. (2000) 'Quality time – can aspect-oriented programming lead to more reliable software?', *IEEE Software*, Vol. 17, No. 6, November–December, pp.19–21.

Wang, B., Liu, X. and Kerridge, J. (2004) 'Scenario-based generative component adaptation in .NET framework', *Proceedings of the IEEE International Conference on Information Reuse and Integration*, Las Vegas, USA, November, pp.73–78.

Wilde, E. (2004) 'Semantically extensible schemas for web services evolution', *Proceedings of European Conference on Web Services (LNCS)*, Vol. 3250, pp.30–45.