Model-driven Composition of Context-aware Web Services Using ContextUML and Aspects

George N. Prezerakos
Technological Education Institute of Piraeus
Petrou Ralli & Thivon 250
122 44 Athens, Greece
prezerak@teipir.gr

Nikolaos D. Tselikas
National Technical University of Athens
Iroon Polytexneiou 9
15773 Athens, Greece
ntsel@telecom.ntua.gr

Giovanni Cortese
RadioLabs c/o Universita' di Roma Tor Vergata
Via Arrigo Cavaglieri 26, 00173 Roma, Italy
g.cortese@computer.org

Abstract

Service Oriented Architectures (SOAs) are constantly gaining ground for the provision of business to business as well as user-centric services, mainly in the form of Web Services technology. SOAs enable service providers to design and deploy new,composite service offerings out of existing component services. In order to match end-user expectations with respect to personalization and ease of use, these services should be designed in a manner that allows them to exhibit a certain level of context-awareness which is a basic element towards a richer end-user experience. However, in the majority of such services, context-handling is still tightly coupled with the core functionality of the service, resulting in a design which is difficult to implement and maintain. The paper proposes the decoupling of core service logic from context-related functionality by adopting a Model-driven approach based on a modified version of the ContextUML metamodel. Core service logic and context handling are treated as separate concerns at the modeling level as well as in the resulting source code where Aspect Oriented Programming (AOP) encapsulates context-dependent behavior in discrete code modules. The design of a restaurant finder service is used to portray the modified ContextUML metamodel and the service modeling process which is covered in full. Respective code snippets belonging to the executable version of the service (part of work in progress) are also provided, illustrating the transition from model to code and the resulting separation of concerns.

1 Introduction

One of the main perceived benefits of Service Oriented Architectures (SOA) is the ability to compose new services by using existing services as building blocks. The majority of research activities regarding service composition is focused on business to business interactions however there is recently relevant activity regarding user-centric services especially for mobile users which are obliged to use services under the well-known constraints posed by a mobile terminal [1].

Several strategies have been proposed for automatic service composition [2], on the other hand service providers can also experience benefits in assembling composite services off-line, either manually or semi-automatically, thus creating novel service offerings for their subscribers [3]. An additional area of service engineering research is concerned with the ability of such services to automatically adapt to contextual information that is related to the end-user and to other entities participating to service execution.

An interesting observation regarding such composite services is that although the interaction between the end-user and the service can be adapted to contextual parameters, the overall goals related to the service logic are usually indifferent to context changes. For example, consider the case of a simple business service that provides hotel room bookings. Depending on the end-user context (e.g. business or leisure trip, financial constraints, smoker/non-smoker etc.) different operations belonging possibly to different subservices can be invoked. In every case though, the overall goal "booking a room" is always the same regardless of the specific end-user context. According to this line of thought, core service functionality and handling of context can be viewed as two separate concerns with context handling cross-cutting into the core service functionality.

Therefore would it be possible to compose a context-aware service while at the same time keeping the dependencies between the context related behavior and the core service logic to a minimum ? Since context handling within

a service usually abides to a respective context model, this question can be tackled first at the modeling level (where a context model as well as a service logic model exist) and consequently at the coding phase of the composite service.

Decoupling core service logic from context handling would allow service providers to easily perform several enhancements to service compositions such as:

- Providing context-aware composite services by using 3rd party provided (context-agnostic) components
- Supporting run-time, dynamic selection of different implementations of a service feature, based on current context conditions
- Enriching the context model in use with minimal modifications to the service logic model
- Plug-in/out different context models with minimal modifications to the service logic model

Moreover, the ability to augment a composite context-aware service with additional component services can lead to a scenario where a base version of such a service (e.g. a museum guide, a travel planner etc) can be extended by other individuals to suite the need of different businesses. For example a service developed for an airport shopping area becomes the foundation for a service targeted towards a shopping mall.

The paper proposes a solution to the above challenges by adopting a Model-Driven approach [4] in the design phase and Aspect Oriented Programming (AOP) [5] during coding. A restaurant finder service has been used as a reference scenario in conjunction with an existing context metamodel. Web Services are used throughout the paper as service building blocks for the reference scenario, however the modeling principles presented herein can also be applied to composite services besides the Web Service paradigm (e.g. Jini services) using several different context metamodels. An additional requirement is that the design and coding processes should be based on existing, off-the-self technologies effectively leading to a more gentle learning curve for the software practitioners involved.

The use of UML modeling in the design phase of an application as well as the possibility to automatically create source code from UML models is a highly debated issue [6] which is beyond the scope of this work. It is assumed that a significant percentage of the software engineering community perceive concrete benefits in the use of UML models during the service design phase. The following section presents related research efforts that inspired the main ideas contained in this paper.

2. Initial motivation and related work

One way to achieve the separation of business logic from context-dependent behaviour is to have the service in question composed of a service logic skeleton responsible for the main business goals of the service combined with several code stubs that would handle context-dependent functionality. This idea of a skeleton and stub approach originates from the work carried out in [7] under the general term *Context-Oriented Programming (COP)*. COP treats context as a first order construct of the programming language that is used for application development. Therefore such an application may contain statements, terms, operators, method calls, expressions etc. as open terms inside a service logic skeleton. Program execution takes place by dynamically matching these open terms with code stubs, characterized by a goal and a context value, in a context dependent fashion. Context is being handled according to a simple model but the issue of application design has been left open. Therefore the developer has to take care of interrupting the service logic skeleton at appropriate points by invoking a `fill_gap()` function. Moreover the degree of distribution is minimal, the service skeleton is executed at the client while the stubs are retrieved from a remote database. Several example COP applications have been provided using the Python programming language, which being an interpreted language with flexible scoping lends itself nicely to the skeleton and stub approach. However it would be more difficult to adopt the same service creation process using a compiled language with less flexible scoping behaviour.

A similar approach that relies on the interruption of the main execution path in order to provide for additional functionality has been presented in [8, 9]. The functionality of an E-commerce application has been augmented by weaving relevant business rules with the application's core business logic using a strategy termed *Hybrid Aspect Weaving*. The same strategy has also been applied in [10] where business rules have been treated as concerns cross-cutting the execution of a Business Process Execution Language (BPEL) flow. The idea of incorporating context-related behavior into aspects is dealt with in [11] where an open reflective framework for Java called Reflex has been used to extend the Java language with constructs dedicated to context handling. These approaches provide solutions for dealing with separation of concerns (including context awareness) but at the level of source code, therefore there is no provision regarding how the application will be designed, moreover the issue of context modeling is not examined at all.

An interesting approach for the transition from model to code, taking into account cross-cutting concerns, is the use of *Subject / Theme* oriented modeling. This approach

has been followed in [12, 13] and encapsulates cross-cutting concerns as separate subjects inside an application model. In the case of a Web Services-based architecture discussed in this paper, context-dependent parameters, context-dependent operation invocations as well as the adaptation of input/output parameters can be modeled as separate subjects and consequently binded with the respective constructs of the actual service to be invoked. Such subject-oriented models can convey all the necessary information for transforming the model into high-level source code however the resulting UML diagrams are quite complex and therefore are not very easy to handle during the modeling process. Therefore we did not pursue subject-oriented modeling any further although the <<bind>> attachment pattern has been adopted for use in the UML diagrams contained in this paper.

Summarizing the main ideas presented above, the main aim is the creation of context-aware composite services out of existing services belonging to third parties (and thus cannot be easily modified) using, as much as possible, off the self software tools. Since third parties services cannot be easily modified, the composite service developer is obliged to handle context-awareness only at the service interface level that consists of operation invocations and the exchange of respective input/output parameters.

The service creation process starts from a UML service model, which, in the resulting service implementation, should be able to support a skeleton and stub approach by treating context as a cross-cutting concern. In addition, service modeling should not contain any explicit reference to any architecture that supports cross-cutting concerns because focus should be placed on modeling the design instead of modeling the implementation. This way of modeling leaves ample room for using technologies other than AOP during the implementation phase. Moreover the service model should support in a clear and well defined manner a respective context model while at the same time the dependencies between the two models should be kept at the absolutely necessary minimum.

Both the service and the context models are created according to one or more respective metamodels. Such a metamodel, which is the basis for the presented service scenario, is described in the next section.

3. The ContextUML metamodel

The metamodel which is used for the creation of service and context models that support the reference scenario is ContextUML [14]. It constitutes one of the first approaches for providing a generic metamodel in UML that can be used for context-aware services. The main philosophy of ContextUML is that service constructs are in fact context-dependent thus they can be associated with context attributes described by a relevant context model. Within this model, context attributes can be populated by invocation of respective operations that belong to context source services.

In the course of this paper, several modifications were applied to the ContextUML in order to (1) minimize the association between the way the services are modeled and the way context is (2) eliminate unnecessary relationships between existing meta-model artifacts and (3) clarify existing metamodel semantics in order to facilitate the translation of the resulting service model into code by a model translation tool. This activity led to a modified ContextUML model which is presented in Figure 1.

The differences between the original and the modified ContextUML metamodel fall broadly into two categories: differences in the stereotypes contained in the metamodel and differences in the use of these stereotypes during the creation of a service or context model. The former differences are discussed in the next couple of paragraphs while the latter are presented in Section 5 which discusses model design.

In the original ContextUML metamodel all the stereotypes related to Web Services inherited from the <<CAObject>> metaclass. In turn this metaclass was associated with context elements via the <<CAMechanism>> metaclass. The use of inheritance introduces an intermediate class between Web Service constructs and context thus in the modified metamodel the association is straight-forward via the <<ContextBinding>> dependency relationship which existed in the original metamodel as an association class. The same holds for the <<ContextTriggering>> dependency relationship which also existed in the original metamodel as a class consisting of at least one operation and at least one constraint. The use of dependency relationships allows for the direct depiction of the relation between service elements (operations, parameters) and context elements as well as between context elements and their respective context sources.

Also a context <<Owner>> class associated with the <<Context>> class has been added to the metamodel according to the context model described in [15]. Web services (left hand side of the figure) are modeled differently with respect to the original ContextUML metamodel using a convention widely adopted in literature and in UML modeling tools [16]. Finally, the modified ContextUML metamodel assumes that context sources also exist as Web Services although there is no limitation in the metamodel that prevents the use of other types of sources.

The following reference scenario is the basis for the creation of respective service and context models according to the modified ContextUML metamodel .

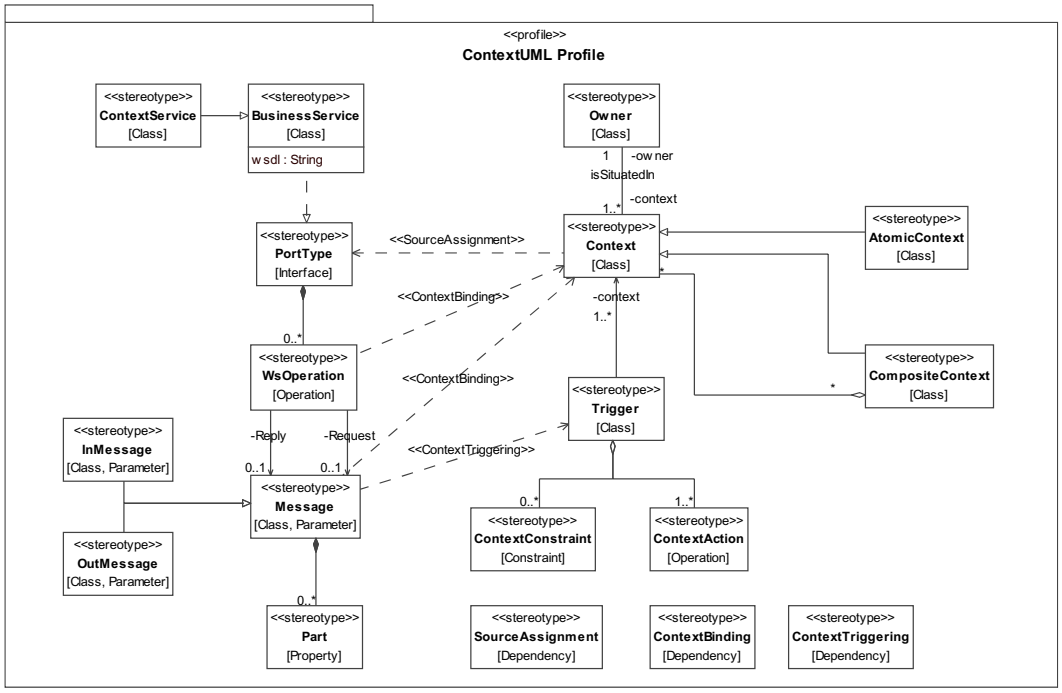

Figure 1. The modified ContextUML meta-model

4. Reference scenario

A service provider decides to offer a service to (mainly) out-of-towners who need some guidance on how to spend their free time. This is a composite service assembled from a greeting service, a restaurant finder service and a route planning service. Initially the greeting service is invoked that greets the user with a welcome message in his/her native language.

Consequently the restaurant finder service exploits the user's location as well as his/her culinary preferences to assemble a list of restaurants in the user's general vicinity. Finally the route planner is invoked which provides routes to the restaurant selected by the user. In case of bad weather at the user's current location, routes involving walking in open air for more than a few minutes are filtered out.

5. The modeling process

The process of modeling such a composite service, such as the one described in the previous section, consists of the following steps: (a) Discovery of relevant Web Services and their translation to UML diagrams (b) Design of a context and service logic model based on these ML diagrams and the ContextUML metamodel, (c) Association of context model elements with respective context sources, (d) Association of service logic elements with respective context model elements and (e) manual assembly of the discovered Web Services into a composite workflow using a UML activity diagram.

Several strategies exist by which the discovery of the required Web Services can be carried out. An interesting approach has been described in [3] where the WSDL files corresponding to the discovered services are translated to UML class diagrams following the conventions described in [16]. Consequently these diagrams are imported to a UML modeling tool. In the case of the reference scenario, the respective UML diagram is contained in Figure 2.

As it has been mentioned in Section 4, there are three categories of business services, the Welcome, FunFinder and RoutePlanner services respectively. The Welcome business service consists of at least one Web Service, namely the Hello Web Service, which requires as input the country where the user comes from as well as the user's name and returns a greeting in the user's native language. The FunFinder business service consists of Web Services that propose recreational activities in metropolitan areas.

Among them there are the FindRestaurants and FindAttractions Web Services. In this scenario, the FindRestaurants Web Service consists of two operations for retrieving restaurants with French and Chinese cuisine respectively. Both operations require the name of an area within a city as input. Finally the RoutePlanner business service offers an operation belonging to the CityRoutes Web Service which

calculates the possible routes from the end-user's current location to the selected restaurants. The context elements

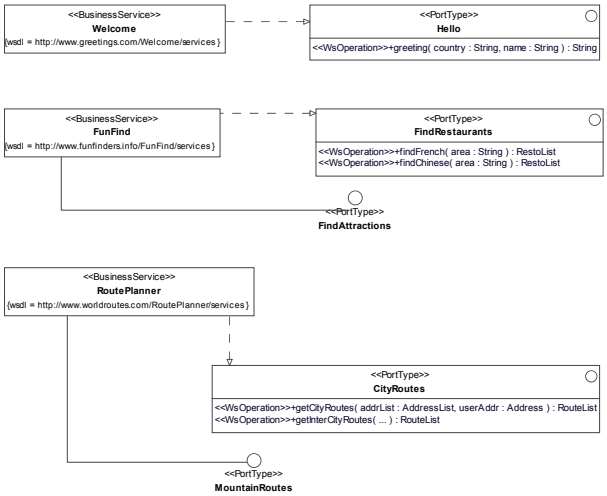

Figure 2. Discovered Services

used by the operations in question consist of attributes of the user profile, location and weather info. The respective context model (based on ContextUML) is presented in Figure 3. The `<<OperationAssignment>>` dependency relationship associates context attributes with respective context sources that provide values for these attributes. It should be noted that in the case of the operations belonging to the `MainWeatherParams` service, the input parameter `city` is context-dependent as well and it is associated with `Location` context class via a `<<ContextBinding>>` dependency relationship. In the case where there are multiple dependencies between elements of two classes, it is more convenient to use the `<<bind>>` attachment pattern proposed in [12] instead of drawing multiple dependency arrows.

The next step in the modeling process is to associate service operations and parameters with attributes from the context model as well as with transformation operations that might affect either input or output parameters depending on context (Figure 4). This is achieved via the `<<ContextBinding>>` and the `<<ContextTriggering>>` dependency relationships respectively. Similar to Figure 3 the `<<bind>>` attachment pattern is used in the case of multiple dependencies.

The `greeting(...)` operation of the `Hello` Web Service is an example of handling context-awareness at the input parameter level. The input parameters `country` and `name` have to be associated with the `country` and `name` attributes contained in the subscriber's profile. The `findFrench(...)` and `findChinese(...)` operations of the `FindRestaurants` Web Service contain context dependent input parameters as well but the operations themselves are also dependent to attributes in the end-user's profile. More specifically, the invocation of either operation depends on the value contained in the `cuisine` attribute of the end-user profile. In the same figure, it is shown that the result from the invocation of the `getCityRoutes(...)` operation is adapted according to the weather conditions that exist at the end-user's location. Figure 5 contains the model of the

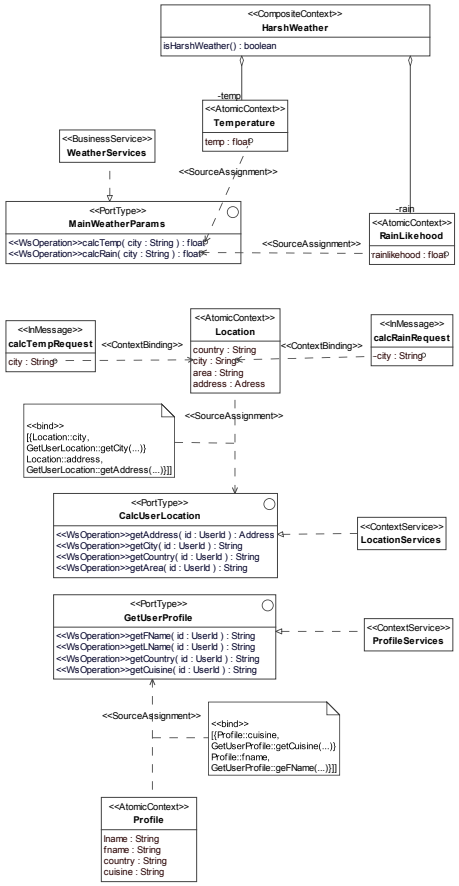

Figure 3. Context model

core service logic by means of a UML activity diagram using relevant notation from [3]. The `<<Alternative Service Providers>>` stereotype realizes the Selector pattern while the `ImmediateStep` stereotype corresponds to a piece of code that transforms the list of restaurants selected by the end-user to a respective list of addresses to be used by the `getCityRoutes(...)` operation. It is evident that the service skeleton has remained relatively simple since context-dependency has been moved out of the core business logic.

Contrary to the work performed in [17, 18] aspects are not explicitly included either in the meta model or in the re-

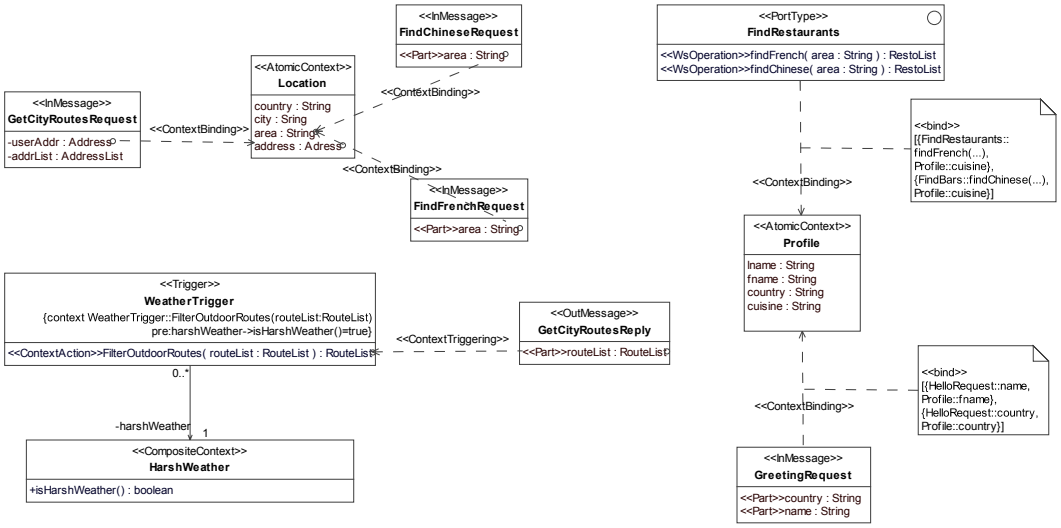

Figure 4. Modeling context dependencies

sulting model. During the modeling phase of the service, the designer should not be concerned with the particular programming architecture that must be followed in order to provide for context-awareness at the source code level. Therefore the explicit modeling of aspects would have violated the necessary separation between the modeling and the coding phases. This approach is also consistent with similar model-based approaches (e.g. [19]) where the use of aspects is implicit in the service model.

6. From model to code

This section describes issues and initial lessons learned with respect to the structure of Java code that is generated from the respective service and context models contained in Section 5. Specific implementation issues that arise because of the transition from the UML model to the high level source code are instead discussed in detail in Section 7. In the approach discussed below, service and context UML models are exported as XMI files and fed into a model translation tool which, driven from the stereotypes of the ContextUML model, converts them into Java source.

The usage of an AOP engine is assumed as the mechanism that enables the extension of existing services behavior. Context binding information (Figure 4) provided in the UML models is used to create pointcuts and related advices, as well as to create the binding between them. Typical logic included as part of the advice performs (1) context-dependent value assignment to input parameters of web services (2) the selection among different alternative web services to be invoked and (3) the transformation of context dependent input/output parameters.

If the invocation of an operation in the main service logic flow is in any way context-dependent then it is intercepted by a respective advice contained in an aspect class. Consequently the context-related activities in question, such as the population of context-related input parameters, the selection of operations to be invoked and the transformation of context related input/output parameters, take place inside the advice and the invocation of the operation continues. Figure 6 contains a part of the main service logic of the RestaurantFinder composite service.

The code snippets provided in the following paragraphs use the aspect-oriented version of Java known as AspectJ[1] and target the combination of the Apache Tomcat[2] Web Server and the Axis[3] SOAPengine. The Axis platform allows the invocation of a Web Service operation either via the initialization of a `Call` object or via the use of proxy objects. Both approaches have been used in Figure 6 where the greeting operation is invoked via a `Call` object while the other two web service operations are invoked by means of proxy classes. Join points in Figure 6 are indicated by the black arrows beside the grey rectangles.

A similar approach can be followed in the case where the core service logic is created using e.g. the Business Process Execution Language for Web Services (BPEL-WS)[4]. However in that case either a special version of BPEL that supports aspects must be used [10, 20] or the operation invocations in the BPEL flow have to be wrapped into Web Services executing locally (at the composite ser-

[1] http://www.eclipse.org/aspectj/
[2] http://tomcat.apache.org/
[3] http://tomcat.apache.org/
[4] http://ifr.sap.com/bpel4ws/

vice provider's server) in order for aspects to be supported. This of course introduces an additional service execution layer in the form of web service wrappers. Figure 7 con-

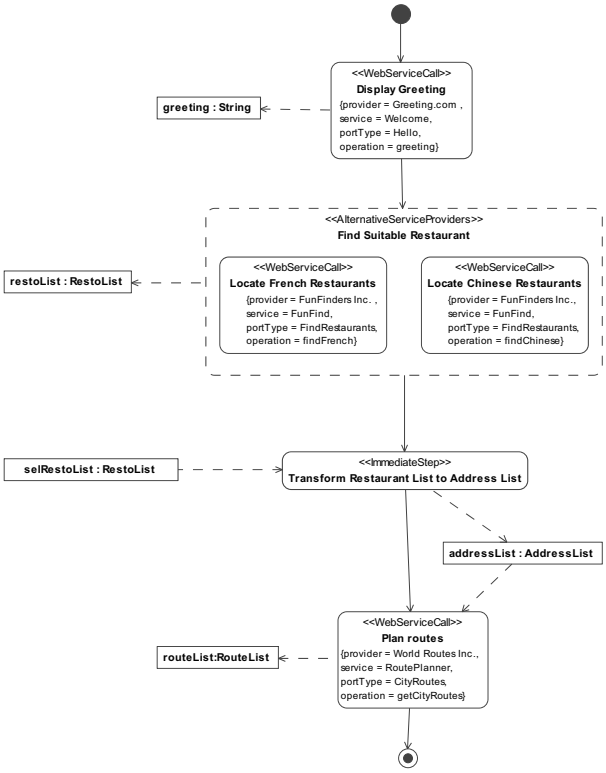

Figure 5. Service logic skeleton

tains the Hello Web Service code as well as the respective aspect class that is responsible for context handling. The execution of the `greeting(..)` operation in the `FinderPlanner` class is intercepted by the advice contained in the `Greeting` aspect class. The context dependent parameters `name` and `country` are populated inside the advice and the execution of the `greeting(...)` operation resumes.

In a similar fashion the invocation of either the `findFrench(...)` or the `findChinese(...)` operation in the `FinderPlanner` class is intercepted in order for the end-user's culinary preferences to be retrieved from the user profile. Depending on the retrieved value the respective operation of the FindRestaurants web service is invoked .

Finally, in the case of the `getCityRoutes(...)` operation, the outcome of this operation needs to be processed further only in the case of bad weather conditions. Depending on the return value from the `isHarshWeather()` method call, the outdoor routes are filtered or not. From the above code snippets, it is quite clear that the same level of separation of concerns that has been achieved at the modeling level is also possible at the coding level via the use of aspects.

FinderPlanner.java

```
class FinderPlanner {
 public void R_Finder() {
      ...
   String hello="";
   try {
    String endpoint ="http://www.greetings.com/
    Welcome/services/HelloWS";
    Service  service = new Service();
    Call call    = (Call) service.createCall();
    call.setTargetEndpointAddress(
     new java.net.URL(endpoint) );
    call.setOperationName(new QName(
     "http://soapinterop.org/", "greeting"));
    Object [] obj = new Object[] { null, null };
 ▶  hello = (String) call.invoke(obj);
   } catch (Exception e) {
     System.err.println(e.toString());
   }

  ...//user to service interaction

   FindRestaurantsWSProxy frp =
    new FindRestaurantsWSProxy();
 ▶ RestoList restoList =
    frp.findFrench(null);

  ...//user to service interaction

   AddressList adresslist =
    TransformRestoList(selRestoList);
   CityRoutesWSProxy crp =
    new CityRoutesWSProxy();
 ▶ RouteList routelist =
    crp.getCityRoutes(null, al);

  ...//user to service interaction
}
```

Figure 6. Restaurant Finder core service logic

In the presented example AOP is used mainly as an interception technique. Since most web service frameworks allow some form of interception, what is it that makes AOP the preferred choice for implementation phase? First of all, AOP is perceived as a very flexible and easy to use interception technique that can lead to a clearer and more compact notation when compared to e.g. a servlet filter. The real benefit, however, comes from the ability of AOP at expressing "cross-cutting" concerns. With aspects, it is easier for the developer to describe a set of related transformations, which impact several components in an application, in a single place or module. In a commercial application, for each CompositeContext (such as "harsh weather"), several operations should be advised. Even in the example scenario, which is relatively simple, it is likely that a harsh

weather context may have several effects on the application functionality therefore it might be desirable to group these effects in a single module. In addition to the selection of routes in the example scenario, other features of the composite service are obvious candidates for coding in the form of aspects (for example, recommendation of most appropriate recreational activity, other than eating at a restaurant, by the FunFinder service).

HelloWS.java

```
public class HelloWS implements Hello {
  public String greeting (String name,
  String country) {
    if (country.equals("France"))
      return "Bonjour"+" "+name;
    else if (country.equals("Sweden"))
      return "Hey da"+" "+name;
    return "Hello"+" "+name;
  }
}
```

Greeting.aj

```
import org.apache.axis.client.Call;

public aspect Greeting {
  Object around(Call c, Object [] obj):
    target (c) && args(obj)
    && call (Object invoke(Object []))
    && withincode(
    void FinderPlanner.R_Finder()) {
      GetUserProfileProxy pp =
        new GetUserProfileProxy();
      obj[0] = pp.getFName(Session.uid);
      obj[1] = pp.getCountry(Session.uid);
      return proceed(c,obj);
  }
}
```

Figure 7. Hello Web Service interception

A related issue is the choice of appropriate mapping techniques from models to AOP constructs. There are several options for this purpose (i.e. one aspect per advised operation, group advices by pointcut, group advices by context etc.) Selecting the most suitable one gets more relevant especially if the aim is to generate code which is readable and modular. As part of ongoing research, we are exploring proper abstractions and design guidelines for the purpose of mapping context-dependent variations to Java classes and AOP constructs, in a way that is easy to read and maintain.

7. Implementation issues

The transformation of UML models to high-level service code can be based either on open source model translators, namely the UML Model Transformation Tool (UMT) [5] and AndroMDA [6] or on the development of a custom model translator. Both approaches are currently evaluated in the

[5] umt-qvt.sourceforge.net

[6] http://www.AndroMDA.org/

Find.aj

```
public aspect Find {
  RestoList around(String area):
  call (RestoList findFrench(String))
  && args(area) && withincode(
  void FinderPlanner.R_Finder()){
    RestoList rl=null;
    CalcUserLocationProxy culp =
      new CalcUserLocationProxy();
    area = culp.getUserArea(Session.uid);
    GetUserProfileProxy gupp =
      new GetUserProfileProxy();
    String cuisine =
      gupp.getCuisine(Session.uid);
    if (cuisine.equals("French"))
      rl = proceed(area);
    else if (cuisine.equals("Chinese")) {
      try {
        FindRestaurantsWSProxy frp =
          new FindRestaurantsWSProxy();
        rl=frp.findChinese(area));
      } catch (java.rmi.RemoteException re) {
        System.err.println(re.getMessage());
      }
    }
    return rl;
  }
  ... //around advice for the
  ...//findChinese() operation goes here
}
```

Routes.aj

```
public aspect Routes {
  RouteList around (CityRoutesProxy crp,
  AddressList al, Address addr) :
  target(crp) && args(a, al) &&
  call(RouteList getCityRoutes(Address,
  AddressList)) && withincode(
  void FinderPlanner.R_Finder()){
    RouteList rl=null;
    try {
      CalcUserLocationProxy culp =
        new CalcUserLocationProxy();
      addr =
        culp.getUserAddress(Session.uid);
      if (HarsWeather.isHarshWeather())
        rl= FilterOutdoorRoutes(
          proceed(crp,al, addr));
      else rl = proceed(crp,al, addr)
    } catch (java.rmi.RemoteException re) {
      System.err.println(re.getMessage());
    }
    return rl;
  }
}
```

Figure 8. FindRestaurants and CityRoutes Web Service interception

framework of our ongoing work. It has already been shown

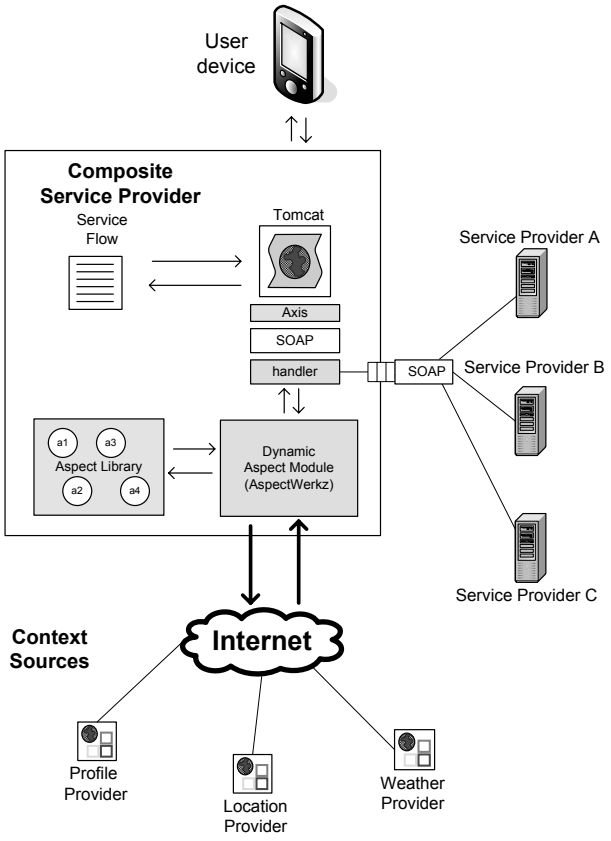

Figure 9. Context-aware WS compositions using Dynamic Aspects

that the use of AspectJ can provide the necessary separation of concerns, however AspectJ also presents one drawback: any modification to the model in use (either to the relation between service elements and context or to the relation between context and context sources) would require the retranslation of the model to source code. This is because AspectJ provides aspect weaving at compile-time. Compile-time aspect weaving is acceptable when service adaptations have a service-wide scope thus they are identically perceived by all users and are not modified as long as the service is up and running. A more flexible service behavior can include session/user and conversation/request scoped adaptations. Session/user scoped adaptations require the service logic to select among alternative implementations of a context-sensitive feature for each user while conversation/request scoped adaptations require the service logic to select the most proper implementation of a feature very often, possibly at every interaction with the user. Using aspects to achieve such a behavior requires aspect weaving during load-time or even during run-time.

In addition the Axis SOAP engine allows the inclusion of dedicated software filters called handlers that can intercept the exchange of SOAP messages between communicating entities. By using such a handler the application of advices can be moved from the application code into the handler code. Respective changes to the composite service model can be applied directly to the handler (e.g. by using the model to produce a configuration file), effectively bypassing the core service logic which will not require re-compilation. Consequently the handler can dynamically apply aspects directly on the intercepted SOAP messages in order to provide for context-dependent behaviour. In such a case, the handler will be able to dynamically turn aspects on and off, to introduce new aspects etc. and to introduce more personalized features corresponding e.g. to the requirements of individual users or groups of users. Such an architecture, which constitutes the goal of our current implementation work is presented in Figure 9.

SOAP messages going in / coming out of the Web server are intercepted by the handler. A framework, which provides dynamic aspect handling (e.g. AspectWerkz[7]), mediates between the handler and the underlying aspect library. It evaluates context information and depending on the retrieved context parameters, appropriate advices are applied which transform the intercepted SOAP messages.

8 Conclusions and future directions

The ability to effectively handle context with minimal user intervention, especially for the provision of services that run on resource -constraint mobile devices is becoming increasingly important. The effective use of Model-driven techniques allows service developers to handle business logic and context-dependent service behaviour as two separate concerns within the same model, enabling the modification of the context dependent behaviour without severely affecting the main functionality of the service. The same separation of concerns can be achieved at the source code level and consequently during service operation by encapsulating context-dependent functionality into separate aspects in a high level programming language. This approach extends the advantage of interoperability and loose coupling between clients and servers as well as between composite and component services offered by SOA to the context handling domain. Along this line of thought, the paper has discussed the application of Model-driven design and Aspect Oriented Programming to a RestaurantFinder service using readily available software platforms and the AspectJ programming language.

Apart from the completion of the implementation activities described in Section 7, an additional area for further

[7] http://aspectwerkz.codehaus.org/

research concerns the creation of a framework dedicated to context-handling which will be based on Plain Old Java Objects (POJOs) in the footsteps of other frameworks for Java such as Spring and Hibernate [21]. Using Java 5 as the service development platform, it would be possible to include the relation between service elements and context attributes directly into the source code in the form of Java annotations. Alternatively it would be also possible to describe these relationships in a separate XML configuration file. Also, the relationship between context and context sources can be specified either in this configuration file or by annotating the WSDL files that describe context source services or even by modeling only this relationship in UML and converting the resulting model to annotated Java code or XML (since most context-aware applications are based on some form of context-modeling in any case).

Such a framework can be easily used to extend the architecture described in Section 7 resulting in an end-to-end solution for the design and implementation of context-aware Web Service compositions.

Acknowledgment

The ideas that are described in this paper originate from work performed within the IST Simple Mobile Services (SMS) project (http://www.ist-sms.org) but represent the personal elaboration of the authors.

References

[1] G. Bartolomeo, N. Blefari-Melazzi, G. Cortese, A. Friday, G. Prezerakos, R. Walker, and S. Salsano. SMS: Simplifying mobile services - for users and service providers. In *Proc. AICT/ICIW 2006*, page 209, Guadeloupe, French Caribbean, February 19-25 2006.

[2] N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, pages 51–59, November-December 2004.

[3] D. Skogan, R. Gronmo, and I. Solheim. Web service composition in UML. In *Proc. 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC)*, Monterey,CA, USA, September 2004.

[4] OMG. MDA executive overview. available online at http://www.omg.org/mda/.

[5] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming. *Communications of the ACM*, 44(10):29 – 32, October 2001.

[6] R. L. Glass. On modeling and discomfort. *IEEE Software*, 21(2):104–103, Mar-Apr 2004.

[7] R. Keays and A. Rakotonirainy. Context-oriented programming. In *Proc. Third International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE03)*, pages 9–16, San Diego, California, USA, September 19 2003.

[8] M. D'Hondt and V. Jonckers. Hybrid aspects for weaving object-oriented functionality and rule-based knowledge. In *Proc. International Conference on Aspect-Oriented Software Development (AOSD 04)*, pages 132–140, Lancaster, UK, March 2004.

[9] B. Verheecke and V. Jonckers. Stateful aspects for conversational messaging with stateful web services. In *Proc. International Conference on Next Generation Web Services Practices (NWeSP05)*, Seoul, Korea, August 22-26 2005.

[10] A. Charfi and M. Mezini. Hybrid web service composition:business processes meet business rules. In *Proc. 2nd International Conference on Service Oriented Computing (ICSOC'04)*, pages 30–38, New York, USA, November 15-19 2004.

[11] E. Tanter, K. Gybels, M. Denker, and A. Bergel. Context-aware aspects. In *Proc. 5th International Symposium on Software Composition (SC 2006)*, Springer-Verlag LNCS, pages 227–249, Vienna, Austria, March 2006.

[12] G. Tandon and S. Ghosh. Using subject-oriented modeling to develop Jini applications. In *Proc. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 111–122, September 20-24 2004.

[13] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In *Proc. 26th International Conference on Software Engineering (ICSE04)*, pages 158–167, Edinburgh, Scotland, May 23-28 2004.

[14] Q. Z. Sheng and B. Benatallah. ContextUML: A UML-based modeling language for model-driven development of context-aware web services. In *Proc. of the International Conference on Mobile Business (ICMB05)*, pages 206 – 212, Sydney Australia, July 11-13 2005.

[15] A. Zhdanova, N. Snoeck, H. van Kranenburg, M. Marengo, M. Valla, J. Zoric, M. Sutterer, O. Droegehorn, C. Rack, and S. Arbanowski. Context acquisition, representation and employment in mobile service platforms. In *Proc. 15th IST Mobile & Wireless Communications Summit*, Mykonos, Greece, June 2006.

[16] D. Skogan, R. Gronmo, I. Solheim, and J. Oldevik. Model-driven web services development. In *2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-04)*, pages 42 – 45, Taipei, Taiwan, 2004.

[17] D. Dahiya and R. Sachdeva. Approaches to aspect oriented design : A study. *ACM SIGSOFT Software Engineering Notes*, 31(5):1–4, September 2006.

[18] J. Iborra, O. Pastor, and V. Pelechano. Dealing with crosscutting concerns in a model based software production method. In *International Conference on Software Engineering (ICSEEA06) - Proceedings of the 2006 international workshop on Early aspects*, Shanghai, China, May 21 2006.

[19] S. Clarke and R. J. Walker. Composition patterns: An approach to designing reusable aspects. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 5–14, Toronto, Ontario, Canada, May 12-19 2001.

[20] C. Courbis and A. Finkelstein. Towards an aspect weaving bpel engine. In *Proc. Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS'04)*, Lancaster, UK, March 2004.

[21] C. Richardson. Untangling enterprise java. *ACM Queue*, 4(5):36–44, June 2006.