Unraveling Crosscutting Concerns Web Services Middleware

Bart Verheecke, Wim Vanderperren, Viviane Jonckers
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
{Bart.Verheecke;Wim.Vanderperren;Viviane.Jonckers}@vub.ac.be

Abstract

With the emergence of Web Service Technology, the need arises for techniques to realize just-in-time integration and composition of services in client-applications. Current approaches to integrate web services are rather inflexible, affecting short-term adaptability and long-term evolution towards the service, the network and the business environments. To enable the development of more flexible and robust applications we propose the Web Services Management Layer (WSML) for the dynamic integration, selection, composition and client-side management of web services in Service-Oriented Architectures (SOA). In this paper we identify several crosscutting concerns in a SOA and show how dynamic AOP can be used to solve them. A realistic industrial case study of the WSML in the context of broadband service provisioning is presented.

Keywords

D.2.0.c Software engineering for Internet projects, D.2.7.e Evolving Internet applications, D.2.7.g Maintainability, D.2.12.b Distributed objects, D.2.11.b Domain-specific architectures

1. Introduction

Service-Oriented Architecture (SOA) is an application architecture designed to achieve loose coupling among interacting software applications. Using Web Service technology a distributed application can be created in a heterogeneous environment. The ultimate goal of SOA is to be able to write applications independently of the concrete services used and to select and to integrate services on the fly. Currently, web services are described in WSDL-format and published in a registry. Service clients can browse these registries to find a matching service and determine how to communicate with it. By analysing the WSDL documentation, the client can integrate the service and invoke it through XML-based SOAP communication.

Basically, there are two ways to create a SOA: the first scenario involves the implementation of an inter- or intra-organisational process with a fixed number of partner roles. Based on an orchestration or workflow specification, each partner implements a web service to fulfil a specific role in the business process. Any modification to the process implementation in a later stage requires a new agreement between the partners before the modification can be deployed. The second scenario takes more advantage of the loosely coupled nature of web services: a client application is build independently from any concrete services; partner roles are specified that need to be filled in by services at runtime. In this approach, *just-in-time integration of services* becomes a crucial process: the entire process of service discovery, selection, integration and invocation is deferred until runtime.

As a motivating example, imagine a web application that allows flights and hotels to be booked for a customer. This system needs to integrate with dozens of different airline company services and hotel reservation systems. Depending on continuously evolving business requirements and changing network and service conditions, different services will be integrated at a given time. However, automating this process is far from straightforward: the web services belong to different domain controllers and as a result might differ on several points including syntactical and semantical differences in the service interface, security measurements, Quality-of-Service, billing mechanisms, etc. All of these variations need to be reflected in the client application, which clearly is a hindrance for a smooth integration process.

Furthermore, using traditional programming techniques, the code dealing with these concerns will result tangled and scattered in the client. A better alternative is to dynamically identify, compose, invoke and manage the appropriate web service(s) independently from the client. In [1], the Web Services Management Layer (WSML) is proposed as an adaptive middleware layer, placed in between the application and the world of web services. The WSML allows dynamic selection and integration of services into an application, client-side service management, and support for service criteria based on non-functional properties that govern the selection, integration and composition.

This paper discusses how Aspect-Oriented Programming (AOP) [2] has been adopted during the development of the WSML. It gives a broad systematic overview and discussion of several identified aspect categories and mentions the different AOP techniques used. AOP aims to provide a better separation of concerns by capturing crosscutting concerns in a new kind of module called *aspects*. An aspect consists of two parts: a *pointcut*, which describes where the aspect needs to be applicable and an *advice*, which describes the actual behavior that needs to be executed at the places where the

aspect is declared to be applicable. These places are named *joinpoints*.

AOP is well suited to build the core functionality of this management layer as service communication details, selection policies and management concerns are all suitable candidates to be modularized in aspects. By exploiting dynamic AOP, the necessary flexibility is provided for successfully realizing just-in-time service integration. The following sections introduce the requirements for just-in-time service integration and motivate why dynamic AOP is well suited for realizing this. Afterwards, the architecture and a prototype of the WSML are presented and the different aspects used in the WSML are explained. The core technology of the WSML is JAsCo [3], a highly-performing state-of-the-art dynamic AOP language. Finally, a realistic industrial case study of the WSML in the context of broadband service provisioning for Video-on-Demand systems is presented.

2. Just-in-time integration of web services

Runtime integration of unanticipated web services in client applications is a complex process including, but not limited to:

- **Service Discovery:** web services that deliver the required functionality for a client need to be looked up on the internet. Services must be semantically compatible in order to be integrated.
- **Service Selection:** if multiple web services or service compositions are available for a given client request, the most optimal service or composition must be determined, based on a set of selection policies.
- **Service Integration:** to invoke a remote service, a client-side stub needs to be created and the appropriate method(s) must be invoked on it. This process must offer support to deal with compositional mismatches between the concrete service interface and what the client expects.
- **Service Composition:** if services are only partially compatible, it might be required to combine multiple services together in order to deliver the required functionality.
- **Service Management:** invoking services belonging to different domain controllers requires monitoring, advanced exception handling, security, Authentication, Authorization, and Accounting (AAA), billing, etc.

The WSML offers an AOP-enabled reusable framework dealing with the last four processes. The WSML contains all service-related code, nicely separated from the code of the client application. Web services, found on the internet, can be registered in the WSML. Based on the WSDL-description of these services, a composition specification can be made to describe how the services should cooperate to deliver the required functionality for the client. A flexible integration mechanism based on dynamic AOP, deals with the invocation, selection and client-side management of the appropriate web services and service compositions. The following section motivates the need for AOP in general and dynamic AOP in particular.

3. Motivation for AOP

Code fragment 1 shows a typical piece of Java code required to invoke a remote web service. The code deals with various concerns including redirection, user authentication, the actual invocation, logging and exception handling. Clearly, the code for each of these various concerns is *tangled*. Moreover, in other places in the core application where a service invocation is required, similar or even identical code can be found: the code is also *scattered*. Note that the code fragment is based on the use of a static stub: clearly, the use of dynamic stubs makes the code even more complicated, especially if specific policies are applicable to determine which stub (and thus which service) to invoke. These policies, driven by constantly evolving business requirements, might need data from various sources including the web service documentation, the web service behaviour or the client state. All these points need to be intercepted to gather the required data, which becomes an impossible task if the system needs to deal with unanticipated selection policies.

```
package staticstub;
import javax.xml.rpc.Stub;

public class HelloClient {

  private String endpointAddress;

  public static void main(String[] args) {

    try {                                                          Redirection
      endpointAddress = args[0];
      Stub stub = createProxy();
      stub._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY, endpointAddress);
      HelloIF hello = (HelloIF)stub;
      stub._setProperty(Stub.USERNAME_PROPERTY, username);
      stub._setProperty(Stub.PASSWORD_PROPERTY, password);         Authentication
      String result = hello.sayHello ("Duke!");                    Invocation
      System.out.println(result);
      log("HelloWorld Result: "+ result);                          Logging
    }
    catch (Exception ex) {                                         Exception
      ex.printStackTrace();                                        Handling
    }
  }

  private static Stub createProxy() {
    return (Stub) (new MyHelloService_Impl().getHelloIFPort());
  }

  private static void log(String s) {
    System.out.println(s);
  }
}
```

Code fragment 1 – Invoking a Web Service in Java

Another issue involves the variety of client-side management concerns that might be applicable on a specific service. In the code fragment, a simple form of authentication is used, but possibly the communication with the service needs to be encrypted, or the service needs to be paid in advance. All these concerns, enforced by the web service, will be reflected in the code of the client. For example: if the service is configured to only process messages with a specific SOAP header containing authentication information, then the client is forced to include this information in all SOAP messages it sends to the service. And as the services belong to different domain controllers, these requirements might change

independently on a frequent basis, even without notice. SOAP actually provides a protocol evolution model based on SOAP headers, exactly for this purpose. Therefore, the client is obliged to co-evolve with the service, even while loose-coupling is one of the key features of web service technology.

Furthermore, also the client might want to enforce a set of client-side concerns to guide the service invocation process. For example, to avoid expensive calls over the network the client might deploy a caching mechanism that returns cached results. Or, to avoid long waiting times, run a pre-fetching mechanism to invoke the service even before the client has made a specific request. Again, these concerns might require changes on various places in the code. Even if these concerns were encapsulated in a separate module, the places where these concerns are triggered and the manner to trigger them, are still spread and duplicated over the client application. For instance, in Java JAX-RPC [4], it is possible to specify dedicated message handlers for a given concern. However, these handlers are limited to adding, reading and manipulating header blocks of the SOAP messages sent to and received from service. Furthermore, they still need to be registered manually in a handler registry of the stub. This seriously hampers evolution of these management concerns.

By opting for an AOP-approach, each of the aforementioned concerns can be cleanly modularized in aspects and enforced in the code in an oblivious manner enhancing the evolution and maintenance of the code. AOP provides an expressive language to select joinpoints, that may identify any kind of method call or execution, even within the client application and the WSML framework, which is not the possible with message handlers. And, unlike with handlers, the full client context is available to aspect advices (e.g. part of the history when using stateful aspects). Furthermore, by opting for a dynamic AOP-approach it becomes possible to anticipate changes in the network and service environment without having to stop and alter the code of the client application. With dynamic AOP, the aspects can be plugged in and out at runtime, and as such enforce various selection policies and management concerns in the client. This is particularly important in critical applications dealing with long-running intra- or inter-organizational processes that cannot be stopped easily.

The prototype of the WSML is implemented in JAsCo, a novel aspect-oriented programming language targeted at component-based software engineering [3]. The main features of the JAsCo language are its highly reusable aspects and its strong aspectual composition mechanism for managing combinations of aspects. Clearly, the advantages of having reusable aspects in a component-based context also apply to the distributed service-based context. The JAsCo technology excels at providing dynamic integration and removal of aspects with a minimal performance overhead. The JAsCo language is an aspect-oriented extension of Java that stays as close as possible to the original Java syntax and concepts and introduces two important additional entities: *aspect beans* and *connectors*. An aspect bean is an extended version of a regular Java bean component that specifies crosscutting behaviour in a reusable manner. A JAsCo connector is responsible for applying the crosscutting behaviour of the aspect beans in a specific context and for declaring how several of these aspects collaborate.

4. Architecture of the WSML

Figure 1 illustrates the architecture of the WSML. JAsCo aspects are used to implement the generic functionality of the management layer while connectors specify when these aspects need to be deployed. The left-hand side of the figure illustrates an application requesting web service functionality via a Service Type. A service type is a generic description of the required service functionality, independently of concrete web services. A service type can be seen as a contract specified by the application towards the services: the client assumes a specific functionality of the service type, and the WSML is responsible to deliver that functionality.

The right-hand side shows three semantically equivalent services that are available to answer the request. By semantically equivalent services we identify services that offer the same functionality but might differ in the way they provide it. A mechanism based on *composition* and *redirection aspects* allows for the redirection of requests and enables hot-swapping. Additional selection policies, encapsulated in *selection aspects* enable advanced service selection. Finally, *management aspects* deal with management concerns such as monitoring, caching and billing.

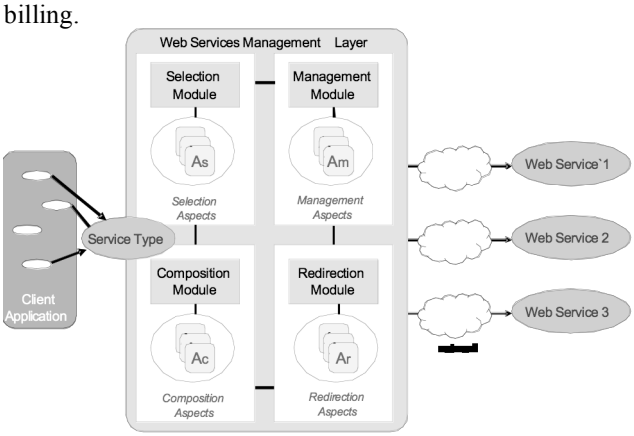

Figure 1 – Main Architecture of the WSML

4.1 Service Redirection Aspects

The first category of aspects in the WSML includes redirection aspects. A redirection aspect encapsulates all *communication details* of a single web service or of a service composition. In case of a single service it contains all necessary code to actually make the service invocation (in code fragment 1 all code annotated with the redirection label). Note that this code can become more complex in case of compositional mismatches: i.e. if glue-code is required to deal with mismatches in the service interface (examples include differences in method names, parameter types, parameter ordering, return types, etc).

More additional code is required to deal with possible semantical differences. For service compositions, the advices will contain calls to multiple services and specify the data- and workflow between the individual web services. Using redirection aspects realizes three important requirements for the WSML:

- **Hot Swapping:** if a service invocation fails, another semantically equivalent service is invoked by triggering another redirection aspect. This process is completely transparent for the client.
- **Just-in-time integration:** when a new service becomes available on the market that better suits the needs of the client, it can be easily integrated by adding a new redirection aspect to the appropriate joinpoints.
- **Pro-active selection:** by temporarily removing aspects by enabling or disabling the appropriate connectors, or by reordering the connectors, the redirection process can be optimized to prioritize one service over another and to accommodate to changes in the service and network environment.

Several forms of web service communication exist, and therefore several categories of redirection aspects have been identified. Each category requires additional specific AOP techniques:

Basic redirection

In case of stateless communication, it does not matter which service or service composition is addressed for a given request. Therefore, any of the available redirection aspects may be triggered at a given time. This category covers exactly the mechanism as stated above.

Conditional redirection

In specific scenario's not all available web services can deal with all kinds of requests: each service is specialized in a sub set of the possible requests. For this purpose, a condition can be added to the redirection aspects' pointcut specification using an *if* pointcut designator. As such, web services are filtered out at two levels: first, services are pro-actively filtered out by enabling or disabling connectors, for instance if the corresponding service is unavailable. Second, the condition of the remaining services is checked against the current request. Only the services that are left over are candidates to be invoked, as only their corresponding aspects are considered.

Stateful redirection

Stateful services keep state of their clients whenever a more complex interaction pattern is required. For example the client needs to successively login, browse for products, make a reservation, checkout his cart, do a payment and logout. In this case, the redirection mechanism cannot redirect successive requests to different services (unless the state migrates between the services, an option that is not further discussed here). Stateful aspects [5] provide a solution here. Stateful aspects are aspects that define a composite pointcut expression based upon a protocol history or execution trace. JAsCo natively supports stateful aspects by declaratively specifying a protocol pointcut based on a deterministic finite automaton [6]. By specifying the communication protocol of the web service as a pointcut of a stateful aspect, the advices (and thus the invocations of the web service) will only be triggered when the correct protocol is followed. As such, this strategy makes sure that the same web service or service composition is used consistently during a complete communication protocol. Using dedicated keywords, JAsCo allows to instantiate the redirection aspects for each instance of the client, in case multiple instances of the process need to run.

Compositional redirection

In case no web service can deliver the needed functionality for a service type, a composition of multiple services that work together can be specified in a dedicated aspect. By modularizing composition details in aspects, pro-active and reactive compositions are deployed. A pro-active composition uses a fixed set of concrete services, while in a reactive composition the services are not determined before hand and no concrete service interfaces are hardwired in the composition specification. Reactive compositions are used to avoid the explosion of the number of service compositions that need to be specified in case multiple partners are available to play a specific role in a composition. Only at runtime a temporal composition is created that best fits the criteria of the client by combining the appropriate redirection aspects.

4.2 Service Selection Aspects

With a powerful service redirection mechanism in place, the need arises to specify selection policies that guide the process of determining the most appropriate service for a given request. With the appearance of loosely coupled web service technology, selection becomes more important as the whole integration and communication process becomes more volatile. Today, selection might be based on the fact that all services must belong to a specific business partner, but tomorrow all services need to offer a specific Service Level Agreement (SLA) in order to become eligible.

A service policy specifies a **constraint** that should be met by the redirection mechanism. A policy can specify a hard constraint on an individual service (i.e. an imperative), or might specify a soft constraint on multiple services (i.e. a guideline). Some constraints can be enforced at any given moment while other ones only over a period of time. Out of the constraint specification two elements can be deduced:

- **Triggers:** the policy must be triggered whenever a change occurs in the environment, affecting the enforcement of the constraint. These triggering points can be located in various places ranging from the client, over the network to the actual web services.
- **Action:** the action of a selection policy typically includes qualifying, disqualifying and prioritizing services for the current or future client requests.

In the WSML, selection policies are represented by aspects: one aspect enforces one policy. The triggers are mapped to pointcuts and the actions to advices. For example, the aspect implementing a policy stating a maximum allowed price for a service will be triggered as soon as the price of that service changes. In the aspect advice the service will be disqualified if necessary. In case of a more complex policy stating the fastest service should be used, based on data monitored over the last month, the aspect needs to collect data on the response times of all services and reorder the services according to their speed. Using aspects to implement the selection policies, realizes the following requirements:

- **Identity:** each policy is modularized into one logical unit even while the policy might need data from various places in order to be able to execute. The policy is not scattered around multiple points in the code, making it easier to implement and maintain the policies.
- **Flexibility:** a wide range of unanticipated policies can be enforced in a unified manner without having to stop the client or rewrite any code.
- **Reusability:** many policies can be generalised in generic patterns: for instance "whenever a property changes, the policy should decide on disqualifying the service". By implementing this behaviour in reusable aspects, a library of reusable aspects is created.

The following categories of selection aspects exist. Each category exists in two flavours: imperative and guideline.

- **Client-initiated selection:** the constraint applies to explicit or implicit client-side business logic (e.g.: if the user of the client application has a gold subscription, use the fastest service). The triggering points reside in the client.
- **Non-functional service property based selection:** the constraint applies to one or more properties of one or more services that are advertised in the documentation (e.g. price). The location of the triggering points depends on the kind of property, the documentation and the notification mechanism used by the service.
- **Service behaviour-based selection:** the constraint is based on the behaviour of one or more services over a period of time (e.g.: the down-time of a service in the last month). The triggering points depend on the kind of property: typically a set of measurement points to collect the required data is necessary.
- **Service-initiated selection:** the constraint applies to explicit or implicit service-side business logic (e.g.: during peak hours the capacity of the service is limited to a certain number of requests for each client). Again, the triggering point depends on the documentation and notification mechanism used by the service.

In case of remote triggering points, a distributed joinpoint model [7] can be applied. However, this is only realistic in case both hosts belong to the same organisation. Another solution is to use a notification mechanism such as WS-eventing or a polling mechanism to detect remote changes and trigger advices. Also note that the last three categories of selection aspects may require detailed service documentation. The WSDL documentation of web services does not suffice for this purpose as no non-functional properties can be expressed in this format. However, as publicly available service descriptions are essential for achieving interoperability between heterogeneous systems, more research and standardisation efforts are needed to resolve this issue [8].

4.3 Service Management Aspects

Integrating unanticipated services from independent domain controllers causes another important issue: each domain might enforce a set of requirements on the service clients and the client must comply with them in order to invoke the service properly. Examples include the authentication and exception handling code in Code fragment 1, encryption, billing, reliable messaging, transactions, etc. A modular approach is required to implement and enforce these concerns in the client. On the other hand, also the client business logic might force additional concerns to be enabled. Examples include the logging in Code fragment 1, monitoring, caching, pre-fetching, etc.

Again, dynamic aspects are ideally suited for this purpose: each concern is modularized in a separate aspect, and deployed for those services that require it. Using aspects to implement the management concerns realizes analogue requirements of the WSML as the selection policy aspects: each concern is cleanly modularized in one aspect, non-anticipated concerns can be implemented in aspects and enforced in an oblivious manner in the client, and code reusability is achieved by generalizing the concerns in patterns. An aspect library is available for a wide range of concerns, and based on a set of parameters needed to instantiate the aspect, a connector can be automatically generated and compiled in the system.

Depending on the way they are deployed, the management aspects are enforced on three possible levels. By deploying the aspect *per service type*, they are enforced for each service composition and web service used to fulfil the functionality of the service type. *Per composition* results in an enforcement of the concern for all services belonging to that composition, and the most fine-grained deployment, *per web service,* only deploys the concern for one specific web service. For instance, a reusable caching aspect deployed per web service, realizes local caching (only the results of that service are cached), while deploying the caching per service type realizes global caching, as all the results returned by the service type are cached. Additional triggering points might reside in the client, the network or the services. To be able to detect changes in the web service, a polling mechanism or notification mechanism based for instance on WS-notification can be employed.

A common issue in current practice AOP consists of being able to manage the cooperation of several aspects applicable to the same joinpoint. Several approaches have

been proposed in order to make the composition of aspects more explicit, examples are Strategic Programming combinators [9] and treating aspect composition as function composition [10]. JAsCo supports a programmatic approach for explicitly representing aspect compositions, named *combination strategies*. In case of the WSML, this is especially important for the management aspects, which are not always completely orthogonal. Consider for instance the combination of both a caching and billing aspect. Whenever the caching aspect successfully retrieves the result for a request from the cache, which means that the actual web service is not invoked, the billing aspect should not be executed. This kind of composition policies can be explicitly captured by combination strategies.

5. Case Study

The research presented in this paper has been carried out in cooperation with Alcatel Bell in the context of broadband service provisioning. Implementing broadband services requires a plethora of different service capabilities, such as profiling, accounting, rating and network access. However, the current situation in the use of broadband internet shows that service capabilities are implemented from scratch by each service provider, increasing the effort of developing service applications. Each service provider uses their own systems and standards, making it difficult for network providers to accommodate to the different approaches employed by each service provider. This places a heavy burden on the network providers since they need to provide enough infrastructures to be able to integrate with all these different systems. As a consequence, there is a need and a market for a service and network management framework that facilitates the adoption of service capabilities. The Service Enabling Platform (SEP) of Alcatel Bell is a service provisioning platform targeted to this market. The WSML has been integrated in a prototype of the SEP to facilitate the integration with different content providers using web service technologies and dynamic AOP.

Several demonstrators have been developed that successfully exploit the WSML's capabilities. One of them uses the WSML to intercept messages between the SEP and proprietary Video-on-Demand (VoD) systems. The SEP and the WSML are hosted at the network provider and the VoD systems at different content providers. The SEP is a client of the WSML, the VoD systems offer web services for requesting, streaming and paying videos. End users can play streaming media on a television connected to a setup box and need to prepay the product with their mobile phone account. In this complex distributed setup with multiple partners, billing and accounting becomes a difficult task. The WSML needs to intercept messages between the SEP and the VoD and apply billing depending on the subscription status of the client (bronze, silver, gold), the product bought, the content provider of the VoD and the mobile phone operator. Typically, each partner receives a percentage or a fixed amount of the price paid by the customer. Furthermore, temporal promotions need to be applied on the billing, including reductions, free purchases, reductions on other products, etc. The redirection aspects of the WSML are used to intercept and redirect calls to the appropriate VoD systems and mobile phone operators. The payment schemas and promotional offers are implemented through a set of management aspects. In a typical setup with 3 VoD systems, 2 mobile operators, 5 billing schemas and 3 promotional actions, around 25 aspects (including 5 redirection aspects, 5 selection aspects and 15 management aspects) were deployed, containing around 2500 lines of aspect code. This is in contrast to the original implementation of this SEP demonstrator, which contains more than 9000 lines of crosscutting service invocation, selection and management code while only offering a fraction of the WSML's functionality.

6. Related Work

A lot of research is going on in the web service context and numerous vendors are currently working on dedicated web service management platforms. However, most of these approaches focus on the server-side management of web services. Our approach provides support for the client applications that want to integrate and manage different third-party web services.

From an industry point of view, WS-BPEL [11] is proposed as a functional approach for service composition. WS-BPEL allows for the specification of the partner roles and the logical flow of the messages in a composition. WS-BPEL and the WSML are complementary, as WS-BPEL is a service orchestration language, while the WSML offers a dynamic service invocation mechanism for clients. On a technical level, the WSML and a WS-BPEL-engine can be integrated by making service types fill in the partner roles in the BPEL-process, instead of concrete web services. Or vice versa: by using an as web service exposed BPEL-process to fulfill the functionality of a service type.

Semantic Web also offers an approach for composition based on ontologies. OWL-S [12] is a web service ontology, which offers a core set of markup language constructs for describing the properties and capabilities of services in a unambiguous, computer-interpretable form. In the WSML, a Matchmaker algorithm has been developed to determine the compatibility between service types and web services when they are both enriched with ontological documentation.

The idea of applying AOP concepts at the client-side to decouple web services concerns is quite innovative; as a result not many approaches have been proposed that focus on this combination. However, Arsanjani et al. [13] have identified the suitability of AOP to modularize the heterogeneous concerns involved in web services. More recently, AO4BPEL [14] has been proposed as an AOP extension for WS-BPEL. Aspect-Sensitive Services (CASS) are proposed in [15] as a distributed aspect platform that targets the encapsulation of coordination,

activity lifecycle and context propagation concerns in service-oriented environments.

7. Summary

This article discusses the different kinds of aspects and AOP techniques used to implement the WSML. The WSML is an AOP-enabled reusable framework for the dynamic integration, selection, composition and client-side management of web services in client-applications. By employing AOP, the client application becomes oblivious of web service related concerns. Because these concerns are now well modularized, flexible and dynamic integration of the web service invocation, composition and management is realized.

8. References

[1] Verheecke, B., Cibrán, M. A., Vanderperren, W., Suvee, D., Jonckers, V., "AOP for Dynamic Configuration and Management of Web services in Client-Applications", *International Journal on Web Services Research (JWSR),* Volume 1, Issue 3, July-Sept 2004.

[2] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J., "Aspect-oriented programming", *In Aksit, M., Matsuoka, S. (eds.): 11th Europeen Conf. Object-Oriented Programming*. Volume 1241 of LNCS., Springer Verlag, 1997.

[3] Suvée, D., Vanderperren, W. "JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development," *proceedings of 2nd Int. Conference on Aspect-Oriented Software Development*, Boston, MA (USA), March 2003.

[4] Loughran, S., Smith, E., "Rethinking the Java SOAP Stack", *to be published in proceedings of IEEE International Conference on Web Services (ICWS) 2005*, , Orlando, FLA (USA), July 2005

[5] Douence, R., Fradet, P., S¨udholt, M., "Composition, reuse and interaction analysis of stateful aspects", *In Lieberherr, K., (ed): proceedings of 3rd International conference on Aspect Oriented Software Development 2004 (AOSD)*, ACM Press, Lancaster (UK), March 2004.

[6] Vanderperren, W., Suvee, D., Cibran, M., De Fraine, B, "Stateful Aspects in JAsCo", *proceedings of Software Composition 2005*, LNCS, Edinburgh (UK), April 2005.

[7] Nishizawa, M., Chiba, S., and Tatsubori, M., "Remote pointcut: a language construct for distributed AOP." In Proceedings of the 3rd international Conference on Aspect-Oriented Software Development (AOSD) '04. ACM Press, Lancaster (UK), March 2004

[8] O'Sullivan, J., Edmond, D., Hofstede, A, "What's in a service: Towards accurate description of non-functional service properties." *Distributed and Parallel Databases Journal*. Special Issue on E-Services 12 (2002), pp. 117-133.

[9] Lämmel, R., Visser, E., Visser, J, "Strategic Programming Meets Adaptive Programming", *proceedings of the second International Conference on Aspect-Oriented Software Development,* Boston (USA), March 2003.

[10] Lopez-Herrejon, R., Batory, D., "Improving Incremental Development in Aspectj by Bounding Quantification", *proceedings of Software Engineering Properties and Languages for Aspect Technologies (SPLAT)*, AOSD conference, Chicago, Ill (USA), March 2005.

[11] Andrews, T., et al. "Business Process Execution Language for Web Services (BPEL4WS)", *specification 1.1*, available at: http://www.siebel.com/bpel.

[12] Ankolekar, A. DAML-S: "Web Service Description for the Semantic Web", p*roceedings of the International Semantic Web Conference,* Sardinia (Italia), June 2002.

[13] Arsanjani, A., Hailpern, B., Martin, J., Tarr, P., "Web Services Promises and Compromises," *ACM Queue*. March. 1(1). 2003, http://www.acmqueue.org/

[14] Charfi, A., Mezini, M., "Apect-Oriented Web Service Composition with AO4BPEL", *proceedings of the European Conference on Web Services 2004 (ECOWS'04)*, Erfurt (Germany), Sept. 2004,

[15] Cottenier, T., Elrad, T., "Dynamic and Decentralized Service Composition*", to be published in proceedings of Web Information Systems and Technologies,* Miami, FLA (USA), May 2005.