AOP for Dynamic Configuration and Management of Web Services

Bart Verheecke, María Agustina Cibrán

System and Software Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Europe

{Bart.Verheecke; Maria.Cibran}@vub.ac.be

Abstract. Web service technologies accelerate application development by allowing the selection and integration of third-party web services, achieving high modularity, flexibility and configurability. However, current approaches only allow this integration by hard wiring the references to concrete web services into the client applications. Moreover they do not provide any management support, which is fundamental for achieving robustness. We observe the need for the application to be independent of specific services and present the WSML, a management layer placed in between the application and the world of web services. In this paper we identify the requirements for this layer to realise the dynamic selection and integration of services, client-side management of services, and support for rules that govern the selection, integration and composition. We show how dynamic AOP is ideally suited to implement the core functionality of the WSML using the JAsCo dynamic aspect-oriented language to conduct the experiments.

1. Introduction

Web services are modular applications that are described, published, localised and invoked over a network. In the relative short time that web services have been around, an impressive range of supporting tools has been developed that enable the creation and deployment of web services and the development of service oriented applications. Large platforms such as Java ONE and Microsoft.NET provide the key technologies, build around W3C standards such as SOAP [1], WSDL [2] and UDDI [3], and allow publishing, looking up and consuming services in a straightforward manner.

However, the approaches typically used to integrate services in client applications are very static and do not allow any dynamic adaptations of the web services themselves or the way they are used. As stated in [4], this leads to unmanageable applications that cannot adapt to changes in the business environment (e.g. a service that is abandoned or changed, a new service that becomes available on the market, etc).

A second obstacle is that by generating hard-wired proxy-classes, and as such treating the services as regular software components, the specific requirements of services are completely ignored. Services are organisationally fragmentized, can be asynchronous and latent, can become unavailable due to unpredictable network conditions and thus require more overall management [5]. To deal with these issues in order to create more robust applications, code has to be written manually and repeated for each service. This way, the code is duplicated, scattered all over the application and becomes an obstacle for future maintenance.

A third limitation encountered is that services can only be selected based on the functionality they offer. The web service documentation provided in WSDL-format does not support the explicit specification of non functional requirements such as constraints based on Quality-of-Service and management statements, classes of service, access rights, pricing information, SLA's (Service Level Agreements) and other contracts between web services. Explicitly specifying these non-functional requirements at the service side in a precise and unambiguous way would allow the composition and integration of services to occur in a more intelligent and customized manner. This way, applications are able to specify criteria to be considered at service selection time to integrate those services that best fit the application requirements.

In this paper we present a management layer called **Web Services Management Layer (WSML)**, which is placed in between the application and the world of web services. This intermediate layer allows: dynamic selection and integration of services into an application, client-side management of the service, and support for rules that govern the selection, integration and composition.

The focus of this paper is to identify the requirements for a technological platform to realise the WSML. We show how dynamic *Aspect Oriented Programming* (AOP) [6] [7] is ideally suited to build the core functionality of this management layer. To deal with the dynamic nature of the service environment we suggest the use of a dynamic aspect-oriented programming language called JAsCo [8] [9]. In the next section we identify the requirements pursued for the WSML and analyse how state-of-the-art approaches fail in achieving them. In section 3 we explain the basic architecture of our solution and the objectives achieved. A following section describes the need for AOP and gives an overview of the characteristics of JAsCo. In section 5 we show how JAsCo is ideal to implement the core functionality of the WSML and provide code examples. Finally, we present related and future work in section 6 and conclude in section 7.

2. Requirements for WSML

Web services constitute a promising middleware technology. However, we believe there are some important management issues that still need to be addressed. Today, the technology only provides standardised connection mechanisms, but this might not be sufficient to drive its world wide adoption. We identify the following challenges that have to be taken into account:

1. **Composition of Web services:** the real strength of web services lies in the concept of combining and orchestrating them in order to deliver added-value services. However, this leads to the question of how to dynamically (i.e. at runtime) integrate a service in the consuming application. How can the required service be found and plugged in? How can one deal with different services delivering similar functionality? How can compositions of services be made that were not anticipated at design time of the application?
2. **Multi-Partner Processes:** consuming web services means that processes of several business partners situated on different locations are being integrated with each other. This raises significant management issues (e.g. organisational fragmentation of the application) and technical consequences (e.g. much longer processing times, security issues, AAA, etc).
3. **Dependencies:** an application that integrates with external services becomes dependent on them. Failure of the service will have a negative impact on the execution of the application.
4. **Flexibility:** since the context is an ever-changing business environment, the application must be flexible enough to deal with changes in the functionality, displacement or use of web services.

If we look at the current approaches of how web services are integrated in applications, we see that these issues are not addressed at all, or if so only partly:

In the **Wrapper Approach** each web service is wrapped and treated as an internal software component (see figure 1). State-of-the-art tools like MS Visual Studio.NET and BEA WebLogic adopt this mechanism [10]. The software developer enters a web reference to the service into the tool. Then, the WSDL document of the service is automatically looked up, analysed and code for a proxy class is generated. Programmers can invoke web methods on this class just like on a regular class and do not have to take into account that they are actually dealing with remote procedure calls (RPCs). Clearly, the issues mentioned above are not addressed: (1) services are selected and integrated in a static manner at design time, (2) code that deals with managing multi-partner processes has to be written manually, needs to be duplicated for each service and becomes scattered in different modules of the application code, (3) the application has to deal with the potential unavailability of services by manually including code that again results tangled and scattered (4) changes in the environment can only be dealt with by stopping and rewriting the application code.

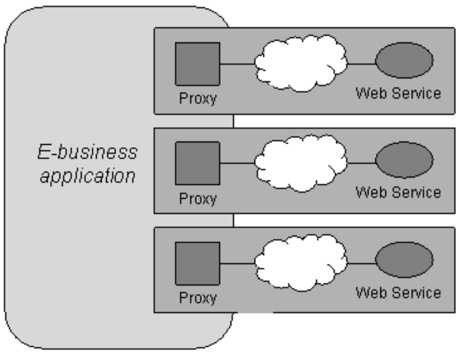

Fig. 1. The Wrapper Approach to integrate services

A more dynamic solution is provided when using a **tModel** [11] instead of a concrete service. In this approach, a service can be invoked to deliver the required functionality if it follows the overall system's tModel specification. However, this only solves some of the identified issues: (1) only services that exactly implement the tModel can be dynamically integrated, while services with other interfaces and service compositions that deliver the same functionality cannot be straightforwardly included, (2) management code still needs to be written manually and remains scattered over the application code. Problem (3) is addressed as the dependency on a certain service is weakened because dynamic switching to other similar services can be achieved in a straightforward manner; however this is only tackled to some degree since that code needs to be written manually. Finally, (4) is addressed as services that match a tModel can be registered at runtime in a UDDI-register [3] and can as such be included in the application without having to make any changes.

We propose an abstraction layer, called **Web Services Management Layer (WSML)**, which is placed between the application and the world of web services and deals with the identified requirements. We introduce the basic architecture and its objectives in the next section.

3. Design of WSML

The WSML allows decoupling web services from the core application. It realises the concept of *just-in-time integration of services*: multiple services or compositions of services can be used to provide the same functionality. The advantages of this approach are:

- The application becomes more flexible as it can continuously adapt to the changing business environment and communicate with new services that were not known or available at design time.

- Extracting all web service related code from the core application facilitates future maintenance of the code.
- By weakening the link between the application and the service, hot swapping functionality can be installed. This mechanism enables switching to other services when they become unreachable due to network conditions or service-related problems.
- The hot-swapping mechanism can take into account specific application requirements. This way, rules specifying criteria based on non-functional requirements of services can be considered as part of the layer and customized for each application, making the selection of services more intelligent.

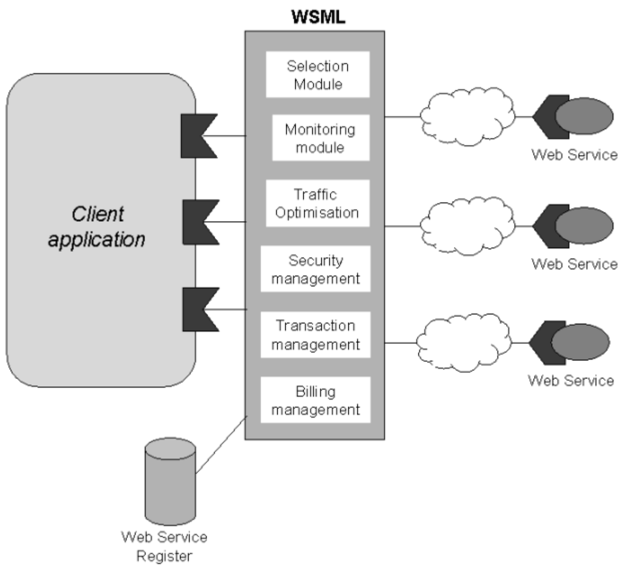

Fig. 2. General Architecture of WSML

Figure 2 shows the WSML as a layer between the application and the web services. On the left side the core application resides, which requests service functionality when needed. The WSML is responsible for intercepting these requests and choosing the most appropriate service or composition. This is realised by the **Selection Module** by considering different service properties. To this end, collaboration with the **Monitoring Module** is required as several properties of services might need observation over time. Available services are looked up in a central UDDI-register [3] or discovered using decentralised Web Services Inspection Language (WSIL) documents [12].

Generic functionality, necessary to provide support for multi-partner processes (e.g. to guarantee security, transactions, etc) also resides in the layer, depending on specific application requirements. Examples of this generic management functionality are:

- **Traffic Optimisation:** in order to reduce traffic over the network to avoid slowing response times, losing packages, congesting networks, etc., a caching mechanism can be considered. Instead of invoking the services each time the application requests certain service functionality, the results are retrieved from a cache.
- **Billing:** services can specify their own billing strategies and the application might want to control how billing is applied. Analogously, services might need to be billed for being integrated in the client applications.
- **Accounting:** many industries are required to provide tracing and logging capabilities for accounting as well as regulatory purposes.

- **Security:** web services might have several security mechanisms (e.g. WS-Security [24]) depending on the environment where they are deployed. Data sent over the network can be encrypted and user authentication and authorisation might be required.
- **Transaction:** if the communication with a web service is transaction based and if in the middle of a transaction the layer must switch to another service, some compensation or roll-back mechanism might be used to restore the state of the previous service.

In section 5 an AOP solution for the encapsulation of these concerns as part of the WSML is presented. It is important to note that the WSML is reusable in different applications and is fully configurable to avoid unnecessary overhead. In the following sections we present the technical decisions made for the implementation of the layer.

4. AOP for WSML

4.1 Motivation for AOP

To achieve high flexibility in the selection of services, hard wiring references to concrete services in the applications must be avoided. Using traditional software engineering methodologies it would be the responsibility of the applications to decide which the most suitable services are for a given request or to explicitly request the layer to do that. In both cases, code for either implementing or triggering the service selection would be written at each point where some service functionality is required. As a consequence selection code would also result scattered in different places in the application and tangled with code that solves other concerns. Thus, we need support for encapsulating this crosscutting code separated from the application and plug it in and out in a non-invasive way. To achieve this we need to intercept the points in the execution of the application where concrete services are invoked and make the layer decide which the most suitable services for those requests are.

Moreover the selection of services also involves other management issues to be considered at the moment the services are selected to be integrated in the applications. For instance, services might need to control security, accounting, billing concerns at the time their functionality is requested. This also results in crosscutting code since the application developer would need to include this management code each time a service is requested.

Thus, to avoid tangling the application code with service related code we believe *Aspect Oriented Programming* (AOP) is needed. AOP argues that some concerns of a system, such as synchronisation and logging, cannot be cleanly modularized using current software engineering methodologies as they are scattered over different modules of the system. Due to this code duplication, it becomes very hard to add, edit and remove such a *crosscutting* aspect in the system. The ultimate goal of AOP is to achieve a better separation of concerns. To this end, AOP approaches introduce a new concept to modularize crosscutting concerns, called an *aspect*. An aspect defines a set of *join points* in the target application where the normal execution is altered. Aspect *weavers* are used to weave the aspect logic into the target application.

Using aspects to express the selection and management concerns as part of the WSML allows the application to remain independent of the service selection infrastructure. Moreover, we also pursue dynamism in the management of services and therefore an AOP technology that provides support for dynamic inclusion and removal of aspects is required.

To this end, we introduce in the next section an aspect-oriented implementation language called JAsCo [8] [9] whose features are ideal for achieving the identified requirements.

4.2 JAsCo

JAsCo is primarily based upon two existing AOP approaches: *AspectJ* [13] and *Aspectual Components* [14]. AspectJ's main advantage is the expressiveness of its language to describe the join points. However, AspectJ's aspects are not reusable, since the context needed by an aspect to be deployed, is specified directly in the aspect definition. To overcome this problem, Karl Lieberherr et al. introduce the concept of Aspectual Components. They claim that when using AOP one must be able to express each aspect separately, in terms of its own modular structure. Using this model, an aspect is described as a set of abstract join points which are resolved when an aspect is combined with the base modules of a software system. This way, the aspect behaviour is kept separate from the base components, even at run time.

JAsCo combines the expressive power of AspectJ with the aspect independency idea of Aspectual Components. The advantages of JAsCo are:

- Aspects are described independent of a concrete context, making them highly reusable.
- JAsCo allows easy application and removal of aspects at run time.
- JAsCo has extensive support for specifying aspect combinations.

The JAsCo language itself stays as close as possible to the regular Java syntax and it introduces two new concepts that are highly valuable in the context of this research:

- **Aspect Beans:** an aspect bean is an extension of the Java Bean component that specifies crosscutting behaviour in a reusable manner. It can hold hook definitions, which specify ***when*** the normal execution of a method should be intercepted and ***what*** extra behaviour should be executed.
- **Connectors:** a JAsCo connector is responsible for applying the crosscutting behaviour of the aspect beans and for declaring how several of these aspects collaborate. It specifies ***where*** the crosscutting behaviour should be deployed.

On a technical level a new, backward compatible component model is introduced that enables the run-time application and removal of connectors. This high flexibility and configurability is exactly what is needed in the context of web services. JAsCo's connectors are ideal to achieve the dynamic behaviour of the WSML and to allow the runtime integration and management of services without invasively having to change the core application. Section 5 shows code examples of how JAsCo connectors and aspect beans are used. For more information about JAsCo and the JAsCo component model, we refer to [8] [9].

5. Implementation of WSML

5.1 Architecture of WSML

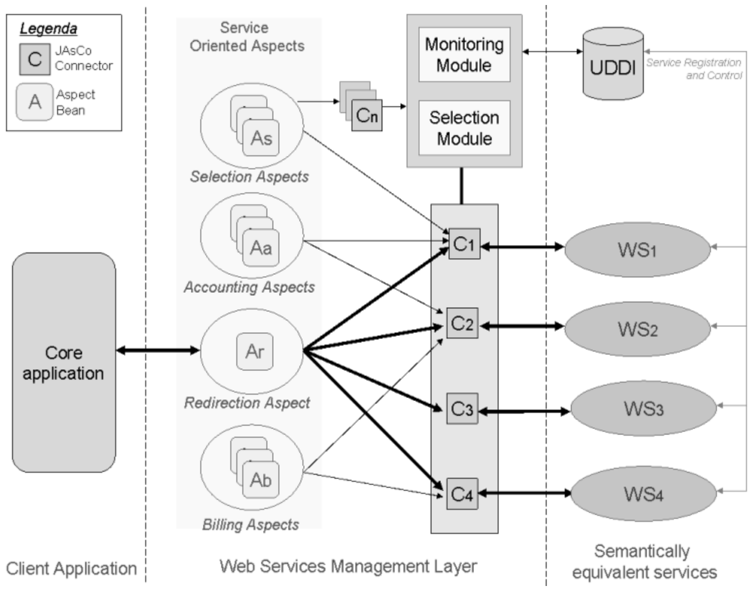

Fig. 3. Detailed Architecture of the WSML

Figure 3 shows the implementation of the WSML using aspect beans for the generic functionality of the layer and dynamic JAsCo connectors to specify when they need to be deployed. The left part of figure 3 illustrates the application requesting web service functionality. The right part shows four semantically equivalent services that are available to answer the request.

To enable the application to make requests without referencing concrete services the concept of *Abstract Service Interfaces* is introduced in section 5.2. The mechanism based on redirection aspects that allow requests redirection and hot-swapping is discussed in section 5.3. Additional management aspects to deal with concerns such as accounting and billing are described in section 5.4. Finally, in section 5.5 the selection and monitoring modules of the WSML responsible for enabling intelligent service selections are discussed.

5.2 Abstract Service Interfaces (ASI)

A basic requirement as identified in section 2 is that hard-wiring of services should be avoided. Therefore, service requests must be formulated in an abstract way at the left side of the layer and the WSML will be responsible for making the translation to a concrete service at the right side. The requests of the application are formulated in an abstract way as specified in an **Abstract Service Interface (ASI)**. This can be seen as a contract specified by the application towards the services. This way the syntactical differences between semantically equivalent services can be hidden. After all, if the same functionality is implemented by two different parties, it will be done in a different manner. Two services may offer the same functionality but differ on a number of considerations like:

- Web method names
- Synchronous / Asynchronous methods
- Parameter types & return types
- Semantics of parameters & return values
- Method sequencing order

By introducing the WSML, we can hide the heterogeneity of those services. This approach differs from the concept of tModels [11] where the services must all have identical interfaces. Our solution is complementary because we also support services with different interfaces and even service compositions. In order to enable this we introduce the concept of mapping schemas to unambiguously describe how the service or service composition maps to the ASI. These schemas can be provided by the service owner or can be specified by the application developer. The idea of using sequence diagrams for expressing this mapping is followed here. This solution is also applied in PacoSuite [15] [16] [17], a component based development tool where sequence diagrams are used to determine if composition patterns and components are compatible and to automatically generate glue code to connect the components. In the same way, sequence diagrams can be used to specify the mapping between ASIs and concrete service interfaces.

To illustrate these ideas an example of a travel agency application is introduced. The application offers the functionality to book holidays online which customers can use to make reservations for both flights and hotels. To achieve this functionality this application integrates different web services. Suppose `HotelServiceA` and `HotelServiceB` are services that offer the same functionality for the online booking of hotels. Each hotel service returns exactly the same results.

Assume in the client-application a list of hotels has to be presented to the customer. Thus, to this end a `HotelServiceASI` is defined which specifies the following method:

```
HotelList giveAvailableHotels (Date, Date, CityCode)
```

At deployment or run time the following two services are available:
`HotelServiceA` provides the method: `giveHotels (CityCode, Date, Date)`
`HotelServiceB` provides the method: `listHotels (Date, Date, CityName)`

Figure 4 shows how the `HotelServiceASI` is mapped to the concrete interfaces provided by these services. In the first example, both the name and the order of the arguments of the ASI differ from the ones specified in `HotelServiceA`. In the second example a simple service composition is needed. Because `HotelServiceB` requires city names instead of city codes, a third service is required to perform the translation between city names and codes.

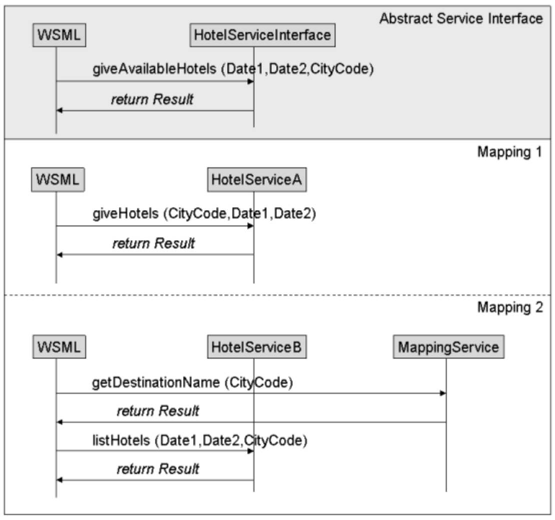
Abstract Service Interface
WSML
HotelServiceInterface
giveAvailableHotels (Date1,Date2,CityCode)
return Result
Mapping 1
WSML
HotelServiceA
giveHotels (CityCode,Date1,Date2)
return Result
Mapping 2
WSML
HotelServiceB
MappingService
getDestinationName (CityCode)
return Result
listHotels (Date1,Date2,CityCode)
return Result

Fig. 4. Example of Sequence diagram for Abstract Service Interface mapping

5.3 Redirection Aspect

In the previous section we introduced how ASIs can be used to make the application independent of concrete details about specific services by referring to them in a generic and uniform way. However, to make this abstraction realisable we need to provide means to map this generic description to concrete service invocations. To achieve this, we make use of the aspect power of JAsCo and define an aspect in charge of redirecting the generic requests to the concrete services that will provide the functionality required.

The redirection aspect defines the logic of intercepting the application requests and replacing them by a concrete invocation on a specific web service. Figure 5 shows the implementation of the redirection aspect. Note that this aspect is generic and does not refer to any concrete web service. The mapping to concrete web services is specified in the connectors that deploy the redirection aspect. Several connectors can exist each in charge of deploying the redirection to a concrete web service. Figure 6 illustrates the deployment of the redirection aspect. The connector `getAvailableHotelsOfServiceA` specifies the mapping between the ASI `giveAvailableHotels(Date, Date, CityCode)` and the particular way to invoke that functionality on the web service `HotelServiceA`, i.e., invoking the method `giveHotels(CityCode, Date, Date).` To actually communicate with `HotelServiceA` the GLUE library is used [18].

```
class getAvailableHotelsRedirection {
  hook RedirectionHook {
    RedirectionHook(method(Date d1, Date d2, CityCode cc)){
      call(method);
    }

  replace(){
    specificMethod(d1, d2, cc);
  }

abstract public List specificMethod(
                        Date d1, Date d2, CityCode cc);
  }
}
```

Fig. 5. The Redirection Aspect Bean for hotel retrieval

```
static connector getAvailableHotelsOfServiceA {
  HotelServiceAStub hotelServiceA = null;

  try {
    hotelServiceA = HotelServiceAHelper.bind();
    // the Stub is instantiated here by analysing the WSDL-file of
    // hotelServiceA by using the GLUE library
  }
  catch(Exception e)
  {
  }

  getAvailableHotelsRedirection.RedirectionHook rhook =
    new getAvailableHotelsRedirection.
      RedirectionHook(Application.giveAvailableHotels(
                                      Date, Date, CityCode){

        public List specificMethod(Date d1, Date d2, CityCode cc){
          return hotelServiceA.giveHotels(cc, d1, d2));
        }
      }
}
```

Fig. 6. Connector that deploys the redirection aspect on HotelServiceA

Each connector encapsulates the mapping between each generic request in the application and the concrete manner to solve that request in a specific service. Thus, there will be one connector for each different request performed by the application. The WSML is responsible for the creation and management of these connectors.

JAsCo allows the creation of connectors to be done dynamically. This characteristic enables the dynamic integration of new services. When the functionality of a new service has to be integrated in the application, a connector realizing the mapping for that service is created at run time. This is achieved transparently for the application. Note that if the description of the services is extended with the specification of the mapping to ASIs as described in Section 5.2, the creation of connectors can be done automatically in the WSML.

This mechanism enables to realise hot swapping of services. If the response time of a service is too slow or the service becomes unavailable the selection module can "hot swap" to another service by activating its connector and deactivating the previous one. Because JAsCo allows the runtime plugging in and out of connectors, this process is achieved in a completely dynamic way. Note that if the communication with the service involves more complex transactions or if the service is state full, a transaction mechanism is required, as mentioned in section 3.

5.4 Client-Side Service management

As mentioned earlier, the layer can also deal with other management issues that need to be controlled at the application side. For instance, suppose `HotelServiceA` describes a strategy for billing its use and the application wants to locally control this for auditing reasons. Imagine the service specifies that each time the method `giveHotels (CityCode, Date, Date)` is invoked, an amount of 5 euros has to be paid. If this

functionality is implemented as part of the application, code for dealing with the billing concern would be needed at each point in the application where a request to that service is done. Thus, this code would crosscut the core application and should then be decoupled and encapsulated in a separate module.
We can achieve this by defining a new aspect that abstracts the logic for a "*pay per use*" billing strategy. Figure 7 shows the implementation of this aspect. Note that this aspect is generic and can be deployed and customised for other services that adopt this billing policy. This deployment is specified as part of the connector shown in Figure 8. In this example, the billing is done when `getAvailableHotels` is invoked in the application. However, as the connector `getAvailableHotelsOfServiceA` implements this method as a call to `HotelServiceA`, the billing is only done when this concrete service is used. Note that the hook can also be initialised with multiple functionalities provided by a web service.

```
class BillingPerUse {
  hook BillingHook {

  private int total = 0;
  private int cost = 0;

  public void setCost(int aCost){
    cost = aCost;
  }

  private void pay(){
    total = total + cost;
  }

  BillingHook(method(Date d1,Date d2,CityCode cc)) {
    call(method);
  }

  after() {
    pay();
  }
 }
}
```

Fig. 7. Billing Aspect

```
static connector getAvailableHotelsOfServiceA {
  …
  BillingPerUse.BillingHook billPerUse =
    new BillingPerUse.BillingHook(List
      Application.giveAvailableHotels(Date, Date, CityCode));

  billPerUse.setCost(5);
  rhook.replace();
  billPerUse.after();
}
```

Fig. 8. Connector extended with Billing

The aspect `BillingPerUse` defines a billing template that can be reused by different services. Other more complex billing aspects can be formulated and implemented in a similar way. This simple example illustrates that a generic library of aspects can be created to achieve high flexibility in the creation and manipulation of aspects that implement several management issues.

5.5 Service Selection & Monitoring

In section 5.3 we discussed how ASIs can be mapped to concrete services and service compositions. This mechanism enables the layer to decide which semantically equivalent services will be effectively chosen when the application invokes certain functionality. This is achieved by means of dynamically selecting and activating the corresponding service connectors. The selection module is responsible for regulating the activation of connectors. A service will effectively deal with a given request if the selection module of the WSML selected and activated its connector.

In order to control the properties of the web services that influence the decision, monitoring is required. A solution using aspects, similar to the one introduced in section 5.4 can be applied to dynamically install the necessary measurement points. For instance, if the response time of the services is an important criterion, then a monitoring aspect can be plugged in to intercept the web service invocations and provide information on the response time to the selection module.

Note that to be able to specify non-functional properties on web services, an extension of WSDL is required. WSOL, the Web Service Offering Language [19] can be used for this purpose. This is however outside the scope of this paper and is the subject of future research.

6. Related work

We realize that a lot of research is going on in this context and that a lot of vendors are currently working on web service management platforms. However, most of these products focus on the service side management of web services. They allow developers to build and deploy web services and also provide some management capabilities such as load balancing, concurrency, monitoring, error handling, etc. On the contrary, our approach provides support for the client applications that want to integrate different third-party web services and manage them. Some of the most relevant approaches are: The Web Service Description Service (WSDF) [20] incorporates ideas from the Semantic Web community [21] suggesting an ontological approach for the side effect free invocations of services. The Web Services Mediator (WSM) [22] also identifies the need for a mediation layer to achieve dynamic integration of services. However, as far as we know there is no implementation available.
The Web Services Invocation Framework (WSIF) [23] supports a Java API for invoking web services irrespective of how and where the services are provided. WSIF mostly focuses on making the client unaware of service migrations and change of protocols. We are analyzing how we can incorporate WSIF capabilities into the WSML.

A lot of work is being done towards standardization of various web service protocols covering security (WS-Security [24], WS-Policy [25], etc), transaction support (WS-Transaction [26]), business process definition languages (WSFL, XLANG [27]), etc. However, they all require the web service clients to address the services respecting specific protocols. The approach presented in this paper allows respecting these protocols in the management layer without invading the application.

The idea of applying AOP ideas to web services is quite innovative and thus not many approaches have been developed focusing on this field. However, Arsanjani et al. [28] have recently identified the suitability of AOSD to modularize the heterogeneous concerns involved in web services. In particular they refer to technologies like Aspect/J [13] and the Hyper/J [29] [30] for Multi-dimension Separation of Concerns (MDSOC). However these approaches only allow static aspect weaving, contrary to JAsCo which supports dynamic pluggability of aspects. They also identify the need for the web services to include the definition of non-functional interfaces that permit control over performance, reliability, availability, metering and level of service to take into account at service selection time.

7. Conclusion

In this paper we propose a new management layer WSML to control the integration and configuration of web services in client application. We identified the need to use AOP as we are dealing with crosscutting concerns. We propose to use a dynamic AOP implementation language JAsCo to enable hot-swapping and runtime management of services.
This approach has the advantage that applications become more robust as the layer can dynamically deal with potential service failures and hot swap to new functionally equivalent services. Moreover, a high flexibility for the integration of services is achieved by supporting the "on the fly" pluggability of services.

We are currently working on the definition of a library of reusable aspects that would allow the application developer to dynamically instantiate and configure the needed aspects to deal with different service management concerns. These reusable aspects can be seen as generic templates that can be customised and integrated "on demand" to accommodate to service requirements.
We are also working on realising the hot swapping mechanism in a more intelligent way by considering service oriented rules. These rules are derived from the application requirements and are based on the non-functional properties of services. They govern the selection of services specifying under which conditions the services have to be selected.
Other direction of research under attention is the automatic generation of the connector code. The idea is that each time a service has to be considered the WSML generates the connector code that will allow the future integration of that service into the application. To achieve this, the mapping between the ASIs and the concrete service functionality has to be dynamically parsed and translated into code. The non-functional properties of services also have to be analysed to instantiate the necessary management aspects. Note that this has to be achieved dynamically, without stopping the client application.

8. References

[1] W3C, "Simple Object Access Protocol (SOAP) v1.2," Whitepaper, W3C Technical Publications, http://www.w3.org/TR/SOAP/

[2] W3C, "Web Service Description Language (WSDL) v1.2," Whitepaper, W3C Technical Publications, http://www.w3.org/TR/wsdl12/

[3] Uddi.org, "Universal Description, Discovery and Integration," UDDI Executive Whitepaper, November 2001

[4] J. Malhotra, Ph.D., Co-Founder & CEO interKeel Inc., "Challenges in Developing Web Services-based e-Business Applications," Whitepaper, interKeel Inc., 2001

[5] C. Szyperski, "Components and Web Services," Beyond Objects column, Software Development. Vol. 9, No. 8, Augustus 2001.

[6] Aspect-Oriented Software Development. http://www.aosd.net/

[7] Communications of the ACM. Aspect-Oriented Software Development, October 2001.

[8] D. Suvée and W. Vanderperren. "JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development". Proc. of 2nd Int. Conf. on AOSD, Boston, USA, 2003.

[9] W. Vanderperren, D. Suvée, B. Wydaeghe and V. Jonckers. "PacoSuite & JAsCo: A visual component composition environment with advanced aspect separation features". Proc. of Int. Conf. on FASE, Warshaw, Poland, April 2003.

[10] Microsoft, "XML Web services," Visual Studio.NET Technical Resources, http://msdn.microsoft.com/vstudio/techinfo/articles/xmlwebservices/default.asp

[11] Microsoft, "Introduction to UDDI Services and tModels," 2002, http://isb.oio.dk/uddi/help/en /intro.whatisuddi.aspx

[12] T. Appnel, "An Introduction to WSIL, the Web Services Inspection Language," O'Reilly on Java.com, 2002, http://www.onjava.com/pub/a/onjava/2002/10/16/wsil.html

[13] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersen, J. Palm, and W. G. Griswold. "An overview of AspectJ". In Proceedings European Conference on Object-Oriented Programming, volume 2072 of Lecture Notes in Computer Science, pages 327--353, Berlin, Heidelberg, and New York, 2001. Springer-Verlag.

[14] K. Lieberherr, D. Lorenz and M. Mezini, "Programming with Aspectual Components. Technical Report," NU-CCS-99-01, March 1999. Available at: http://www.ccs.neu.edu/research/demeter/biblio/aspectual-comps.html.

[15] B. Wydaeghe, and W. Vanderperren, "Visual Component Composition Using Composition Patterns," In Proceedings of Tools 2001, Santa Barbara, USA, July 2001

[16] W. Vanderperren, "Applying aspect-oriented programming ideas in a component based context: Composition Adapters," In Proceedings of NetObjectDays 2001, Erfurt, Germany, September 2001

[17] W. Vanderperren, B. Wydaeghe, "Separating concerns in a high-level component-based context,"EasyComp workshop @ ETAPS 2002, Grénoble, France, April 2002. Also published in Electronic Notes in Theoretical Computer Science, Vol 65(4), 2002.

[18] The Mind Electric, "The Glue Platform," 2003, http://www.themindelectric.com/glue/index.html

[19] V. Tosic, B. Pagurek, K. Patel, "WSOL – A Language for the Formal Specification of Classes of Service for Web Services". To be published in Proceedings of ICWS'03 (The First International Conference on Web Services), Las Vegas, USA, June 2003.

[20] A. Eberhart, "Towards Universal Web Service Clients," Proceedings of Euroweb 2002, Oxford, UK, December 2002

[21] "The Semantic Web Community Portal," http://www.semanticweb.org/, November 2002

[22] S. Chatterjee , "Developing Real World Web Services-based Applications", The Java Boutique, http://javaboutique.internet.com/articles/WSApplications/

[23] Apache, "The Web Services Invocation Framework (WSIF)," http://ws.apache.org/wsif/

[24] IBM, "Web Services Security (WS-Security) version 1.0," http://www-106.ibm.com/developerworks/webservices/library/ws-secure/ , April 2002

[25] Microsoft, “Web Services Policy Framework (WS-Policy) version 1.0,” http://msdn.microsoft.com/webservices/, December 2002

[26] IBM, ‘Web Services Transaction (WS-Transaction) version 1.0,” http://www-106.ibm.com/developerworks/library/ws-transpec/, august 2002

[27] D. O'Riordan, “Business Process Standards for Web Services, The candidates,” Web Services Architect, April 2002, http://www.webservicesarchitect.com/content/articles/BPSFWSBDO.pdf

[28] A. Arsanjani, B. Hailpern, J. Martin, P. Tarr, “Web Services Promises and Compromises”, ACM Queue, Vol. 1 No. 1, March 2003 http://www.acmqueue.org/

[29] H. Ossher and P. Tarr, “Using multidimensional separation of concerns to (re)shape evolving software,” Communications of the ACM, 44(10):43--50, Oct. 2001.

[30] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, Jr., “N degrees of separation: Multi-dimensional separation of concerns,” Proceedings of the 1999 International Conference on Software Engineering, pages 107-119. IEEE Computer Society Press / ACM Press, 1999.