Aspect-Oriented Programming for Dynamic Web Service Monitoring and Selection

Bart Verheecke, María Agustina Cibrán, Viviane Jonckers

System and Software Engineering Lab
Vrije Universiteit Brussel

{Bart.Verheecke, Maria.Cibran}@vub.ac.be, vjoncke@info.vub.ac.be

Abstract. In Service-Oriented Application Development, applications are composed by selecting and integrating third-party web services. To avoid hardwiring concrete services in client applications we introduced in previous work the Web Services Management Layer (WSML) and suggested a redirection mechanism based on Aspect Oriented Programming (AOP). Even though this mechanism enables hot swapping between semantically equivalent services based on their availability, this is not enough to create applications that are driven by business requirements. In this paper we introduce a more advanced selection mechanism that allows dynamic switching between services based on business driven requirements that can change over time. Choosing a service may be done based on cost, presence on approved partners list, as well as binding support, quality of service classifications, historical performance and proximity. We introduce a modular monitoring mechanism that is able to observe these criteria and trigger a more advanced service selection procedure. We show how the AOP language JAsCo with its dynamically pluggable aspects is well suited to achieve this.

1. Introduction

Web service technologies accelerate application development by allowing the selection and integration of third-party web services, achieving high modularity, flexibility and configurability. However, as it is identified in [3], existing approaches used in state-of-the-art CASE tools, such as Microsoft Visual Studio.NET, typically hard wire service proxies in client applications. This makes it difficult to achieve just-in-time discovery and integration of web services. Furthermore, the issue of selecting the best or the most appropriate service from those that provide the required functionality is currently not addressed by these tools. As stated in [1], this leads to unmanageable applications that cannot adapt to changes in the business context. The Internet is an ever-changing and unpredictable environment; therefore, clients need to be flexible enough to deal with failures and changing network conditions, old services that are abandoned and new services that become available. The goal of our research is to fully decouple a client application from the concrete web services it uses. In our approach clients can be written in a service independent way, referring only to the

functionality that is needed. As such, hot-swapping between services becomes possible: a service that is unavailable due to network conditions or service related problems can be easily replaced by a functionally equivalent one. While this mechanism allows high flexibility in the integration of web services, it is not enough to achieve applications that are driven by business requirements and need to stay competitive in a changing environment.

The aim of this paper is to extend this hot-swapping mechanism for the dynamic and optimal selection of services and service compositions taking into account business considerations. Our solution allows selecting web services based on the non-functional properties they advertise in their documentation or on their runtime behaviour which is monitored by dynamically introduced measurement points in the system. We show how *Aspect-Oriented Programming* (AOP) technology and in particular JAsCo [2] is well suited to realise the decoupling and modularisation of service selection policies and monitoring concerns.

The next section introduces the Web Services Management Layer as the context of this research. In section 3 we identify selection policies and motivate why AOP is well suited to modularize them. Section 4 introduces the main ideas behind AOP and JAsCo. Section 5 introduces our approach based on selection and monitoring aspects. A prototype implementation is discussed in section 6. Finally, section 7 presents related work and conclusions and future work are given in section 8.

2. Web Services Management Layer

To achieve a higher flexibility in Service Oriented Application Development (SOAD), we have presented in [3] [4] an intermediate management layer in between client applications and web services called **Web Services Management Layer (WSML).** In this earlier work we focused on eliminating the hard wiring of the services in client applications by introducing a flexible redirection mechanism which allows switching between multiple services and compositions that offer the same functionality. The WSML functions as a repository for services that can be used to functionally integrate with the application. Available services can be looked up in a central UDDI-registry or discovered using decentralised Web Services Inspection Language (WSIL) documents. The advantage of this approach is that the client application becomes more flexible as it can continuously adapt to the changing environment and communicate with new services that were not known or available at design time. Switching between services (or even service compositions) is done in a transparent way for the clients. This is done using Service Types, which are explained in more detailed in section 6.1. As such, the WSML actually realises dynamic integration of services.

In this paper, we suggest to extend our approach by adding support for dynamic and optimal selection of services based on business requirements.

3. Service selection criteria

In this section we identify selection policies that guide the just-in-time integration of web services in client applications and motivate the suitability of AOP to implement a flexible service selection and monitoring mechanism.

3.1. Identifying selection policies

In order to accomplish dynamic service selection based on changing business requirements, client applications need to define, as part of their requests, not only which functionality is needed but also which *non-functional requirements* are pursued. The consideration of non-functional requirements introduces the concept of the *most appropriate services*: the dynamic service selection mechanism should select the services that best fit the specified requirements and swap to them when needed. The non-functional requirements specified as part of the application requests imply the definition of *selection policies* used to govern the service selection. A *selection policy* defines a condition based on non-functional properties of services. These non-functional properties might need to be monitored during the execution of the application. Examples of non-functional properties of services are: response time, service cost, network bandwidth, service reliability and service dependability.

A service complies with a selection policy if it satisfies its condition. Only if a service complies with all specified selection policies it is *approved* to be considered during the selection process. An approved service can be selected and integrated in the application. If a service does not satisfy at least one selection policy, then it is *disapproved* and not considered for selection. Service selection policies can be classified as imperatives and guidelines. An **imperative** is a constraint on a service that should be satisfied at the moment the service is invoked. The constraint can contain absolute conditions (e.g. the cost of the service cannot exceed a fixed amount, the response time of a service cannot drop under some threshold), or can involve interrelationships with other services or the system (e.g. the cost of the service must be below the average of the cost of all registered services). If two or more services satisfy an imperative it is not determined which one must be chosen. This is where guidelines come in. A **guideline** specifies that if multiple services are available to provide the required functionality, one is preferred over the other (e.g. the cheapest service or the service with the highest encryption level must be selected). This implies that the services are compared with each other and a ranking is made.

3.2. Towards a flexible implementation of selection policies

After having identified what selection policies are, the challenge is how to implement them in a flexible way, avoiding the applications to change each time the business requirements change. We have conducted previous work [5] [6] [7] on the decoupling of business rules in the context of software applications developed using object-oriented or component-based software development techniques. These applications

have substantial core application functionality and are normally driven by business policies. Thus, they need to constantly cope with changes in the business requirements embodied in their policies. In this context it is increasingly important to consider *business rules* as a means to capture some business policies explicitly. A business rule is defined by the Business Rules Group as *a statement that defines or constraints some aspect of the business. It is intended to assert business structure or to control the behaviour of the business* [8]. As business rules tend to evolve more frequently than the core application functionality [9] [10], it is crucial to *separate* them from the core application, in order to *trace* them to business policies and decisions, *externalize* them for a business audience, and *change* them. In that work, contrary to typical approaches in the field which only focus on decoupling the business rules themselves [11] [12], we observe the need to decouple the code that links the business rules to the core application functionality. This linking code *crosscuts* the core application and thus need to be separated in order to achieve highly reusable and configurable business rules. In this previous research we show how AOP (see section 4) is ideal to decouple the crosscutting business rules links. Moreover, the following requirements are pursued:

- connect business rules to core application events which depend on run-time properties,
- pass necessary business objects to events in order to make business rules executable at that event,
- reuse a business rule link at different events,
- combine, prioritize and exclude business rules in case of interference,
- control the instantiation, initialization and execution of business rule links.

Successful experiments using AspectJ [13] and JAsCo [2] (see section 4) were performed achieving the identified requirements. In the context of Service-Oriented Application Development, it should be possible to select and integrate the services that best accommodate to the application requirements. For instance, it may be required to integrate the cheapest service at a given time. Analogously to business rules, selection policies are driven by business requirements and need to dynamically reflect the changes in the environment. A possible situation is that the cheapest service is not needed anymore but the fastest as result of changing the business requirements. As a conclusion, selection policies should be kept separated from the services and applications that integrate them to enhance maintainability, reusability and adaptability. The same way AOP resulted ideal to separate business rules links and connect them with the core application, AOP can be successfully used to plug-in and out selection policies that govern the selection and monitoring of services.

4. Aspect-Oriented Programming

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 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. Nowadays, several AOP approaches, such as AspectJ, Composition Filters [14], HyperJ [15] [16] and DemeterJ [17] are available. These technologies have already been applied on large industrial projects by for instance Boeing, IBM and Verizon Communications. For more information about AOP in general, we refer to [18].

However, a major drawback of most AOP approaches is that the aspects are woven with the core application at compile time. This would mean that every time the set of applicable business requirements changes, the application needs to be recompiled. Therefore, we need to use dynamic AOP technology. A suitable AOP language for achieving our objectives is JAsCo [2], a dynamic aspect oriented programming language that provides support to dynamically plug-in and out aspects in the application. Moreover, JAsCo aspects are described independently of a concrete deployment context, making them highly reusable in different concrete contexts. These characteristics make JAsCo suitable for web services applications and allow us to achieve the degree of flexibility pursued in the service selection. JAsCo is built on top of Java and basically introduces two new concepts:

- **Aspect Beans:** an aspect bean is an extension of the Java Bean component that specifies crosscutting behaviour in a reusable manner. It can define one or more logically related hooks as a special kind of inner classes. A hook specifies ***when*** the normal execution of a method should be intercepted and ***what*** extra behaviour should be executed at those points.
- **Connectors:** a JAsCo connector is responsible for deploying 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.

The dynamism in the creation of connectors constitutes JAsCo's key feature for the realisation of the dynamic service selection and integration as it is shown in the following section. For more information about JAsCo and the JAsCo component model, we refer to [2]. The use of JAsCo in the WSML is explained in section 6.

5. AOP for Dynamic Selection and Monitoring

We observe that selection policies and monitoring concerns are good candidates for aspects. After all, if web services are hard-wired in a client application, selection code needs to be included at each point where some service functionality is requested. This code results scattered in different places and tangled with code that solves other concerns of the application. Furthermore, the monitoring of services requires introducing measurement points at different points in the system. This monitoring

code would not only be scattered and tangled, but also it would require the client application to anticipate the introduction of all these monitoring points. As such, all these points should be known at design time, which might not be the case.

To achieve the required flexibility, we propose to use **selection aspects** to nicely modularize selection policies. One selection aspect contains the code on how to select the most appropriate web service for a given business requirement. Possibly, these selection policies require the monitoring of unforeseen non-functional properties. Therefore, it should be possible to plug in and out additional monitoring functionality on demand. To accomplish this, we use **monitoring aspects** to observe system, environmental or service changes. For instance, if a selection policy specifies to use the fastest service available at any given time, the performance of the involved services needs be monitored by measurement points introduced by monitoring aspects. The monitoring points are precise moments in the execution of the system where the desired properties are affected. Monitoring aspects allow identifying these execution points and specifying which monitoring logic should be executed at exactly that time. This way, monitoring aspects allow the decoupling of monitoring from the applications. As such, AOP allows realizing a modularized non-invasive selection and monitoring mechanism. The use of selection and monitoring aspects in the context of the WSML is depicted in Fig. 1.

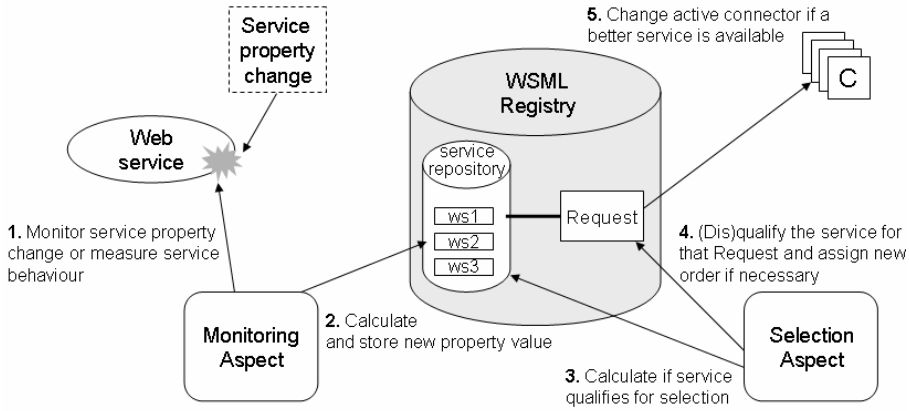

Fig. 1. Service Monitoring and Service Selection Aspects in the WSML

6. Prototype Implementation

A fully implemented prototype of the WSML is available at [19]. It is developed in Java and uses JAsCo for implementing the identified aspects, since its features are well suited to achieve the required flexibility. This implementation is deployed and integrated with a platform of Alcatel Bell to realise a demonstrator in the domain of service provisioning in broadband networks.

6.1. Decoupling web services from client applications

Fig. 2 illustrates the overall architecture of the WSML using JAsCo aspect beans and connectors. To decouple the client application from specific web services, the notion of *Service Type* is introduced in the WSML: a generic specification of the required functionality without references to specific web services. A service type can be seen as a contract specified by the application towards the services and allows hiding the syntactical differences between semantically equivalent services. 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

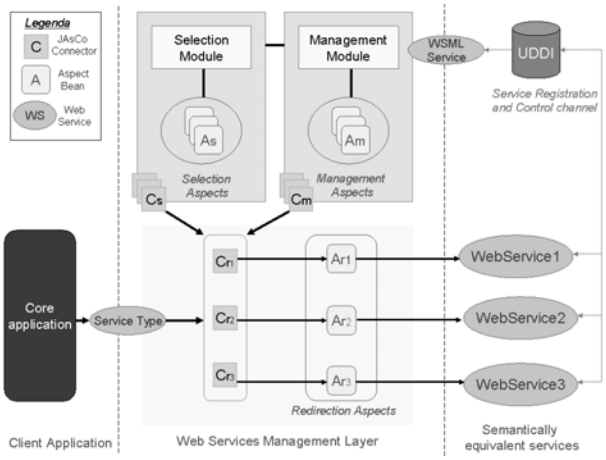

Fig. 2. Detailed Architecture of the WSML

Concrete services with different interfaces can be registered for a service type. Then, client applications can make a service type request and the WSML translates these generic requests to concrete web services invocations. In this mechanism we identify *redirection aspects*, which define the logic of intercepting client application requests and replacing them with concrete web service invocations. As such they encapsulate all *communication details* for a specific service or service composition. In addition, redirection aspects need to be dynamically plugged-in and out to reflect the volatility of the service environment. By creating a new connector and redirection aspect bean at runtime the new service or service composition can be integrated in the client application in a transparent way. The current version of the WSML supports full automatic generation of the connectors and provides tool support for the creation of the aspects. The proposed mechanism also enables hot swapping between 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 simply activating its connector and deactivating the previous one. Notice that this mechanism assumes simple request-response communications between the WSML and the Web Services: a hot-swap can take place at any time. If more advanced business processes need to be executed between services that keep state, swapping to another service becomes more difficult. It is clear that in this case compensation actions might be required to rollback the state of the service and maybe even the state of the application. A detailed discussion on this topic is however outside the scope of this paper.

6.2. Example application

To illustrate these ideas, an example of a travel agency application is introduced. The application offers the functionality to book holidays online that customers can use to make reservations for both flights and hotels. To achieve this functionality, the application integrates several web services. Suppose *HotelServiceA* and *HotelServiceB* are services that offer the same functionality for the online booking of hotels. Each hotel service returns the same type of results. The client application has to present a list of available hotels to the customer. To this end a *HotelServiceType* is defined which specifies the following method: `List listHotels (Date, Date, CityCode)`. At deployment or run time the following two services are available:

HotelServiceA provides the method:
`HotelList giveHotels(CityCode,Date,Date)`
HotelServiceB provides the method:
`HotelList getHotels (Date,Date,CityName)`

6.3. Dynamic service selection and monitoring

The goal of this section is to show how the WSML considers non-functional properties and Quality of Service constraints (QoS) to guide the selection of the most appropriate services. Selection criteria can be based on properties defined in the services descriptions and that can be retrieved and checked at the moment the criteria are applied (e.g. price, distance). Another possibility is that the properties involved in the selection criteria are not anticipated and defined in the services. These properties depend on the behaviour of the service at run-time. Examples of such properties are average response time, number of successful invocations, etc. Current approaches only provide limited or no support in the consideration of non-functional properties in the service selection. Following an example of an imperative selection policy that works on the properties of services is given. For instance, assume the hotel application wishes to communicate only with web services that can respond within 5000 milliseconds. As a consequence of this imperative, the distinction between approved and disapproved hotel services has to be made. Suppose the system did not foresee the functionality to actually monitor the response time of the web services. Using traditional software engineering approaches, it would not be straightforward to introduce and remove non-invasively the required measurement points at runtime. On the contrary, in our approach we can achieve this by defining a monitoring aspect that

will calculate the average speed for each of the available services. The aspect needs to hook before and after the method execution of `HotelServiceType.listHotels(args)` in the service type:

- before the method is executed, a timer is started
- after the method is executed, the timer is stopped, the average speed is calculated and stored.

This is depicted in figure 3 in lines 23 to 39. First, the aspect needs to be initialised in lines 3 to 10. There, the new property *AvSpeed* is added to each service and initialised with a start value *zero*. The same is needed for each new service that is registered later in the WSML. This is done in the *ServiceAddedHook* in lines 13 to 20.

```
1. class ServiceSpeedLogging {
2.    ...
3.    public void initialiseAspect (String ruleName,
4.       String serviceTypeName, String reqName) {
5.       ...
6.       for (int i=0, WSML.getNumberOfServices(), i++) {
7.          Service ws = WSML.getService(i);
8.          ws.addProperty ("AvSpeed", initValue);
9.       }
10.   }
11.
12    //Aspect Hook when a new service is added
13    hook ServiceAddedHook {
14.      ServiceAddedHook(method(Service ws)){
15.        execute(method);
16.      }
17.      after () {
18.        ws.addProperty ("AvSpeed", initValue);
19.      }
20.    }
21.
22.    //Aspect Hook to do the actual monitoring of the speed
23.    hook SpeedLoggingHook {
24.       private long start, stop = 0,
25.       private Service service;
26.
27.        //... hook constructor omitted here
28.        before () {
29.           service = WSML.getActiveService();
30.           start = System.currentTimeMillis();
31.       }
32.       after () {
33.          stop = System.currentTimeMillis();
34.          if (service == WSML.getActiveService()) {
35.              //calculate average speed
36.              service.setProperty ("AvSpeed", average);
37.           }
38.       }}}
```

Fig. 3. Monitoring aspect for determining the average speed of a service

After having defined the monitoring aspect which makes the necessary property available, a selection aspect can be defined to approve and disapprove the available

hotel services based on their average speed. The WSML defines a library of reusable and generic selection policy templates which determines which web services to consider when satisfying a service type request. Fig. 4 shows a simplified version of the code for a basic selection policy template which approves and disapproves the services registered for a service type according to the values of a certain property.

```
1.  class ServicePropertySelection {
2.    …
3.    public ServiceType serviceType;
4.    public List approved, disapproved;
5.    public Object maximum, minimum;
6.    public String property;
7.
8.    // Aspect hook to approve or disapprove a service
9.    // when it is added for the first time depending on its
10.   // value of property
11.   hook WebServiceAddedHook {
12.     WebServiceAddedHook(method(WebService ws)){
13.       execute(method); }
14.     …
15.     after () {
16.        if (checkPropertyOfWebService(ws)) approve(ws);
17.            else disapprove(ws); }
18.   }
19.
20.   // Aspect hook to approve or disapprove a service
21.   // whenever its value of property changes
22.   hook PropertyChangedHook {
23.     PropertyChangedHook(method(String mproperty,..args)){
24.       execute(method); }
25.     …
26.     after() {
27.       WebService ws = (WebService)calledobject;
28.       if (checkPropertyOfWebService(ws)) approve(ws);
29.       else disapprove(ws);
30.       // trigger the activation of the most appropriate
31.       // service for serviceType
32.       global.serviceType.activate(); }
33.     }
34.
35.   // Aspect hook that computes the list of services
36.   // (among the ones registered for serviceType) that
37.   // satisfy the policy
38.   hook GetApprovedServicesHook {
39.     GetApprovedServicesHook (method(..args)){
40.       execute(method); }
41.     …
42.     replace() {
43.       List listPreFiltered = (List)proceed();
44.       List myApprovedServices = global.getApproved();
45.       myApprovedServices.retainAll(listPreFiltered);
46.       return myApprovedServices; }
47    }}
```

Fig. 4. Selection policy aspect for the approval of services according to values of a property

In the `ServicePropertySelection` aspect, the hooks `WebServiceAddedHook` (lines 11 to 18) and `PropertyChangedHook` (lines 22 to 33) approve or disapprove a

web service depending on the value of the property the policy is defined upon. This behaviour is executed when the service is added for the first time or when the service changes the values of its pertinent property respectively. The hook `ActivationListHook` (lines 38 to 47) retrieves from the list of services registered for `serviceType`, only the services that satisfy the policy. Note that in the case more than one policy is defined, the different selection aspects need to cooperate to come up with only one list of services that satisfy all the policies. If this is the case, the `proceed()` (line 43) proceeds with the enforcement of the other policies and afterwards, on this pre-filtered list of services, only the ones approved by the current policy will be kept and considered for activation by the WSML.

Because of the reusability of JAsCo aspect beans, a selection policy aspect can be instantiated and initialised with different parameters at runtime by creating new connectors. In order to define a selection policy which filters services according to their average speed (property captured in the monitoring aspect presented in figure 3), the ServicePropertySelection aspect needs to be deployed with the specific values: `serviceType` = HotelServiceType, `minimum` = 0, `maximum` = 5000, `property` = "avSpeed", as well as the abstract methods of the hooks need to be linked to the concrete methods for the addition of a web service, the setting of a web service property and the listing of registered web services respectively. This deployment is done in a JAsCo connector (omitted here for space reasons). Note that if a new service is added in the WSML, the *average speed* property will not be set until it is actually invoked. To avoid new services to never have the chance to be invoked, as they have no reputation yet, their selection could be based on the recommendations or endorsement by trusted third parties.

Other reusable selection policy templates are provided in the WSML, such as a policy which orders the approved services according to the values of their properties and a policy ensuring the addressing of always the same service for the requests in a transaction. These policies can be deployed dynamically in JAsCo connectors, allowing a wide variety of selection policies to be specified and reinforced at runtime. Note that, even if only shown in this paper for service types, selection policies can also be specified individually at the level of each service type request to realise a more fine-grained selection.

For each service policy, one connector and one aspect bean instance is created. This could imply a certain overhead if many selection policies need to be enforced at run-time for a given service type. In JAsCo, an aspect-oriented just-in-time compiler called Jutta [20] is provided to optimize the aspect interpreter system and hence enhance performance. It is important to consider that sometimes searching for the "best" solution could be computationally expensive. Moreover, because the "best" may change frequently over time, unwanted oscillations can seriously affect performance. In this regard, selection policies could be useful as a mechanism to prevent undesired oscillations between services.

7. Related Work

A lot of research is going on in the web service context and a lot of vendors are currently working on dedicated web service management platforms. However, most of these products focus on the server-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. 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 Framework (WSDF) [21] incorporates ideas from the Semantic Web [28] 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) 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.

A related approach to enable more advanced web service selection is agent-based architectures. Agents are autonomous, computational entities that perceive their environment through sensors and act upon their environment through effectors [23]. As such, they can be used to monitor and select services in a non-invasive manner. In [24] a solution is suggested to collect, aggregate and disseminate rating information on the usage of web services by proxy agents. It uses a social algorithm to make selections. A prototype of this system is under development.

The idea of applying AOP concepts to decouple web services concerns is quite innovative and thus not many approaches have been developed focusing on this field. However, Arsanjani et al. [25] have recently identified the suitability of AOSD to modularize the heterogeneous concerns involved in web services. However, they refer to approaches like AspectJ [13] and the Hyper/J [15] [16] which only allow static aspect weaving though, 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.

8. Conclusions and Future Work

In this paper we show how dynamic service selection can be realised as part of the WSML using AOP. We identified the need to consider service criteria to drive dynamic service selection and showed how JAsCo, a dynamic AOP language, can be successfully used for its realisation. JAsCo aspects are used to enhance the hot-swapping mechanism with the consideration of service criteria that can be added dynamically and to monitor service properties that will affect the selection. The advantage of using JAsCo is that the creation, addition and removal of aspects can be

done at run-time, achieving the desired flexibility needed in the evolving and changing web services environment.

In state-of-the-art distributed computing infrastructure supporting Web Services, the service functionality is written down in terms of XML syntax without a well-defined semantics, such as WSDL, XLANG, WSFL or the more recent WS-BPEL [26]. The Semantic Web project [27] addresses these issues by developing a rich language allowing to describe all the complex interrelationships and constraints between services and other web objects. During the discussion in this paper we specify selection criteria using universal terms assuming that both the WSML and the Web Services talk the same language and understand each others capabilities. However, to realise this outside an experimental scenario and deploy the WSML in a complex heterogeneous environment like the internet, such a language is highly needed. Several ontology languages are currently under development, they include the Resource Description Framework [28], the Web Ontology Language [29] and DAML+OIL [30]. While this paper focuses only on the technical part of using AOP for advanced selection, a semantic web language can be adopted by the WSML to unambiguously describe all service criteria. Note that this also requires that the web services themselves must be described in this expressive language. OWL-S (formerly known as DAML-S) [31] is an example of such a language, using the DAML+OIL ontology.

This implementation of the WSML can be continued in different directions. In the implementation we have presented, selection aspects and their connectors are written by hand. There are two lines of work in this respect: either to choose one of the formal languages based on ontologies to specify the selection criteria and to (semi) automatically translate these descriptions into aspects and connectors. To achieve this, automatically code generation for the aspects and connectors is required. The other alternative is to provide a library of well defined aspect templates that can be easily deployed and used by the developer of the client application through wizard-based tool support.

So far, the selection aspects suggested are based on service properties that can be queried or monitored. Other kinds of service criteria involving other services characteristics (not necessarily properties) can be considered. For instance, we can imagine that we want to select services depending on certain behavioural patterns or dependencies with other services. Also the service selection can be extended to perform the service choice depending on conditions applied to the result of service invocations. The identification and definition of other selection aspects to contemplate these cases is subject of ongoing research.

Acknowledgements

The research presented in this paper is conducted with the support of the Flemish Government Project MOSAIC, funded by the "*Institute for the Innovation of Science*

and Technology in Flanders" (IWT) or in Dutch "*Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen*", and is carried out in cooperation with Alcatel Bell.

References

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

[2] D, Suvée, W. Vanderperren and V. Jonckers, "JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development", Proc. of Int. Conf. on Aspect-Oriented Software Development (AOSD), Boston, USA, pp 21-29, ISBN 1-58113-660-9, ACM Press, March 2003.

[3] B. Verheecke, M. A. Cibrán and V. Jonckers, "AOP for Dynamic Configuration and Management of Web Services," Proc. of ICWS'03-Europe, Erfurt (Germany), September 2003. To be published in the Int. Journal on Web Services Research (JWSR), Vol.1 No. 3, July-September 2004.

[4] M. A. Cibrán, B. Verheecke and V. Jonckers, "Modularizing Client-Side Web Service Management Aspects", Proc. of 2nd Nordic Conf. on Web Services (NCWS'03), Växjö (Sweden), November 2003

[5] M. A. Cibrán, M. D'Hondt and V. Jonckers, "Aspect-Oriented Programming for Connecting Business Rules", 6th Proc. of Int. Conf. on BIS, Colorado Springs, USA, 2003.

[6] M. A. Cibrán, M. D'Hondt, D. Suvée, W. Vanderperren and V. Jonckers, "JAsCo for Linking Business Rules to Object-Oriented Software", Proc. of Int. Conf. on CSITeA, Rio de Janeiro, Brazil, June 2003.

[7] M. A. Cibrán, D. Suvée, M. D'Hondt, W. Vanderperren and V. Jonckers, "Integrating Rules with Object-Oriented Software Applications using Aspect-Oriented Programming", to be published in Proc. of the 5th Argentine Symposium on Software Engineering (ASSE'04), Córdoba, Argentina, September 2004.

[8] The Business Rules Group. "Defining Business Rules: What Are They Really?", http://www.businessrulesgroup.org/, July 2000.

[9] G. Kappel, S. Rausch-Schott, W. Retschitzegger, and M. Sakkinen, "From rules to rule patterns", Proc. of Int. Conf. on Advanced Information Systems Engineering, pp. 99-115, 1996.

[10] B. von Halle, "Business Rules Applied", Wiley, 2001.

[11] B. N. Grosof, Y. Kabbaj, T. C. Poon, M. Ghande, and et al., "Semantic Web Enabling Technology (SWEET)", http://ebusiness.mit.edu/bgrosof/

[12] I. Rouvellou, L. Degenaro, K. Rasmus, D. Ehnebuske and B. McKee, "Extending business objects with business rules", TOOLS Europe 2000, St-Malo, France, pp. 238-249, 2000.

[13] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersen, J. Palm, and W. G. Griswold, "An overview of AspectJ", Proc. of ECOOP'2001, volume 2072 of Lecture Notes in Computer Science, pp. 327-353, Berlin, Heidelberg, and New York. Springer-Verlag.

[14] L. Bergmans, M. Aksit, "Composing Crosscutting Concerns Using Composition Filters", Communications of the ACM, Vol. 44, No. 10, pp. 51-57, October 2001.

[15] H. Ossher, P. Tarr, "Using multidimensional separation of concerns to (re)shape evolving software", Communications of the ACM, 44(10):43-50, October 2001.

[16] P. Tarr, H. Ossher, W. Harrison, S. Sutton, "N degrees of separation: Multi-dimensional separation of concerns", ICSE 1999, pp.107-119. IEEE Computer Society Press/ACM Press.

[17] K. Lieberherr, D. Orleans, J. Ovlinger, "Aspect-oriented programming with adaptive methods", Communications of the ACM, Vol. 44, No. 10, pp. 39-41, October 2001.

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

[19] Web Services Management Layer (WSML), http://ssel.vub.ac.be/wsml/
[20] Vanderperren, W., Suvee, D. "Optimizing JAsCo dynamic AOP through HotSwap and Jutta", Proceedings of Dynamic Aspects Workshop published as RIACS Technical Report 04.01., Lancaster, UK, March 2004.
[21] A. Eberhart, "Towards Universal Web Service Clients", Proc. Euroweb 2002, UK.
[22] S. Chatterjee , "Developing Real World Web Services-based Applications", The Java Boutique, http://javaboutique.internet.com/articles/WSApplications/
[23] G. Weiss, "Multi agent Systems, a Modern Approach to Distributed Artificial Intelligence", The MIT Press, Cambridge, Massachusetts, 1999, isbn 0-262-23203-0
[24] E. M. Maximilien, M. P. Singh, "Agent-based Architecture for Autonomic Web Service Selection", Proc. of Workshop on Web Services and Agent-based Engineering, Melbourne, Australia, 2003
[25] 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/
[26] BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, "Business Process Execution Language for Web Services (BPEL4WS) v 1.1", May 2003
[27] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web," Scientific American, vol. 284, no. 5, May 2001, pp. 34-43
[28] Resource Description Language http://www.w3.org/RDF/
[29] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein "OWL Web Ontology Language Reference", W3C Working Draft 31 March 2003.
[30] D.L. McGuinness, R. Fikes, J. Hendler, L.A. Stein., "DAML+OIL: An Ontology Language for the Semantic Web," IEEE Intelligent Systems, vol. 17, no. 5, Sept./Oct. 2002, pp 72-80.
[31] A. Ankolekar, M. Burstein, J. R. Hobbs,O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K.Sycara, "DAML-S: Web Service Description for the Semantic Web," The Semantic Web – ICWC 2002: Proc. 1st Int. Semantic Web Conference (ISWC), Springer-Verlag, Berlin 2002, pp. 384-363