WEB SERVICE COMPOSITION USING THE WEB SERVICES MANAGEMENT LAYER

Niels Joncheere, Bart Verheecke, Viviane Jonckers
System and Software Engineering Lab (SSEL), Vrije Universiteit Brussel, Belgium
{*njonchee,bverheec,vejoncke*}*@ssel.vub.ac.be*

Sofie Van Hoecke, Gregory Van Seghbroeck, Bart Dhoedt
INTEC Broadband Communication Networks (IBCN), Universiteit Gent, Belgium
{*sofie.vanhoecke,gregory.vanseghbroeck,bart.dhoedt*}*@intec.ugent.be*

Keywords: Aspect-oriented programming, brokering, performance and scalability measurement, web services

Abstract: The Web Services Management Layer (WSML) is an academic middleware platform that is placed in between clients and web services. It takes care of dynamic integration, selection, composition and client-side management of web services, thus removing the need to take care of these concerns in clients. This paper describes how the WSML can be used to express web service compositions, and presents the results of a number of experiments which evaluate the performance and scalability of the current WSML implementation. These results show that the WSML induces only a small overhead compared to systems in which the WSML has not been deployed. The advanced features of the WSML therefore come a low cost in terms of performance and scalability, which makes it a viable option in real-life web services applications.

1 INTRODUCTION

An important domain in web services research is the design and development of languages that clearly and naturally describe the logic of web service compositions. This has given rise to a number of dedicated *web service composition languages*, most of which are based on earlier research on workflow languages (Du and Elmagarmid, 1997). Currently, the most popular web service composition language is the *Business Process Execution Language* (BPEL) (Andrews et al., 2003). BPEL processes are platform- and transport-independent, and are expressed using XML.

In current dynamic web services environments, web services are offered by different service providers. This poses a variety of challenges for the clients of these services:

- *Redirection.* Services can become unavailable due to service or network issues, services may be replaced by newer versions or taken out of business, syntactical and semantical mismatches can occur, etc. The clients need to be able to communicate with services that were not known at development or deployment time, and possibly compose services together if necessary.
- *Selection.* Multiple services may offer the same or similar functionality. In this case, a client must be able to swap transparently to another service when this service better suits the needs of the client. The selection of the most optimal service can be based on non-functional business requirements of the client (e.g. preferring a cheaper service over a faster one), or may be part of a load balancing strategy such as round-robin, broker-side monitoring, and server-side monitoring.
- *Client-side management.* Web services impose a variety of requirements on clients such as complying with an encryption protocol, performing a pre-payment or following a specific authentication protocol. These concerns may vary from service to service and from time to time. Furthermore, clients may want to perform logging, caching, pre-fetching, monitoring, etc. when they engage in service communication.

This paper discusses web service composition in the context of the *Web Services Management Layer* (WSML). The WSML is an academic middleware platform for dynamic integration, selection, composition and client-side management of web services, and is based on *aspect-oriented* techniques. The WSML is already reported on in previous work (Verheecke et al., 2003; Verheecke et al., 2004; Verheecke et al., 2006). In this paper, we focus on how the WSML can be used to express web service compositions, and measure the performance and scalability of our approach, i.e. the time required to handle individual requests to a composition (response time), and the number of requests to a composition that can be handled concurrently (throughput).

The outline of the paper is as follows. Section 2 introduces the context of our work. Section 3 describes web service composition using the WSML. Section 4 describes our tests and discusses their results. Section 5 gives an overview of related work, and Section 6 states our conclusions and proposes some directions for future work.

2 THE WEB SERVICES MANAGEMENT LAYER

2.1 Introduction

The Web Services Management Layer (WSML) (Verheecke et al., 2003; Verheecke et al., 2004; Verheecke et al., 2006) is a mediation framework targeted at dynamic service environments. As discussed in the introduction, such environments are associated with the problems of redirection, selection and client-side management. In order to avoid that service clients need to deal with these problems in an ad-hoc fashion on a service-per-service basis, the WSML aims to deal with each of these concerns in a client- and web service-independent manner. The WSML is placed in between the client and the world of web services, as depicted in Figure 1. On the left-hand side, the client requests service functionality without referencing the concrete web services. The WSML is responsible for intercepting the client requests and choosing the most appropriate service available on the right-hand side or combining a number of services together in a composition, invoking them in a manner that complies with any management requirements imposed by service providers and returning the results to the client. Using the WSML for service-oriented applications has the advantage that more robust and flexible applications can be developed without having to rewrite service-related code. Furthermore, replacing the web service-specific invocations with generic service requests and extracting all extra web service selection and management code from the client applications facilitates future maintenance of the client application code.

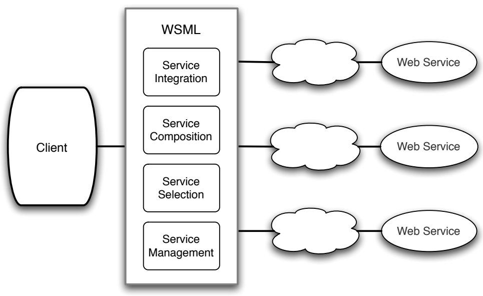

Figure 1: The Web Services Management Layer

2.2 Aspect Oriented Programming

Traditional software engineering methodologies do not allow modularizing the challenges of web service redirection, selection and client-side management as separate entities. Instead, the implementation of these concerns will end up scattered across the different modules of the system, and tangled with other concerns. The code for different concerns becomes intermixed, possibly at multiple places in the system. Because these *crosscutting concerns* are spread and repeated over several modules in the system, it becomes very hard to add, edit, verify, test or remove such concerns individually. Moreover, the scattered and tangled code seriously hampers the evolution of the concerns and the base application.

Aspect-Oriented Programming (AOP) (Kiczales et al., 1997) is a software engineering paradigm that aims to solve these problems by achieving better *separation of concerns* (Parnas, 1972). The goal of AOP is to allow developers to cleanly modularize crosscutting concerns. Therefore, AOP introduces an additional module construct, which is called an *aspect*, to encapsulate the crosscutting concern. These aspects can then be added, edited, verified, tested or removed without without impacting the base application. The points in the program's execution where an aspect must be applied are called *joinpoints* and they are described using a *pointcut language*.

2.3 AOP in the WSML

In the WSML, a variety of aspects are employed:

- *Redirection aspects* modularize the logic to compose multiple services together in a service-independent manner, contain the mapping for individual web services and deal with syntactical or semantical mismatches between what the client expects and what the service offers.

- *Selection aspects* enforce the selection of the most optimal service based on quality-of-service (QoS) properties.
- *Management aspects* enforce client-side management concerns imposed by the service or the client, such as logging, billing and caching.

Figure 2 depicts the architecture of the WSML, based on AOP. The client makes service requests on *service types*, i.e. invariable interfaces that are exposed to the client. These requests are intercepted by redirection aspects that invoke the appropriate web service(s) and return the result(s) to the client. Which particular redirection aspect is triggered is determined by one or more selection aspects. Additionally, management concerns are enforced by management aspects. Each concern is nicely modularized in the system by a different aspect and deployed by a specific connector (indicated in the figure with "C"). Each connector specifies the deployment logic of an aspect. As such, the aspect remains completely reusable as it only contains "what" needs to be done to enforce the concern. The connector specifies how and where the concern needs to be enforced in the system.

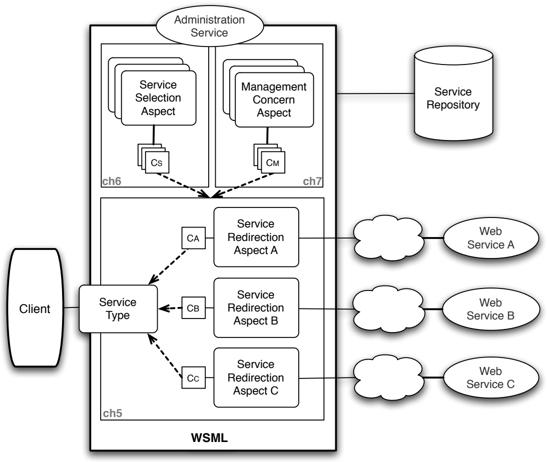

Figure 2: WSML Architecture

With a static AOP approach, aspects are woven into the base code at compilation time. This means that adding or removing aspects requires recompiling the system. Using a dynamic AOP approach it becomes possible to add and remove aspects at runtime by providing traps at the possible joinpoints where an aspect might be applied. A WSML prototype has been implemented using JAsCo (Suvée and Vanderperren, 2003), a dynamic AOP technology built on top of Java. By exploiting the dynamic capabilities of JAsCo, the WSML can be reconfigured at runtime and can deal with events and changes in a dynamic service environment without having to stop, or worse, rewrite the system.

3 SERVICE COMPOSITION IN THE WSML

Service composition is the *orchestration* of a number of existing services to provide a richer composite service assembled to meet the requirements of the client. As explained in (Chakraborty and Joshi, 2001), a differentiation can be made between proactive compositions and reactive compositions. *Proactive compositions* are composed off-line for deployment in a stable, always-up, resource rich platform. Typically, the composition is specified with a fixed set of partners in mind. *Reactive compositions* on the other hand are composite services that are created on the fly based on more volatile partner agreements, often optimizing for real-time parameters such as available network bandwidth. Reactive compositions reference partner roles that are filled in at runtime.

Our dynamic service binding mechanism supports both kinds of compositions, which makes it possible for compositions to be adaptable: for instance, a non-responding service in the composition can be replaced by a semantically equivalent one. Compositions in the WSML can also better deal with long-term changes and evolution, as each composition is modularized as a first-class entity: a composition aspect can be changed and recompiled easily without affecting the rest of the system. For pro-active compositions, a dedicated composition aspect is written. The invocation of the web services is addressed in the composition aspect's advice. To create reactive compositions that do not reference concrete services, the aspect is adjusted to reference service types again. Invoking service types from within a service composition allows for the specification of a composition in a generic way without hardwiring concrete service interfaces. It also avoids the explosion of the number of service compositions that needs to be specified in case multiple partners are available to fulfill a specific role in a composition. By specifying for each individual service type which service(s) and composition(s) can be used to handle a request, a temporal composition is created that best fits the criteria of the client. The complete redirection mechanism, supporting transparent mappings between service types, web services and service compositions is illustrated in Figure 3. A service type is either fulfilled by a single concrete web service, or by a service composition. A composition is composed out of concrete web services and/or service types.

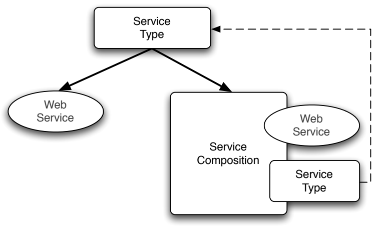

Figure 3: WSML service composition

An important implication of reactive compositions is that a temporal composition is “created” at the moment a client request comes in. Each service type the composition refers to is a partner role which is being filled in by a concrete service. The service types’ support for transparent hot-swapping introduces the concept of *dynamic partner roles*: each partner in a reactive composition can be replaced by another one.

4 EVALUATION OF THE WSML

In this section, we investigate the performance and scalability of the WSML. The investigated performance metric is the response time. To indicate the scalability of the WSML, the maximum throughput is measured. In the throughput investigation, an exception rate of 10% is allowed in order to discard the possible exception fluctuations caused by the test environment.

4.1 Test Environment

Figure 4 shows our client behavior using the BPMN notation. It consists of a number of consecutive invocations of a broker service. The broker service will forward each of these invocations to a dummy web service with a very small execution time. Thus, the overhead incurred by the middleware plays a very important role in the overall execution times and is not hidden by large server execution times.

The dummy web service has two identical endpoints, each situated on different physical machines (see Figure 5: Server 1 and Server 2). These two endpoints can be used for load balancing and thus provide a certain level of QoS for the composed process. Load balancing is an example of the selection challenge talked about in the introduction. The broker service (see Figure 5: Broker), which also runs on a different physical machine, will choose at runtime which web

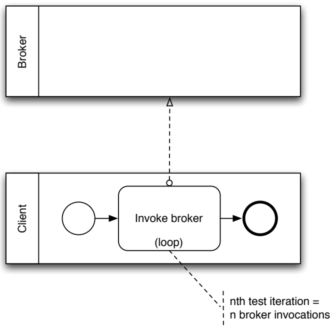

Figure 4: Client behavior

service endpoint will be invoked. To make this decision, three different brokering solutions have been implemented: round-robin, broker-side monitoring, and service-side monitoring. Each of these is elaborated below.

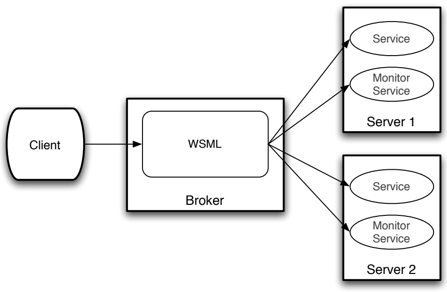

Figure 5: Test environment

The WSML runs on top of a Systinet for Java 6.5 server. The client machine (see Figure 5: Client) will fire service invocations to the broker in a multi-threaded fashion. Each thread simulates a client that wishes to use the web service. During a certain period of time every second a set of client threads will be instantiated. The number of instantiated threads resembles the throughput. Each individual client thread is responsible for logging its own response time.

4.2 Brokering Solutions

Round-robin is a simple and easy to implement brokering solution. It alternates in a deterministic way between the different service endpoints. This means that all the service endpoints will be invoked an equal number of times. Round-robin is especially suited for brokering when the different service endpoints have (almost) the same QoS properties.

Broker-side monitoring will allow the broker to make a decision based on some monitored property.

For example, this monitored property can be the response time of a service invocation or the network latency between the broker server and the service endpoints. The broker is responsible for gathering the data that defines the property; this can be handled by a separate thread or even by a separate tool installed on the broker. The data (or an aggregated version of it) can be requested by the broker to decide which service endpoint will be invoked. This will result in one additional internal function or internal service call per actual service invocation. The implementation used in the test environment monitors the response times for the service endpoints in a separate thread. These response times are aggregated to represent the service rate (the inverse of the average response times) per endpoint.

Server-side monitoring is almost the same as broker-side monitoring, except the property is monitored on the service endpoint itself. To retrieve the data that defines the property, the broker has to invoke an external web service on the service's endpoint. This will give per actual service invocation a number of additional external service calls equal to the amount of service endpoints. The advantage of server-side monitoring is that server related properties (e.g. total server load, service price) can be monitored as well.

4.3 Test Results

The graphs in Figure 6 and Figure 7 illustrate the results of our tests with regard to response time and throughput, respectively. Each graph shows a number of series, three of which concern the WSML:[1]

- *wsmlRR:* WSML approach with round-robin. A dedicated round-robin selection aspect makes sure each new client request is redirected to the next service through the appropriate redirection aspect.
- *wsmlBM:* WSML approach with broker-side monitoring. A dedicated monitoring aspect inserts measuring points in the composition and monitors the response times of the dummy services. This monitoring data is used by a selection aspect to select the fastest service with a fixed probability.
- *wsmlSM:* WSML approach using server-side monitoring. The monitoring aspect does not do any internal monitoring but rather obtains the monitoring data from external monitoring services. In our tests, the monitoring data is simply a random number.

[1]Series are combined when there is no significant difference between them.

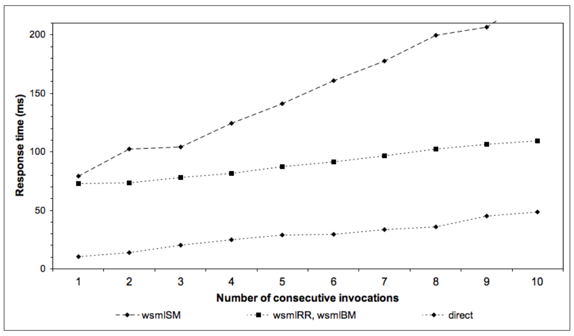

Figure 6: Detailed view of the response times for the test cases

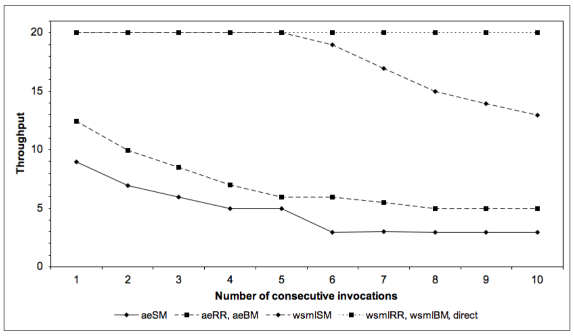

Figure 7: Throughput for the test cases

Figure 6 shows the average response times (in milliseconds) measured during 2 minutes of testing with 2 clients. We performed each test 10 times, with the number of consecutive invocations executed by the client composition ranging from 1 to 10.

In order to identify the performance overhead induced by the WSML, we measured the response time and throughput of the same client behavior, but with a broker on which the WSML has not been deployed, i.e. a broker that simply forwards each request to a single web service. The results of this additional test is shown in our graphs as the "direct" series. Because this test reduces the amount of code that is executed by the broker in response to each service invocation, response times are lower than our previous tests. The round-robin and broker-side monitoring tests induce a constant overhead, while the server-side monitoring test induces an overhead that increases with the number of consecutive invocations. However, all WSML series exhibit a linear behavior, which gives a first indication that the WSML performs well.

Figure 7 shows the maximum number of invocations that could be executed concurrently without causing exceptions. We started a maximum number of 20 simultaneous invocations, because this number is sufficient to show the differences between the investigated approaches. We performed each test 10 times, with the number of consecutive invocations executed by the client composition ranging from 1 to 10. The graph shows that the round-robin and broker-side monitoring tests scaled perfectly, but the server-side monitoring test started to have lower throughput when the client composition executed more consecutive invocations. In this scenario, the direct test scaled perfectly as well, but because it does not perform any load balancing, its throughput would be lower if more invocations were to be started simultaneously.

The previously mentioned tests give some initial indications on the performance and scalability of our approach. The direct test acts as a reference point for our measurements: our broker can never perform better than a broker that does nothing. In order to obtain an additional reference point and get an appreciation of our performance and scalability in relation to other state-of-the-art composition approaches, we re-implemented our three brokering solutions as BPEL processes and measured their performance and scalability on a modern BPEL engine, more specifically the ActiveBPEL 3.0 engine. We do not expect our results to be representative of all BPEL engines: we merely selected the ActiveBPEL engine because it is the most commonly used open-source BPEL engine. Although commercial offerings may have different behavior, the ActiveBPEL engine does put our approach into perspective. Therefore, Figure 7 and Figure 8 each contain three additional graphs:[2]

- *aeRR:* ActiveBPEL approach with round-robin. The different versions of the service endpoints need to be known at design time. This round-robin implementation just cycles over the list of service endpoints.
- *aeBM:* ActiveBPEL approach using broker-side monitoring. A SOAP handler intercepts all outbound service calls. When a synchronous web service is invoked, the response time is logged. An internal function, which is used inside the workflow, will calculate the service rate upon which the BPEL script will make a decision.
- *aeSM:* ActiveBPEL approach with server-side monitoring. The invoked external monitoring service randomly generates a value, which will be used to make the load balancing decision.

The WSML outperforms ActiveBPEL in each of the test scenarios. The round-robin and broker-side monitoring series are still linear, but have a much

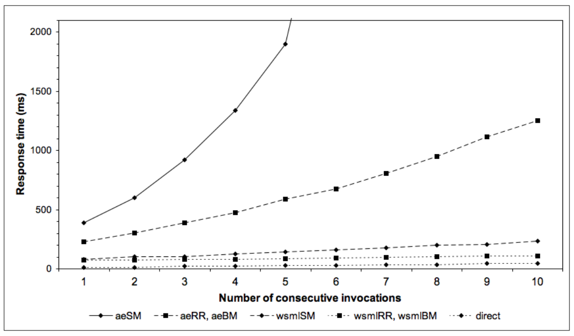

Figure 8: Response times for the test cases

greater slope than the corresponding WSML scenarios. The server-side monitoring service seems to exhibit exponential behavior. Based on our comparison of the WSML with a direct approach and a state-of-the-art composition approach, we can safely conclude that the performance and scalability of the WSML is good.

5 RELATED WORK

5.1 QoS Brokering of Web Services

Related work in this area focuses mostly on web service brokers limited to service lifecycle management. There is however a need for service brokers taking into account QoS in order to ensure total response time of composed services, prioritize time-critical services or ensure bandwidth or robustness. In (Nahrstedt and Smith, 1995) a QoS broker model is described for general distributed systems. Contrary to general distributed systems, web services have a dynamic nature in terms of service availability and the clients invoking them. Brokers must support more flexible service selection and be able to adapt to the dynamic server load. In (Yu and Lin, 2004) a web service architecture supporting QoS is presented. However, once the services are selected and the link is established, the client communicates with the server directly without any broker intervention during the actual service process. Due to the dynamic nature of web services, this introduces QoS shortcomings since abrupt failure or unavailability of services needs dynamic selection of another equivalent service. This broker also leaves composition as the client's responsibility. In (Benatallah et al., 2003), a middleware infrastructure for web service composition named Self-Serv is proposed. Like the WSML, it facilitates dynamic web service selection and composition. Self-

[2]The WSML and direct series of Figure 8 are the same as those of Figure 6, but the scale of the Y axis is different in order to allow including the ActiveBPEL series.

serv introduces two concepts: composite services and service containers. Composite services can be compared to the WSML's service compositions, while service containers can be compared to the WSML's service types.

5.2 AOP for Web Services

In (Singh et al., 2005), a software architecture called *Aspect-Oriented Web Services* (AOWS) is presented. It is targeted at describing crosscutting concerns between web services in order to give a more complete description of the web services, supporting richer dynamic discovery and seamless integration. An implementation is made on the .NET platform and all AOWS subsystems and their relationships have been formally modeled. While aiming to achieve similar goals as the WSML, AOWS does not support third-party independent services as services need to be modeled in an AOWSDL language, and registered in a dedicated AOUDDI registry. Clients communicate with AOConnectors, which address the web services through adaptors; if necessary multiple services are bundled in an AOComposite. The aspectual features of the AOWS framework are used to provide more efficient and effective dynamic description, discovery and integration. Similar to the WSML, service related code is extracted from the client, and the client only needs to communicate with the AOConnectors.

Contextual Aspect-Sensitive Services (CASS) (Cottenier and Elrad, 2005) is a distributed aspect platform that targets the encapsulation of coordination, activity life cycle and context propagation concerns in service environments. CASS advocates the decomposition of applications into a set of collaboration layers, next to the well-known class, component or service-based decomposition. In a CASS-enabled service oriented architecture, a collaboration layer captures the protocols all services should implement to fulfill an interaction. CASS aspects factor out the crosscutting concerns that arise when services are combined into distinct collaboration layers.

The WSML employs aspect-oriented principles in order to better modularize the different concerns that make up a service composition. Several approaches have been proposed which apply aspect-oriented principles to BPEL:

AO4BPEL (Charfi and Mezini, 2004) allows specifying crosscutting concerns in separate aspects. Aspects contain an advice written BPEL code, and pointcuts expressed using XPath, which allows expressing queries on the structure of XML documents. This may cause pointcuts to break when the structure of the base BPEL process changes. AO4BPEL does not have a separate aspect deployment construct, which makes it impossible to reuse aspects in other processes without duplicating code. AO4BPEL uses a runtime weaver, which is implemented by extending the BPWS4J engine.

Courbis and Finkelstein (Courbis and Finkelstein, 2005) have proposed an aspect-oriented approach similar to AO4BPEL. They also use XPath as a pointcut language, and propose extending a BPEL engine for supporting dynamic aspect addition and removal. Advices are expressed using Java instead of BPEL. There is no publicly available implementation.

Padus (Braem et al., 2006) is an aspect-oriented approach for BPEL that aims to improve on the two former approaches by providing a declarative, logic-based pointcut language which is more robust when the structure of the BPEL process changes. Aspects are reusable entities which are bound to a concrete process using separate aspect deployments. Padus is implemented as a static weaver which produces standard BPEL processes which can be executed on any standard BPEL engine.

5.3 Relation Between BPEL and the WSML

BPEL and the WSML are complementary approaches: BPEL offers dedicated support for modeling business logic involving multiple web services and allows for the centralized deployment of these service compositions on a dedicated engine. The WSML on the other hand is specifically targeted at dealing with dynamic service environments. In the WSML approach, the actual business process is part of the client, although it is possible to express more complicated business process constructs in the aspects, including more advanced control flow such as conditional executions, loops and parallel branching. Other features such as logging, exception handling, compensation, monitoring and stateful context are supported through additional aspects. *The result will be a collection of aspects working together to realize a composition.* This more modularized approach differs from dedicated composition languages such as BPEL where all concerns are scattered and tangled in one monolithic composition specification.

6 CONCLUSIONS AND FUTURE WORK

This paper describes how the Web Services Management Layer (WSML) can be used to express both pro-active and reactive web service compositions through its dynamic service binding mechanism. We present the results of a number of experiments which evaluate the performance and scalability of the WSML implementation with regard to three brokering strategies: round-robin, broker-side monitoring and server-side monitoring. Although the WSML induces a small performance overhead, this overhead is small enough to be compensated by the various ways in which the WSML relieves web service clients of complex tasks such as redirection, selection and client-side management. This makes the WSML a viable option in real-life web services applications.

Although this paper evaluates the performance and scalability of our approach, future work should be directed at evaluating the expressiveness of its composition mechanism. Existing evaluations of workflow languages (Wohed et al., 2003) are often based on a number of frequently used and required workflow patterns (van der Aalst et al., 2000), and a similar evaluation of our composition mechanism will be investigated. Based on this evaluation, changes to the WSML could be required in order to solve possible shortcomings.

ACKNOWLEDGEMENTS

The research presented in this paper is funded by the Research Foundation — Flanders (Fonds Wetenschappelijk Onderzoek — Vlaanderen) in the context of the DyBroWS project.

REFERENCES

Andrews, T. et al. (2003). Business Process Execution Language for Web Services version 1.1.

Benatallah, B., Sheng, Q. Z., and Dumas, M. (2003). The Self-Serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48.

Braem, M., Verlaenen, K., Joncheere, N., Vanderperren, W., Van Der Straeten, R., Truyen, E., Joosen, W., and Jonckers, V. (2006). Isolating process-level concerns using Padus. *Lecture Notes in Computer Science*, 4102:113–128.

Chakraborty, D. and Joshi, A. (2001). Dynamic service composition: State of the art and research directions. Technical Report TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore, MD, USA.

Charfi, A. and Mezini, M. (2004). Aspect-oriented web service composition with AO4BPEL. *Lecture Notes in Computer Science*, 3250:168–182.

Cottenier, T. and Elrad, T. (2005). Dynamic and decentralized service composition with contextual aspect-sensitive services. In *Proceedings of the 1st International Conference on Web Information Systems and Technologies (WEBIST 2005)*, Miami, FL, USA.

Courbis, C. and Finkelstein, A. (2005). Towards aspect weaving applications. In *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, St. Louis, MO, USA. ACM Press.

Du, W. and Elmagarmid, A. (1997). Workflow management: State of the art vs. state of the products. Technical Report HPL-97-90, Hewlett-Packard Labs, Palo Alto, CA, USA.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwing, J. (1997). Aspect-oriented programming. Technical Report SPL97-008 P9710042, Xerox PARC, Palo Alto, CA, USA.

Nahrstedt, K. and Smith, J. M. (1995). The QoS broker. *IEEE MultiMedia*, 2(1):53–67.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058.

Singh, S., Grundy, J., Hosking, J., and Sun, J. (2005). An architecture for developing aspect-oriented web services. In *Proceedings of the 3rd IEEE European Conference on Web Services (ECOWS 2005)*, Växjö, Sweden. IEEE Computer Society.

Suvée, D. and Vanderperren, W. (2003). JAsCo: An aspect-oriented approach tailored for component based software development. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (AOSD 2003)*, Boston, MA, USA. ACM Press.

van der Aalst, W. M. P., Barros, A. P., ter Hofstede, A. H. M., and Kiepuszewski, B. (2000). Advanced workflow patterns. *Lecture Notes in Computer Science*, 1901:18–29.

Verheecke, B., Cibrán, M. A., and Jonckers, V. (2003). AOP for dynamic configuration and management of web services. *Lecture Notes in Computer Science*, 2853:55–85.

Verheecke, B., Cibrán, M. A., and Jonckers, V. (2004). Aspect-oriented programming for dynamic web service monitoring and selection. *Lecture Notes in Computer Science*, 3250:15–29.

Verheecke, B., Vanderperren, W., and Jonckers, V. (2006). Unraveling crosscutting concerns in web services middleware. *IEEE Software*, 23(1):42–50.

Wohed, P., van der Aalst, W. M. P., Dumas, M., and ter Hofstede, A. H. M. (2003). Analysis of web services composition languages: The case of BPEL4WS. *Lecture Notes in Computer Science*, 2813:200–215.

Yu, T. and Lin, K.-J. (2004). The design of QoS broker algorithms for QoS-capable web services. In *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE 2004)*, Taipei, Taiwan. IEEE Computer Society.