

Web Service Offerings Infrastructure (WSOI) - A Management Infrastructure for XML Web Services

*V. Tasic, W. Ma, B. Pagurek, B. Esfandiari
Dep. of Systems and Computer Engineering, Carleton University, Ottawa, Canada
{vladimir, weima, bernie, babak}@sce.carleton.ca*

Abstract

Our Web Service Offerings Language (WSOL) enables formal specification of important management information--classes of service (modeled as service offerings), various types of constraint (functional, QoS, access rights), and management statements (e.g., prices, penalties, and management responsibilities)--for XML (Extensible Markup Language) Web Services. To demonstrate the usefulness of WSOL for the management of Web Services and their compositions, we have developed a corresponding management infrastructure, the Web Service Offerings Infrastructure (WSOI). WSOI enables monitoring and accounting of WSOL service offerings and their dynamic manipulation. To support monitoring of WSOL service offerings, we have extended the Apache Axis open-source SOAP engine with WSOI-specific modules, data structures, and management ports. To support dynamic manipulation of WSOL service offerings, we have developed appropriate algorithms, protocols, and management port types and built into WSOI modules and data structures for their implementation. Apart from provisioning of WSOL-enabled Web Services, we are using WSOI to perform experiments comparing dynamic manipulation of WSOL service offerings and alternatives.

Keywords

Web Service, management, infrastructure, monitoring, dynamic adaptation.

1. Introduction

An **XML (Extensible Markup Language) Web Service** is “a software application identified by a URI (Uniform Resource Identifier), whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols” [1]. The three main Web Service technologies are the SOAP protocol for XML messaging, the WSDL (Web Service Description Language) language, and the UDDI (Universal Description, Discovery, and Integration) directory. While there has been a lot of recent progress regarding Web Services, a number of management-related issues have not yet been studied completely. This paper presents a management infrastructure that addresses several of these issues.

Let us first define terminology. The Web Service technologies are intended for application-to-application (A2A) and business-to-business (B2B) integration, so their true power is leveraged through **compositions** (orchestrations) of Web Services. By a **consumer** (requester) of a Web Service *A* we assume another Web Service that is composed with *A* and collaborates with it, not a human end user. On the other hand, we refer to *A* as the **provider** (supplier) Web Service. The composed Web Services can be distributed over the Internet, run on different platforms, implemented in different programming languages, and provided by different vendors.

We have noticed the need for enabling Web Services to provide multiple classes of service and to perform management activities, such as monitoring and dynamic (i.e., run-time) manipulation, with them [2, 3]. By a '**class of service**' we mean a discrete variation of the complete service and quality of service (QoS) provided by one Web Service. Classes of service of one Web Service refer to the same WSDL description, but differ in constraints and management statements. For example, they can differ in usage privileges, response times guaranteed to consumers, verbosity of response information, prices, payment models, and/or management entities. Using classes of service is not as powerful as using custom-made service level agreements (SLAs), consumer profiles, or separate Web Services. However, specification and, particularly, management of classes of service is generally simpler, faster, and incurs less run-time overhead than alternatives [2]. As we will illustrate in this paper, it is often easier and faster for a consumer to switch to another class of service of the same Web Service than to search for a replacement Web Service or to renegotiate an SLA.

For the formal specification of classes of service (modeled as service offerings), various types of constraint (functional, QoS, access rights) and management statements (prices, penalties, management responsibilities), we have developed the **Web Service Offerings Language (WSOL)**. WSOL is complementary to WSDL 1.1 - a WSOL file references one or more WSDL files and specifies additional information. To demonstrate monitoring and dynamic manipulation of WSOL service offerings, we have developed the corresponding management infrastructure - the **Web Service Offerings Infrastructure (WSOI)**. WSOI monitoring activities include measurement and calculation of used QoS metrics, evaluation of WSOL constraints, calculation of prices and penalties to be paid, and accounting of executed operations and evaluated constraints. WSOI dynamic manipulation of WSOL service offerings achieves adaptation of a Web Service composition without breaking relationships between provider and consumer Web Services. We have published several papers about WSOL [3, 4, 5] and an overview of our work on dynamic manipulation of classes of service for Web Services [2]. This paper contains additional technical details about WSOI, recent results, and discussions of management issues.

In this section, we introduced the topic of our research. In the next section, we give a brief overview of WSOL and our mechanisms for dynamic manipulation of WSOL service offerings. Then, we summarize the primary and secondary goals and requirements for WSOI in Section 3. In Section 4, we present how WSOI implements monitoring of WSOL service offerings, while in Section 5 we discuss how WSOI implements the mechanisms for dynamic manipulation of WSOL service offerings. Section 6 contains a brief survey of closely related work by other authors. We summarize conclusions and directions for future work in Section 7.

2. Background

2.1 Web Service Offerings Language (WSOL)

The main concept in WSOL [3, 4, 5] is a **service offering (SO)** - a formal representation of one class of service for a Web Service. It can contain formal definitions of constraints, management statements, and/or reusability constructs:

1. Every WSOL **constraint** formally states some condition to be evaluated before and/or after invocation of some operations or periodically, at particular date/time instances. We have defined XML schemas for description of functional constraints (e.g., pre- and post-conditions), quality of service (QoS) constraints (e.g. about response times or availability), and access rights.
2. A WSOL **statement** is any construct, other than a constraint, that states management information about the represented class of service. We have defined XML schemas for statements for management responsibilities, validity periods, subscription prices, pay-per-use prices, and monetary penalties to be paid if constraints are not met. Using the XML Schema mechanisms, WSOL can be extended with the formal specification of additional types of constraint and management statement.
3. The **reusability constructs** in WSOL enable easier specification of new service offerings, e.g., by using inheritance (extension), inclusion, or template instantiation. They also determine static relationships between WSOL service offerings, which show similarities and differences between service offerings and do not change during run-time.

In addition to service offerings, WSOL files can contain specifications of **service offerings dynamic relationships (SODRs)**. One such relationship states what service offering is an appropriate replacement if particular constraints from the used service offering cannot be met. Since SODRs can change during run-time (e.g., after creation of a new service offering), they are specified outside service offerings to avoid frequent modifications of service offering definitions.

To verify the WSOL syntax, we have developed a WSOL parser [5], which we plan to extend to a full WSOL compiler. This WSOL compiler will be able to automatically generate, without programmer intervention, modules that measure and/or calculate QoS metrics, evaluate WSOL constraints, and perform accounting.

2.2 Mechanisms for dynamic manipulation of WSOL service offerings

Contrary to service level agreements (SLAs), service offerings are, in principle, non-negotiable agreements - the vendor of a provider Web Service defines its service offerings and a consumer chooses one from the offered alternatives. The development of WSOL service offerings can be automated or assisted by different tools, such as tools collecting QoS statistics to define appropriate QoS constraints. In addition, human Web Service administrators can create or modify WSOL service offerings manually. Input from consumers can, but need not, be used for the creation of new service offerings. In many cases, WSOL service offerings are defined statically, before the provider starts serving consumers. As will be mentioned later, there is also a possibility of dynamic creation of new service offerings. The newest version of [3] describes how a consumer chooses between service offerings of the same provider.

In addition, we have studied [2] and implemented in WSOI several dynamic manipulation mechanisms for run-time adaptation of which service offerings a provider supports and a consumer uses. The five main mechanisms are switching, deactivation, reactivation, deletion, and creation of service offerings. These mechanisms can be used between operation invocations inside one session. The crucial WSOL language support for the manipulation of service offerings is the specification of various relationships, both service offerings dynamic relationships and static relationships determined by WSOL reusability constructs.

Dynamic switching between service offerings means changing which service offering a consumer uses. It is initiated by the consumer or by the provider. WSOL service offerings dynamic relationships are used to choose the replacement service offering. The consumer can initiate switching to dynamically adapt the service and/or QoS it receives without searching for another provider. The provider can initiate switching to gracefully upgrade or degrade its service and/or QoS in case of changes. In this case, the consumer is asked for confirmation, either before or after the switching (as determined by an attribute of the used service offering). This means that a consumer can reject the replacement service offering suggested by the provider. We have also developed solutions for complex switching scenarios and for handling of special cases. Switching is the basic adaptation mechanism in our work.

Deactivation of service offerings is used by a provider Web service when changes in operational circumstances affect what service offerings it can provide. We have developed support for handling consumers using the deactivated offering [2].

The deactivated service offering may be **reactivated** later, after another change of circumstances. After the reactivation, the provider suggests the affected consumers to switch to their original service offerings. This can help in achieving, as much as possible, the originally intended level of service and QoS.

If the probability of future reactivation is zero or very low, the provider Web Service can decide to **dynamically delete a deactivated service offering**.

Dynamic creation of new service offerings can be used after a change in the implementation of the provider Web Service, in the Web Services that the provider uses, in management third parties, in the execution environment, or in consumer needs. Dynamic creation of new service offerings can be non-trivial and incur non-negligible overhead. It cannot be performed arbitrarily due to various possible conflicts. Therefore, we are researching only simple and limited creation of new service offerings as variations of existing service offerings. While we concentrate on provider-initiated creation of service offerings, we also leave the possibility of consumer-initiated creation in special cases.

Other mechanisms related to the manipulation of service offerings--such as deactivation, reactivation, deletion, and creation of service offerings dynamic relationships (SODRs)--can also be studied.

3. Goals and Requirements for WSOI

The WSOL language is not very useful without a management infrastructure that monitors WSOL service offerings. Consequently, the first primary goal for WSOI was to **enable practical use of WSOL** and thus demonstrate that WSOL can be used

for the monitoring and management of Web Services and Web Service compositions.

In addition, we are interested in researching dynamic adaptation of Web Service compositions based on the manipulation of WSOL service offerings without human intervention. Therefore, the second primary goal for WSOI was to **implement appropriate mechanisms for dynamic manipulation of service offerings** and enable experiments with them. These two goals were the main goals for WSOI. We have also identified several additional goals and requirements for WSOI, outline next.

While WSOI empowers the provider Web Service to provision, monitor, and manage service offerings, we also wanted to leave open the possibility that, if needed, humans or management software external to the composed Web Services **can be involved** in the manipulation of WSOL service offerings.

Our vision was that WSOL and WSOI can **accommodate relatively simple provider and consumer Web Services**. We did not assume that Web Services are provided by enterprises that already have complex management frameworks and/or application servers supporting management. One argument against monitoring and management activities is their run-time overhead. Consequently, we researched and built into WSOL and WSOI features with relatively low run-time overhead [4], such as the specification and the manipulation of classes of service instead of SLAs.

WSOL enables specification of management third parties, which perform monitoring independently from the provider and the consumer. WSOL management third parties usually act as SOAP intermediaries, but can also act as probes. Consequently, WSOI had to **support management third parties**.

It is often necessary to group monitoring and management information, e.g., for the measurement or calculation of periodic QoS metrics, the evaluation of periodic QoS constraints, the calculation of subscription prices, and the manipulation of service offerings. One simple way to achieve this was to have WSOI **support sessions**. Note that this does not mean that the actual implementation of business-related WSDL operations of provider Web Services hosted by WSOI must support sessions.

4. Monitoring of WSOL Service Offerings with WSOI

The part of WSOI that performs monitoring of WSOL service offerings is based on extensions of **Apache Axis** (Apache eXtensible Interaction System) version 1.0 [6], a popular open-source SOAP engine implemented in Java. A SOAP engine is an application that receives, processes, and sends SOAP messages. Axis contains the standard functionality of hosting Web Services described in WSDL and processing of their SOAP messages. WSOI adds modules specific to the provisioning, monitoring, and dynamic manipulation of WSOL service offerings. We run Axis using the popular Apache Tomcat open-source application server.

Figure 1 shows how WSOI is used in a provider Web Service. Consumer requests are transported over a network and received by Tomcat, which passes the SOAP content to WSOI. WSOI processes the SOAP message, performs monitoring activities, and invokes appropriate Java code that implements the WSDL operations of the Web Service. The results are returned to WSOI, which processes them and performs additional monitoring activities. Afterwards, Tomcat sends the reply to the consumer.

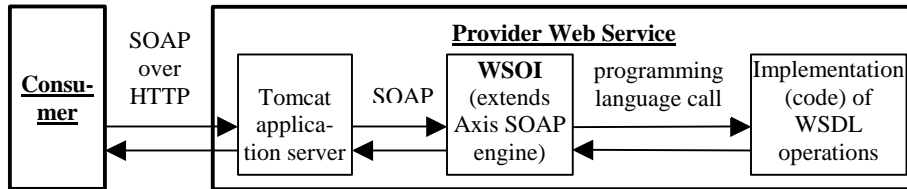


Figure 1: Position of WSOI inside a Provider Web Service

Since it includes Axis, WSOI can also be used for provider Web Services that do not support WSOL. Axis, and thus WSOI, can be used for providers, consumers, and SOAP message intermediaries, such as WSOL management third parties. A scenario of using WSOL and WSOI with management third parties was presented in [3].

Figure 2 shows modules in WSOI. WSOI modules can be categorized into WSOL service offering monitoring modules (in the left half of Figure 2), WSOL service offering manipulation modules (in the right half), and modules used for both purposes (crosscutting both halves). In this section, we present WSOL service offering monitoring modules, while in the next section we will discuss the other modules.

Axis has a modular, flexible, and extensible architecture based on configurable chains of pluggable SOAP message processing components, called handlers. An Axis **handler** can perform message processing, e.g., measurement of QoS metrics or evaluation of constraints. It can also alter the processed SOAP message. An Axis **chain** is an ordered, pipelined collection of handlers. It can also be treated as a handler. Handlers exchange information through an instance of the Axis' **MessageContext** class, which contains information about the request message, the response message, and a bag of properties. Handlers can use the MessageContext properties for decisions related to message processing and can modify these properties.

WSOI adds specialized Axis handlers performing WSOL-related measurement and calculation of QoS metrics, evaluation of constraints, calculation of prices and penalties to be paid, accounting activities. Hereafter, we refer to these handlers and their chains as '**WSOI-specific handlers and chains**'. Some design decisions for WSOI-specific handlers were discussed in [2]. As discussed in [3], there is a correspondence between WSOL constructs and the management activities performed in WSOI-specific handlers. While a WSOL compiler will be able to generate WSOI-specific handlers automatically from WSOL files, we have manually implemented some of these handlers in our WSOI prototype.

In different contexts, different WSOI-specific handlers are used. A **context** is determined by the full name of the invoked operation (containing Web Service, port, and operation names), the name of the service offering, and the name of the management party in which this handler is used. We have developed a special XML file format, called Web Service Offering Descriptor (WSOD), for describing the order of WSOI-specific handlers used in a particular context. WSOD files can be generated by a WSOL compiler or human Web Service administrators. The **WSODEngine** module inside WSOI loads this ordering information from WSOD files into the **contextHandlersTable hashtable**. Using this hashtable, an instance of the **WSOIChain** class--the crucial WSOI module for the monitoring of service offerings--dynamically

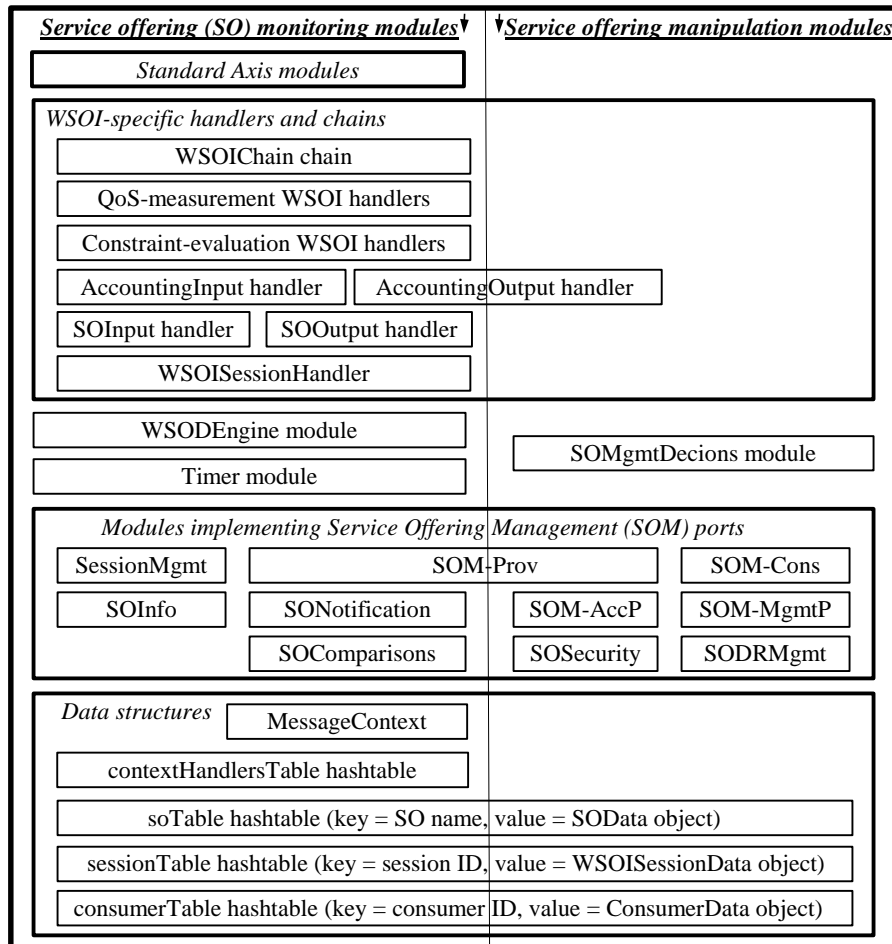


Figure 2: Modules in the Web Service Offerings Infrastructure (WSOI)

constructs the chain of appropriate WSOI-specific handlers for the given context.

Several WSOI-specific handlers are used by all management parties. For efficiency reasons, they are implemented outside the WSOIChain chain. One of them is WSOISessionHandler (SH) that performs WSOL-related session management activities. (Note that the actual implementation of business-related WSDL operations of a Web Service need not support sessions, although it may. In our work, sessions are opened and closed using special operations--presented in Section 5--that are built into WSOI and provided independently from the hosted Web Services.) Further, the WSOI-specific handlers ServiceOfferingInput (SOI) and ServiceOfferingOutput (SOO) exchange the management information between MessageContext properties and SOAP headers. While MessageContext properties are used to transport WSOL-related management information between WSOI modules, SOAP headers are used to transport this information between management parties.

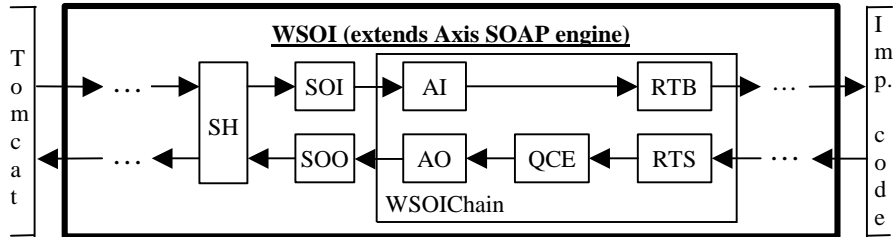


Figure 3: An Example Configuration of Handlers inside Provider-side WSOI

Figure 3 shows an example configuration of WSOI-specific handlers inside a provider-side WSOI. In this example, the provider Web Service measures response time in handlers ResponseTimeBegin (RTB) and ResponseTimeStop (RTS), evaluates a QoS constraint limiting this response time in the handler QoSConstraintEvaluation (QCE), and performs accounting in handlers AccountingInput (AI) and AccountingOutput (AO). Standard Axis handlers used for all Web Services are not shown ('...' indicates their position). A detailed explanation of the example is given in [3].

The measurement or calculation of periodic QoS metrics and evaluation of periodic constraints differs from the example in Figure 3. It is initiated by **Timer**, a special active WSOI-specific module. Timer invokes WSOIChain, which creates and executes a chain of WSOI-specific handlers. The results can be stored locally and/or reported to other management parties in a special notification message.

We have performed a number of **experiments** to demonstrate that WSOI can be used for monitoring of Web Services and to estimate the overhead that such monitoring places on Web Services. The experiments compare average response time and average Java Virtual Machine (JVM) memory usage when Web Services are hosted with Axis and when they are hosted with WSOI. Table 1 shows results of one representative experiment when the consumer and the provider (a simple stock notification Web Service) executed on different computers in a local network. When Axis was used, no WSOL constraint was evaluated. When WSOI was used, the provider evaluated a pre-condition, a post-condition, and a response time QoS constraint. The presented values are results of averaging 1000 measurements. In our opinion, the 15% increase in response time and the 4% increase in memory usage are acceptable. When Web Services are distributed over the Internet, the network delay increases and the relative WSOI overhead on the total response time decreases.

Table 1: A Comparison of Average Response Time and Average Provider-side JVM Memory Usage of Axis and WSOI, both running over Tomcat

<i>Measured Value [Units]</i>	<i>Axis (= A)</i>	<i>WSOI (= B)</i>	<i>Difference (= B-A)</i>	<i>Relative Difference [%] (= (B-A)/A)</i>
<i>Response time [ms]</i>	140	161	21	15.00 %
<i>JVM memory usage [bytes]</i>	6 135 821	6 397 559	261 738	4.27 %

5. Dynamic Manipulation of WSOL Service Offerings in WSOI

Figure 2 shows that WSOI implements our mechanisms for dynamic manipulation of WSOL service offerings with:

1. several data structures,
2. the SOMgmtDecisions module, and
3. several modules that implement Service Offering Management (SOM) port types.

Unlike the modules discussed in the previous section, these modules are not based on Apache Axis. We emphasize modules in provider-side WSOI because they are essential for the dynamic manipulation of WSOL service offerings.

Most **data structures** in WSOI are used for both monitoring and manipulation of WSOL service offerings. Inside WSOI, descriptions of WSOL service offerings and service offerings dynamic relationships are stored in instances of the **SOData** class. An SOData instance also stores other information about a service offering, such as whether it is active or deactivated. The **soTable hashtable** stores one SOData instance for every used service offering. Further, provider archives run-time monitoring information in instances of the **WSOISessionData** class because MessageContext properties only store the monitoring information for the latest invocation. Particularly important are the information about what service offering is used in a particular session and the history information about satisfied and unsatisfied constraints. The **sessionTable hashtable** stores one WSOISessionData instance for every session. The **consumerTable hashtable** stores some information about consumers.

The **SOMgmtDecisions** module in provider-side WSOI implements operations that decide whether, what, how, and when the manipulation of service offerings should be performed. These operations use the data structures discussed above. For example, when the AccountingOutput WSOI-specific handler discovers that one or more constraints were not satisfied, it starts a separate thread that invokes the checkSwitch() operation of the SOMgmtDecisions module. This operation compares the WSOISessionData history of unsatisfied constraints in the given session and the SOData descriptions of service offerings dynamic relationships for the given service offering. If this operation finds an appropriate replacement service offering, the protocol for provider-initiated switching between service offerings is started.

To achieve monitoring of WSOL service offerings and particularly their manipulation, it is necessary to coordinate the involved parties. We have developed appropriate protocols to govern this coordination. The operations that participate in these protocols, as well as other externally-accessible operations related to WSOL service offerings are grouped into several **Service Offerings Management (SOM) port types**. Figure 2 shows whether a SOM port type is used for monitoring of service offerings, their manipulation, or both; while Figure 4 shows example operations from all SOM port types. Operations from SessionMgmt manage WSOI sessions, operations from SOInfo provide information about available service offerings and their activity, while operations from SOComparisons determine static relationships between service offerings. (Operations from SOInfo and SOComparisons can be implemented by providers and/or by Web Service brokers.) Operations from SOM-Prov (implemented by providers), SOM-AccP (accounting parties), SOM-Cons (consumers), and SOM-MgmtP (all management parties) are used in monitoring and, particularly, ma-

SessionMgmt	SOM-Prov	SOM-Cons	SONotification
openSession() closeSession()	startWithSO() switchSO() prInitSwitchSO() deactivateSO() reactivateSO() createSO() deleteSO()	switchInProgress() switchSuggested() switchCancelled() switchingTo() soDeactivated() soReactivated() newSOCreated() soDeleted()	inform() readValue()
SOInfo	SOM-AccP	SOM-MgmtP	SODRMgmt
listSOsForMe() descCurrentSO() listActiveSOs() listAllSOs()	switchSO() switchInProgress() switchCancelled() switchingTo() forwardRequests() readBalance() readHistory()	assignSO() initializeWork() finalizeWork() initAndFinal() listSOMOps() listSOMOpsForMe()	listSODRsForMe() listAllSODRs() listActiveSODRs() deactivateSODR() reactivateSODR() createSODR() deleteSODR()
SOComparisons			SOSecurity
isExtension() listExtensions() doesInstantiate() doesInclude() compare()			allowSO() disallowSO()

Figure 4: Example Operations in Service Offering Management (SOM) Port Types

nipulation of WSOL service offerings. Operations from SONotification exchange WSOL-related management information between management parties, operations from SODRMgmt enable use and manipulation of service offerings dynamic relationships, while operations from SOSecurity manage security of service offerings. Further details can be found in [7]. Note that WSOI for a particular management party need not implement all SOM port types or all operations in one port type.

We performed a number of analytical studies and practical experiments comparing the manipulation of service offerings with alternative approaches to dynamic adaptation of Web Service compositions. The main alternative is ‘re-composition of Web Services’, which breaks a current Web Service composition and creates a new composition, possibly with Web Services that have to be found. A special simple case is ‘switching between (provider) Web Services’ when a consumer simply searches for and chooses another provider Web Service. Another alternative is ‘re-negotiation of SLAs’ when a provider and its consumer negotiate a new custom-made SLA.

Our **analytical studies** are based on comparisons of the number of exchanged SOAP messages. On the other hand, in our **experiments** we measure average delay and average Java Virtual Machine (JVM) memory usage. Our prototype implementation of WSOI was crucial for conducting these experiments.

Let us describe one representative experiment comparing consumer-initiated switching between Web Services and switching between WSOL service offerings. For switching between service offerings, only one provider (a simple stock notification Web Service) was used, while for switching between Web Services two such providers were used. The consumer and the providers executed on the same computer. For simplicity, the information about the appropriate replacement Web Service or replacement service offering was hardcoded into consumer’s implementation.

For switching between Web Services, 4 SOAP messages had to be exchanged: closeSession() and its reply, and openSession() and its reply. The average delay was

28 ms. On the other hand, for switching between service offerings 2 SOAP messages were enough: `switchSO()` and its reply. Consequently, the average delay in this case was 13 ms, which is about 54% less than for switching between Web Services.

The JVM memory usage was calculated as a sum of memory used on consumer side and on provider side. For switching between Web Services, the provider infrastructure contained Tomcat, standard Axis provider-side modules, only session management modules from WSOI, and implementation code. While the two providers in this case used different Axis instances (which were counted twice), they used the same Tomcat instance and the Tomcat memory overhead was counted only once. For switching between WSOL service offerings, the provider infrastructure contained Tomcat, standard Axis provider-side modules, all relevant WSOI modules, and implementation code. In both cases, consumer infrastructure contained standard Axis consumer-side modules, `WSOISessionHandler`, and simple implementation code. The average total JVM memory usage for switching between Web Services was about 8.84 Megabytes. For switching between WSOL service offerings, the average total JVM memory usage was about 6.89 Megabytes, which is about 16% less.

In a similar experiment, provider-initiated switching between service offerings was about 15% faster and consumed about 17% less memory than provider-initiated switching between Web Services. Details about both mentioned experiments are given in [7]. Note that in both experiments switching between Web Services is relatively simple and straightforward. When consumer and provider execute on separate computers and/or switching between Web Services requires searching a UDDI directory, the advantages of the manipulation of service offerings are greater.

These practical experiments and analytical studies support our initial observation that manipulation of service offerings is generally **simpler, faster, and incurs less run-time overhead** than the re-composition of Web Services and the re-negotiation of SLAs. However, compared to the re-composition of Web Services, manipulation of service offerings has limitations because service offerings differ only in constraints and management statements and because appropriate service offerings cannot always be found or created. Therefore, we advocate the manipulation of service offerings as a complement to, but not a complete replacement for, the re-composition of Web Services and other alternatives. Further details about our analytical studies, experiments, and conclusions can be found in [2].

6. Related Work

Our work on WSOL draws from previous work on differentiated classes of service and formal representation of constraints in other areas. While at the beginning of our research there was no work of this kind for Web Services, several related works appeared in the meantime.

The most important are two languages for the formal, XML-based, specification of custom-made SLAs for Web Service: the IBM's **Web Service Level Agreements (WSLA)** [8, 9] and the HP's **Web Service Management Language (WSML)** [10, 11]. SLAs in these two languages contain QoS constraints and management information, e.g., prices. Both WSLA and WSML are oriented towards management applications in inter-enterprise scenarios and are accompanied by appropriate management

infrastructures. While WSLA and WSML are more powerful in some aspects, WSOL also has advantages, such as support for classes of service and their dynamic manipulation, specification of different constraints and management statements, broader set of reusability constructs, relative simplicity, and features with lower runtime overhead [4]. Recently, several works related to the specification of policies or QoS for Web Services have appeared, such as WS-Policy [12], DAML-S [13], and UX [14]. However, these works are not yet accompanied by research of management-related issues and appropriate management infrastructures.

On the other hand, products of several companies--such as Hewlett-Packard, Talking Blocks, Flamenco Networks, and Actional--perform some management, often performance management, of Web Services and/or Web Service compositions. Several recent papers also concentrate on particular management functional areas, such as security management, for Web Services. WS-Manageability [15] and WS-Agreement (a.k.a. OGSi-Agreement) [16] are two examples of new standardization initiatives that tackle some aspects of the management of Web Services.

An important distinction between our research and these products, papers, and standardization initiatives is that our work is focused on the specification, monitoring, and dynamic manipulation of classes of service for Web Services. We are not aware of commercial products or academic works addressing these issues.

7. Conclusions and Future Work

The work on the Web Service Offerings Infrastructure (WSOI) is closely related to our work on the Web Service Offerings Language (WSOL) [3, 4, 5] and the mechanisms for dynamic manipulation of WSOL service offerings [2]. WSOL and WSOI enable specification, monitoring, and manipulation of classes of service (i.e., service offerings) for Web Services to the extent that is not provided by related works.

WSOL enables specification of important management information and WSOL service offerings can be viewed as simple SLAs or contracts between Web Services. Compared to recent related languages, such as WSLA, WSML, and WS-Policy, our WSOL has several advantages [4], including support for classes of service.

WSOI is the management infrastructure that enables practical use of WSOL. It enables measurement and calculation of QoS metrics, evaluation of WSOL constraints, calculation of prices and penalties to be paid, accounting of executed operations and evaluated constraints, and dynamic manipulation of WSOL service offerings. WSOI demonstrates that WSOL can be used for provisioning and management of Web Services and their compositions. Due to relative simplicity and low overhead of WSOL, WSOI is simpler and with less run-time overhead than management infrastructures for WSLA and WSML. WSOI extends Apache Axis, a SOAP engine used by many relatively simple Web Services. Our experiments showed that the additional overhead of WSOI is acceptable, so Web Services that now use Axis can use WSOI.

We have integrated into WSOI, on top of Axis, original solutions for monitoring of WSOL service offerings. For monitoring and accounting activities, we have developed WSOI-specific handlers, the Timer module, modules implementing operations from Service Offering Management (SOM) port types, as well as appropriate data structures. To support sessions in WSOI, we have built in the WSOISessionHandler

and the SessionMgmt port. To support management third parties, we have designed the format for management information transfer in SOAP headers and built into WSOI the ServiceOfferingInput (SOI) and ServiceOfferingOutput (SOO) handlers.

Our work on the mechanisms for dynamic manipulation of WSOL service offerings is completely original. We have analyzed these mechanisms, developed appropriate algorithms and protocols, integrated into WSOI appropriate modules and data structures, and performed a number of experiments with these mechanisms and their alternatives. If needed, external management software or human administrators can access these mechanisms through operations of SOM ports. The issue of dynamic manipulation of classes of service for Web Services is not researched in related works, and our use of WSOI prototype as experimental tool and environment lead us to demonstrations of usefulness, benefits (speed, simplicity, low run-time overhead), and limits of dynamic manipulation of WSOL service offerings.

While we have designed and partially implemented the main parts of WSOI, some parts are not yet fully implemented. For example, we currently have only rudimentary support for creation of service offerings. Further, we have not integrated support for manipulation of service offerings dynamic relationships and for security. While our current WSOI prototype demonstrates the main concepts, the improved prototype will demonstrate the complete system. We also want to more precisely determine benefits, usability, and limits of dynamic manipulation of WSOL service offerings. In this respect, we are currently conducting additional analytical studies and experiments with the manipulation of service offerings, and WSOI in general. We plan experiments comparing WSOL and languages using custom-made SLAs.

The WSOL language is relatively complete and stable, but we also have some items for future work in this area [3]. We plan to make WSOL compatible with WSDL version 2.0. The major WSOL issue related to WSOI is the full implementation of a WSOL compiler to enable automatic generation of WSOI-specific handlers from WSOL files. A Java API for the generation of WSOL files would be beneficial.

While we have achieved significant results on the specification, monitoring, and dynamic manipulation of classes of service for Web Services, there are many other issues for future work in the area of management of Web Services and Web Service compositions. Our results from the work on WSOL, WSOI, and the mechanisms for manipulation of WSOL service offerings can be integrated into future Web Service management standards and platforms.

References

- [1]. Schlimmer, J.C. (ed.): Web Services Description Requirements. World Wide Web Consortium (W3C) working draft. Oct. 28, 2002. On-line at: <http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028/> (2002)
- [2]. Tasic, V., Ma, W., Pagurek, B., Esfandiari, B.: On the Dynamic Manipulation of Classes of Service for XML Web Services. In Proc. of the 10th Hewlett-Packard Open View University Association (HP-OVUA) Workshop (Geneva, Switzerland, July 2003). Hewlett-Packard (2003)
- [3]. Tasic, V., Pagurek, B., Patel, B., Esfandiari, B., Ma, W.: Management Applications of the Web Service Offerings Language (WSOL). Submitted to a journal.

- Early version in: Proc. of CAiSE'03 (Velden, Austria, June 2003). Lecture Notes in Computer Science (LNCS), No. 2681. Springer-Verlag (2003) 468-484
- [4]. Tomic, V., Patel, K., Pagurek, B.: WSOL - A Language for the Formal Specification of Classes of Service for Web Services. Proc. of ICWS'03 (Las Vegas, USA, June 2003), CSREA Press (2003) 375-381
 - [5]. Patel, K.: XML Grammar and Parser for the Web Service Offerings Language. M.A.Sc. thesis, Carleton University, Ottawa, Canada. Jan. 30, 2003. On-line at: <http://www.sce.carleton.ca/netmanage/papers/KrutiPatelThesisFinal.pdf> (2003)
 - [6]. The Axis Development Team: Axis Architecture Guide, 1.0 Version. Oct. 7, 2002. The Apache Software Foundation. On-line at: <http://archive.apache.org/dist/ws/axis/1.0/xml-axis-10.zip> (2002)
 - [7]. Tomic, V., Ma, W., Pagurek, B., Esfandiari, B.: Web Service Offerings Infrastructure (WSOI) - A Management Infrastructure for XML Web Services. Research Report SCE-03-19, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, Aug. 16, 2003. On-line at: <http://www.sce.carleton.ca/netmanage/papers/TomicEtAlResRepAug2003.pdf> (2003)
 - [8]. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement (WSLA) Language Specification, Version 1.0, Revision wsla-2003/01/28. International Business Machines Corporation (IBM). On-line at: <http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf> (2003)
 - [9]. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, Vol. 11, No 1 (Mar. 2003) Plenum Publishing (2003)
 - [10]. Sahai, A., Durante, A., Machiraju, V.: Towards Automated SLA Management for Web Services. Research Report HPL-2001-310 (R.1), Hewlett-Packard (HP) Laboratories Palo Alto. July 26, 2002. On-line at: <http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf> (2002)
 - [11]. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA Monitoring for Web Services. Proc. of DSOM 2002 (Montreal, Canada, Oct. 2002). Lecture Notes in Computer Science (LNCS), No. 2506. Springer-Verlag (2002) 28-41
 - [12]. Hondo, M., Kaler, C. (eds.): Web Services Policy Framework (WS-Policy), Version 1.0. Dec. 18, 2002. BEA/IBM/Microsoft/SAP. On-line at: <ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf> (2002)
 - [13]. The DAML Services Coalition: DAML-S: Semantic Markup for Web Services. WWW page for DAML-S version 0.9. May 5, 2003. On-line at: <http://www.daml.org/services/daml-s/0.9/daml-s.html> (2003)
 - [14]. Chen, Z., Lianf-Tien, C., Silverajan, B., Bu-Sung, L.: UX - An Architecture Providing QoS-Aware and Federated Support for UDDI. Proc. of ICWS'03 (Las Vegas, USA, June 2003), CSREA Press (2003) 171-176
 - [15]. Potts, M., Sedukhin, I., Kreger, H., Stokes, E.: Web Service Manageability - Specification (WS-Manageability). V. 1.0. Sept. 2003. Submitted to the OASIS Web Services Distributed Management TC. IBM/CA/Talking Blocks (2003)
 - [16]. Czajkowski, K., Dan, A., Rofrano, J., Tuecke, S., Xu, M.: Agreement-based Grid Service Management (OGSI-Agreement), Version 0. June 12, 2003. Global Grid Forum (2003)