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

Vladimir Tosic, Wei Ma,
Bernard Pagurek, Babak Esfandiari

Research Report SCE-03-19
August 16, 2003

The Department of Systems and
Computer Engineering,
Carleton University, Ottawa, Canada
August 16, 2003

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

Vladimir Tosic, Wei Ma, Bernard Pagurek, Babak Esfandiari

Department of Systems and Computer Engineering, Carleton University,
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada
Tel.:+1(613)520-2600x3548; Fax:+1(613)520-5727
{vladimir, weima, bernie, babak} @ sce.carleton.ca

Abstract. While the recent technologies for XML (Extensible Markup Language) Web Services are an important step towards the goal of application-to-application (A2A) and business-to-business (B2B) integration, they do not address all management-related issues. 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 Web Services. To demonstrate the usefulness of WSOL for the management of XML 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 XML Web Services, we are using WSOI to perform experiments comparing dynamic manipulation of WSOL service offerings and alternatives.

1 Introduction and Motivation

Technologies for XML (Extensible Markup Language) Web Services attempt to address the problem of application-to-application (A2A) and business-to-business (B2B) integration using a set of standards based on XML. 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 XML Web Services, a number of management-related issues have not yet been studied completely. This paper presents a management infrastructure that explores and addresses several of these issues.

An **XML 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]. Hereafter, we use the term 'Web Service' as a synonym for the term 'XML Web Service'. Since Web Service technologies are intended for A2A and B2B integration, 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.

When SOAP, WSDL, and UDDI were first published, we 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. 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, service priorities, 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]. For example, it is often much easier and faster for a consumer to switch to another service offering of the same Web Service than to search for a replacement Web Service or to renegotiate an SLA. We will discuss switching and other mechanisms for dynamic manipulation of service offerings later in the paper.

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 compatible with and complementary to WSDL 1.1. 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, 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. While we have published several papers about WSOL [3, 4, 5], the main topic of this paper is WSOI.

In this section, we introduced the topic of our research. In the next section, we give a brief overview of related work, both our work on WSOL and the work of other authors. Then, we summarize the primary and secondary goals and requirements for

WSOI in Section 3. In Section 4, we present how WSOI implements the monitoring of WSOL service offerings. In Section 5, we give an overview of mechanisms for dynamic manipulation of WSOL service offerings, discuss how WSOI implements them, and explain experiments in which we use WSOI to research these mechanisms. We summarize conclusions and directions for future work in Section 6.

```
<wsol:serviceOffering name = ″SO2″
 service = ″buyStock: buyStockService ″
 extends = ″tns: SO1″
 accountingParty = ″WSOL-SUPPLIERWS″ >
  <wsol:constraint name = ″QoScons2″
   service = ″WSOL-ANY″
   portOrPortType = ″WSOL-EVERY″ operation = ″WSOL-EVERY″ >
    <expressionSchema:booleanExpression>
     <expressionSchema:arithmeticExpression>
      <expressionSchema:QoSmetric
       metricType = “QoSMetricOntology: ResponseTime ″
       service = ″WSOL-ANY″
       portOrPortType = ″WSOL-ANY″ operation = ″WSOL-ANY″
       measuredBy = ″WSOL_INTERNAL″ />
      </expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticComparator type = ″&lt;″ />
      <expressionSchema:arithmeticExpression >
        <wsol:numberWithUnitConstant>
          <wsol:value>0.3</wsol:value>
          <wsol:unit type = ″QoSMeasOntology: second ″ />
        </wsol:numberWithUnitConstant>
      </expressionSchema:arithmeticExpression>
    </expressionSchema:booleanExpression>
  </wsol:constraint>
  …
  <wsol:price name = “Price1″
   service = “buyStock: buyStockService″
   portOrPortType = “buyStock: buyStockServicePort″
   operation = “buyStock: buySingleStockOperation″ >
    <wsol:numberWithUnitConstant>
      <wsol:value>0.01</wsol:value>
      <wsol:unit type = “currencyOntology: CanadianDollar″ />
    <wsol:numberWithUnitConstant>
  </wsol:price>
  …
  <wsol:managementResponsibility name = ″MangResp1″ >
    <wsol:supplierResponsibility scope = ″tns: AccRght1″ />
    <wsol:consumerResponsibility scope = ″tns: Precond3″ />
    <wsol:independentResponsibility scope = ″tns: QoScons2″
     entity = ″http://www.someThirdParty.com ″ />
  </wsol:managementResponsibility>
</wsol:serviceOffering>
```

Figure 1. Parts of an Example WSOL Service Offering

2 Related Work

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.

Figure 1 shows example parts of the WSOL service offering *SO2* for the *buyStockService* Web Service. *SO2* extends another service offering, *SO1*, and contains the QoS constraint *QoScons2*, the pay-per-use price statement *Price1*, and the management responsibility statement *MangResp1*. The other constraints and statements are left out for brevity.

In addition to service offerings, WSOL files can contain specifications of **service**

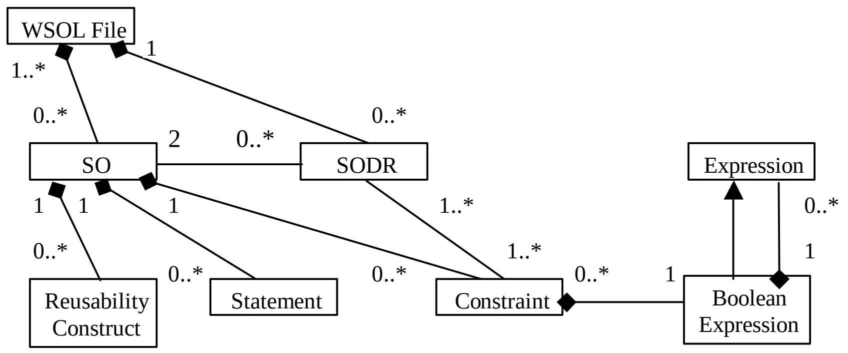


Figure 2. Partial UML Class Diagram for WSOL Concepts

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. Figure 2 shows relationships between the main WSOL concepts.

To verify the WSOL syntax, we have developed a WSOL parser, which we plan to extend to a full WSOL compiler.

2.2 Other Related Works

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 the two recent languages for the formal, XML-based, specification of custom-made SLAs for Web Service: the IBM's **Web Service Level Agreements (WSLA)** [6, 7] the HP's **Web Service Management Language (WSML)** [8, 9]. 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 run-time overhead [4].

Recently, several works related to the specification of policies or QoS for Web Services have appeared, such as WS-Policy [10], DAML-S [11], and UX [12]. However, these works are not yet accompanied by research of management-related issues and appropriate management infrastructures.

On the other hand, several companies—such as HP, Talking Blocks, Flamenco Networks, and Actional—have products performing 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 [13], for Web Services. An important distinction between our research and these products and papers 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.

3 Goals and Requirements for the Web Service Offerings Infrastructure (WSOI)

3.1 Primary Goals and Requirements

The WSOL language is not very useful without a management infrastructure that monitors WSOL service offering. Consequently, the first 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 were interested in researching dynamic adaptation of Web Service compositions based on the manipulation of WSOL service offerings without human intervention. Therefore, another essential goal for WSOI was to **implement appropriate mechanisms for dynamic manipulation of service offerings** and enable experiments with them.

3.2 Secondary Goals and Requirements

Related to the latter primary goal, one of our secondary goals was to leave open the possibility that, if needed, humans or external software managing Web Service compositions **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 who 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**, such as specification and manipulation of classes of service.

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**.

Finally, we intended that a WSOL compiler would be able to **automatically generate**, without programmer intervention, WSOI modules that measure and/or calculate QoS metrics, evaluate WSOL constraints, and perform accounting.

4 Monitoring of WSOL Service Offerings

Figure 3 shows modules in WSOI. WSOI modules can be categorized into WSOL service offering monitoring modules, WSOL service offering manipulation modules, and modules used for both activities. In this section, we discuss how WSOI implements monitoring activities, while in the next section we will discuss manipulation of WSOL service offerings.

The part of WSOI that performs monitoring of WSOL service offerings is based on extensions of **Apache Axis** (Apache eXtensible Interaction System) [14], a popular open-source SOAP engine implemented in Java. A SOAP engine is an applica-

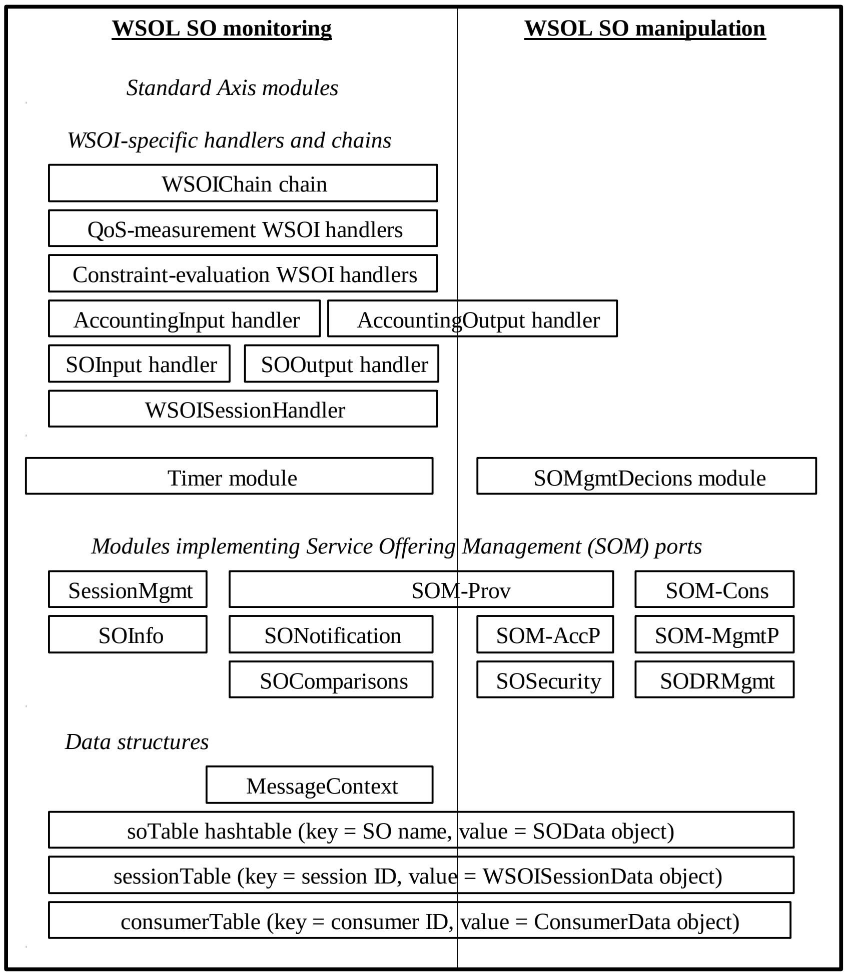
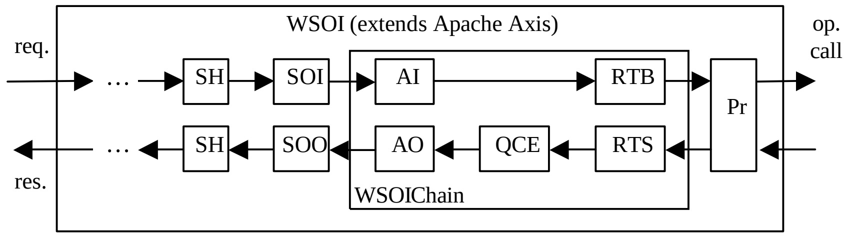

WSOL SO monitoring
WSOL SO manipulation
Standard Axis modules
WSOI-specific handlers and chains
WSOIChain chain
QoS-measurement WSOI handlers
Constraint-evaluation WSOI handlers
AccountingInput handler
AccountingOutput handler
SOInput handler
SOOutput handler
WSOISessionHandler
Timer module
SOMgmtDecions module
Modules implementing Service Offering Management (SOM) ports
SessionMgmt
SOM-Prov
SOM-Cons
SOInfo
SONotification
SOM-AccP
SOM-MgmtP
SOComparisons
SOSecurity
SODRMgmt
Data structures
MessageContext
soTable hashtable (key = SO name, value = SOData object)
sessionTable (key = session ID, value = WSOISessionData object)
consumerTable (key = consumer ID, value = ConsumerData object)

Figure 3. Modules in the Web Service Offerings Infrastructure (WSOI)

tion that receives, processes, and sends SOAP messages. We run Axis using the popular Apache Tomcat open-source application server.

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, e.g., add/remove headers. An Axis **chain** is an ordered, pipelined collection of handlers. Every Axis chain can also be treated as a handler. Axis handlers exchange information through an instance of the **MessageContext** class, which contains information about the request message, the response message, and a bag of properties. Axis handlers can use the MessageContext properties for decisions related to message proc-



Legend:

req.	request (input) message
res.	response (output) message
…	standard Axis modules, executed for all Web Services
SH	WSOISessionHandler, performs session management activities
SOI	SOInput, reads WSOL information from SOAP headers and writes it into appropriate message context properties
SOO	SOOutput, reads WSOL information from message context properties and writes it into SOAP headers
WSOIChain	WSOIChain, examines the context of an operation invocation and description of the current service offering and dynamically constructs the chain of appropriate WSOI-specific handlers
AI	AccountingInput, records the request message
AO	AccountingOutput, uses the information from the message context to calculate prices and eventual penalties to be paid
QCE	QoSConstraintEvaluation, evaluates the QoS constraint limiting response time, stores result into the message context
RTB	RequestTimeBegin, stores into the message context the start time for measuring response time
RTS	RequestTimeStop, stores into the message context the stop time for measuring response time and the difference between this stop time and the start time stored by RequestTimeBegin
Pr	Provider, standard Axis module that dispatches the call to the Java object implementing the requested operation
op. call	operation call, i.e., the Java language call to the implementation of the Web Service's operation

Figure 4. An Example of Message Processing in Provider-side WSOI, based on Axis

essing and can modify these properties. Axis can be used for providers, consumers, and SOAP message intermediaries, such as WSOL management third parties.

In WSOI, specialized Axis handlers perform WSOL-related measurement and calculation of QoS metrics, evaluation of constraints, and accounting activities. Hereafter, we refer to these handlers and their chains as '**WSOI-specific handlers and chains**' (see Figure 3). Some design decisions for WSOI-specific handlers were discussed in [2]. While a WSOL compiler will be able to generate WSOI-specific

Table 1a. Description of an Experiment Comparing Average Response Time and Average Provider-side JVM Memory Usage of Apache Axis and WSOI, both Running over Apache Tomcat

Description	Axis	WSOI
Number of Web Services	2 (consumer and provider)	2 (consumer and provider)
Distribution	Different computers in a local network	Different computers in a local network
Provider Web Services	Simple stock notification Web Service	Simple stock notification Web Service
Number of evaluated constraints	0	3 (1 pre-condition, 1 post-condition, 1 QoS constraint)
Number of exchanged SOAP messages	2 (1 request and 1 reply)	2 (1 request and 1 reply)
Start time for measuring response time	Consumer sends the SOAP request message	Consumer sends the SOAP request message
Stop time for measuring response time	Consumer receives the SOAP reply message	Consumer receives the SOAP reply message
How was the average response time calculated?	1000 tests were run, then the average was calculated	1000 tests were run, then the average was calculated
Software participating in provider-side JVM memory usage	Tomcat, standard provider-side Axis modules, Java implementation of the provider	Tomcat, standard provider-side Axis modules, WSOISessionHandler, other WSOI-specific modules, Java implementation of the provider
How was the average JVM memory usage calculated?	For 1000 continuous test runs, JVM memory usage was periodically measured every 20 ms, then the average was calculated	For 1000 continuous test runs, JVM memory usage was periodically measured every 20 ms, then the average was calculated

Table 1b. Results of the Experiment Comparing Average Response Time and Average Provider-side JVM Memory Usage of Apache Axis and WSOI, Both Running over Apache Tomcat

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

Machine (JVM) memory usage. Table 1a describes one such experiment, while Table 1b shows its results. 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.

5 Dynamic Manipulation of WSOL Service Offerings

5.1 Overview of Mechanisms for Dynamic Manipulation of Service Offerings

The five main dynamic manipulation mechanisms we have studied [2] and implemented in WSOI are:

1. switching,
2. deactivation,
3. reactivation,
4. deletion, and
5. creation of service offerings.

These mechanisms can be used between operation invocations inside one session.

Dynamic switching between service offerings means changing which service offering a consumer uses. Either a consumer or a provider can initiate it. In the latter case, the consumer is asked for confirmation. 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. Switching between service offerings is the basic mechanism in our research. Figure 5 shows a UML (Unified Modeling Lan-

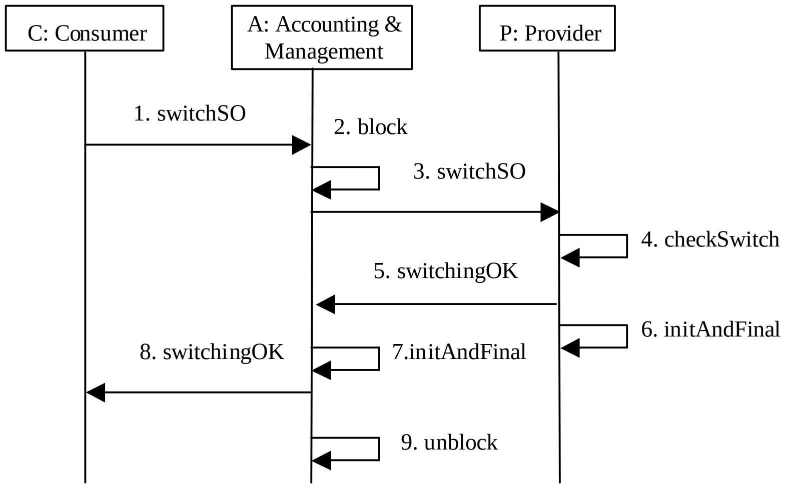


Figure 5. A UML Sequence Diagram of Consumer-initiated Switching of Service Offerings

guage) sequence diagram for simple consumer-initiated switching. Table 2 explains the sequence steps from Figure 5. We have also developed solutions for more complex scenarios and for handling of special cases.

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** at a later time 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

Table 2. Sequence Steps for Consumer-initiated Switching of Service Offerings

No.	Sender->Recipient : Message	Explanation
-	-	Before this sequence, *P* provides at least two active service offerings, *SO1* and *SO2*, that *C* may use. *C* uses *SO1*, but wants to use *SO2*. *A* performs all monitoring and accounting activities for *SO1* and *SO2*
1.	C->A : switchSO	*C* sends *A* the name of *SO2*
2.	A->A : block	*A* blocks and queues new requests from *C* to *P* (*A* uses *SO1* to finish processing of requests received before this message)
3.	A->P : switchSO	*A* sends *P* the name of *SO2*
4.	P->P : checkSwitch	*P* checks whether switching is possible, e.g., whether *SO2* exists, is active, and *C* may use it
5.	P->A : switchingOK	*P* informs *A* that switching is possible
6.	P->P : initAndFinal	*P* initializes its data structures and activities related to *SO2* and finalizes its data structures and activities related to *SO1*
7.	A->A : initAndFinal	*A* initializes its data structures and activities related to *SO2* and finalizes its data structures and activities related to *SO1*
8.	A->C : switchingOK	*A* informs *C* that switching is completed
9.	A->A : unblock	*A* unblocks queued requests from *C* to *P* (*A* processes these requests using *SO2*)
-	-	After this sequence, *C* uses *SO2*

Table 3. Explanation of Service Offering Management (SOM) Port Types

Port Type Name	Implemented by	Explanation
SessionMgmt	Providers	Operations for session management, e.g., opening or closing sessions between the provider and its consumers
SOInfo	Providers	Operations about available service offerings and their activity, e.g., operations informing a consumer about service offerings it may use
SOComparisons	Providers	Operations for determining static relationships between service offerings (e.g., extension); used during selection of Web Services and service offerings
SOM-Prov	Providers	Provider-specific operations for monitoring and manipulation of service offerings
SOM-AccP	Accounting parties	Operations enabling an accounting party to participate in monitoring and manipulation of service offerings
SOM-Cons	Consumers	Consumer-specific operations for monitoring and manipulation of service offerings
SOM-MgmtP	All management parties	Operations that all management parties (providers, accounting parties, management third parties, and consumers) implement to participate in monitoring and management of WSOL service offerings
SONotification	All management parties	Operations used to exchange WSOL-related management information, e.g., to pull or push management information
SODRMgmt	Providers	Provider-side operations for use and manipulation of service offerings dynamic relationships
SOSecurity	Providers	Operations for security management related to service offerings, e.g., allowing or disallowing consumers to use service offerings; not yet implemented in WSOI

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.

5.2 WSOI Implementation of the Manipulation of Service Offerings

WSOI implements the above mechanisms with several different modules (see Fig. 3):
1. several data structures,
2. the SOMgmtDecisions module, and
3. several modules that implement Service Offering Management (SOM) port types.

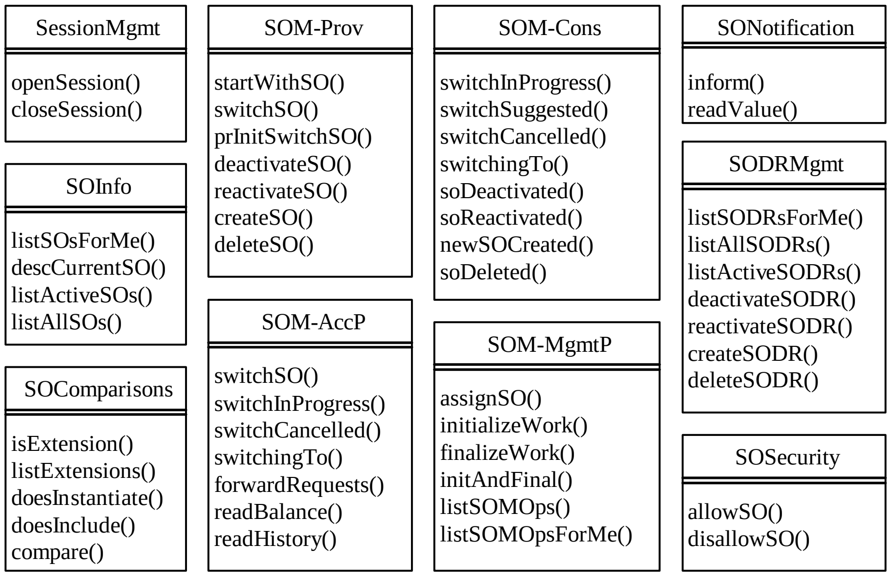


Figure 6. Example Operations in Service Offering Management (SOM) Port Types

Unlike the modules discussed in Section 4, 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. The crucial WSOL language support for manipulation of service offerings is the specification of various relationships, both service offerings dynamic relationships (SODRs) and static relationships determined by WSOL reusability constructs. 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 **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 (e.g., see Figure 5). 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**. Table 3 explains these port types, while Figure 3 shows whether

Table 4. Explanation of Representative Operations in SOM Port Types

Port Type Name	Operation Name	Explanation
SessionMgmt	openSession	Opens a session between the provider implementing this operation and the consumer invoking it; returns the session ID (identity) number
SOInfo	listSOsForMe	Returns a WSOL file describing all WSOL service offerings that the consumer can use
SOComparisons	listExtensions	Returns names of all available WSOL service offerings that are extensions of the service offering whose name is provided as parameter in the operation invocation
SOM-Prov	startWithSO	Used by a consumer at a beginning of a session to select a service offering; assigns a service offering to the current session
SOM-AccP	forwardRequests	Used during switching of service offerings when the old and the new service offering have different accounting parties; forwards to the new accounting party all consumer requests queued at the old accounting party
SOM-Cons	switchSuggested	Used during provider-initiated switching of service offerings; invoked by the provider to suggest a replacement service offering to the consumer; the consumer can accept the suggested service offering, suggest another replacement service offering, or close the session
SOM-MgmtP	listSOMOps	Returns a list of all SOM operations that the management party implements
SONotification	inform	Used to push WSOL-related management information to the management party that implements this operation
SODRMgmt	deactivateSODR	Deactivates the service offerings dynamic relationship the name of which is supplied as the operation parameter
SOSecurity	allowSO	Gives a consumer the right to use a service offering; consumer name and service offering name are provided as parameters; invoked by Web Service management entities

a SOM port type is used for monitoring of service offerings, their manipulation, or both. Further, Figure 6 shows example operations from all SOM port types, while Table 4 explains some of these operations. Further information can be found in [2]. Note that WSOI for a particular management party need not implement all these ports or all operations in one port.

Table 5a. Description of an Experiment Comparing Consumer-initiated Switching between Web Services and Consumer-initiated Switching between WSOL Service Offerings

Description	Switching of Web Services	Switching of Service Offerings
Number of Web Services	3 (consumer and 2 providers)	2 (consumer and provider)
Distribution	Same computer	Same computer
Provider Web Services	Simple stock notification Web Service (both providers)	Simple stock notification Web Service
Infrastructure for providers	Tomcat, 2 * standard provider-side Axis modules (2 Axis engines – one per provider), 2 * WSOISessionHandler	Tomcat, standard provider-side Axis modules, WSOISessionHandler, WSOI-specific modules
Infrastructure for consumers	Standard consumer-side Axis modules, WSOISessionHandler	Standard consumer-side Axis modules, WSOISessionHandler
How the consumer finds the replacement WS or SO	Hardcoded into consumer's implementation	Hardcoded into consumer's implementation
Number of exchanged SOAP messages	4 (closeSession and its reply, openSession and its reply)	2 (switchSO and its reply)
Start time for measuring delay	The consumer sends to the old provider closeSession message	The consumer sends to the provider the switchSO message
Stop time for measuring delay	The consumer receives from the new provider the reply for the openSession message	The consumer receives from the provider the reply for the switchSO message
Software participating in JVM memory usage	Tomcat, 2 * standard provider-side Axis modules, standard consumer-side Axis modules, 3 * WSOISessionHandler, implementation of the consumer and both providers	Tomcat, standard provider-side Axis modules, standard consumer-side Axis modules, 2 * WSOISessionHandler, other WSOI-specific modules, implementation of the consumer and the provider

Table 5b. Results of an Experiment Comparing Consumer-initiated Switching between Web Services and Consumer-initiated Switching between WSOL Service Offerings

Measured Value [Units]	Switching between Web Services (= A)	Switching between Service Offerings (= B)	Difference (= B-A)	Relative Difference (= (B-A)/A) [%]
Delay [ms]	28	13	- 15	- 53.57 %
JVM memory usage [bytes]	9 269 418	7225873	- 1 497 647	- 16.16 %

5.3 Service Offering Dynamic Manipulation Experiments Using WSOI

We have 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 in which a Web Service is no longer appropriate and creates a new composition with some other Web Service, which may have to be found. A special simple case is 'switching between (provider) Web Services' when the consumer simply searches for and chooses another provider Web Service. Another alternative is 're-negotiation of SLAs' when the provider and the consumer negotiate a new custom-made SLA. A special simple case of this approach is the dynamic creation of new service offerings.

Our **analytical studies** are based on comparisons of the number of exchanged SOAP messages. On the other hand, in the **practical experiments** we measure average delay and average Java Virtual Machine (JVM) memory usage. For these experiments, we have set up a test-bed environment with several Web Service compositions in a local network. Some of these Web Services use WSOI, while some use only Axis. Consequently, our prototype implementation of WSOI was crucial for these experiments.

Tables 5a and 5b summarize one experiment comparing consumer-initiated switching between Web Services and switching between WSOL service offerings. Averages for delay and JVM memory usage were calculated as explained in Table 1a, but using 100 test runs. Consumer-initiated switching between service offerings was about 54% faster and consumed about 16% less memory. Tables 6a and 6b summarize a similar experiment for provider-initiated switching. In this experiment, provider-initiated switching between service offerings was about 15% faster and consumed about 17% less memory. Note that in both experiments switching between Web Services is relatively simple and straightforward. In more complex experiments, 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].

Table 6a. Description of an Experiment Comparing Provider-initiated Switching between Web Services and Provider-initiated Switching between WSOL Service Offerings

Description	Switching of Web Services	Switching of Service Offerings
Number of Web Services	3 (consumer and 2 providers)	2 (consumer and provider)
Distribution	Same computer	Same computer
Provider Web Services	Simple stock notification Web Service (both providers)	Simple stock notification Web Service
Infrastructure for providers	Tomcat, 2 * standard provider-side Axis modules (2 Axis engines – one per provider), 2 * WSOISessionHandler	Tomcat, standard provider-side Axis modules, WSOISessionHandler, WSOI-specific modules
Infrastructure for consumers	Standard consumer-side Axis modules, WSOISessionHandler	Standard consumer-side Axis modules, WSOISessionHandler
Number of exchanged SOAP messages	4 (closeSessionSuggestedReplacementWS and reply, openSession and reply)	2 (switchSuggested and reply)
How the consumer finds the replacement WS or SO	The old provider recommends replacement	The provider recommends replacement
How the provider finds the replacement WS or SO	Searches simple internal data structures	Searches WSOI data structures
Start time for measuring delay	The old provider determines during accounting that some constraint was not satisfied	The provider determines during accounting that some constraint was not satisfied
Stop time for measuring delay	The consumer receives from the new provider the reply for the openSession message	The provider finishes switching the consumer to the new service offering
Software participating in JVM memory usage	Tomcat, 2 * standard provider-side Axis modules, standard consumer-side Axis modules, 3 * WSOISessionHandler, implementation of the consumer and both providers	Tomcat, standard provider-side Axis modules, standard consumer-side Axis modules, 2 * WSOISessionHandler, 1 * other WSOI-specific modules, implementation of the consumer and the provider

Table 6b. Results of an Experiment Comparing Provider-initiated Switching between Web Services and Provider-initiated Switching between WSOL Service Offerings

Measured Value [Units]	Switching between Web Services (= A)	Switching between Service Offerings (= B)	Difference (= B-A)	Relative Difference (= (B-A)/A) [%]
Delay [ms]	319	271	- 48	- 15.05 %
JVM memory usage [bytes]	9 415 472	7 771 771	- 1 643 701	- 17.46 %

6 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, 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 lightweightness of WSOL, WSOI is simpler and with less run-time overhead than management infrastructures for WSLA and WSML.

We have integrated into WSOI, on top of Apache 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. 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. 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 are still working on support for manipulation of service offerings dynamic relationships. Likewise, we currently have only rudimentary support for creation of service offerings. While our current WSOI prototype demonstrates the main concepts, the improved prototype will demonstrate the complete system. We are currently conducting additional analytical studies and experiments with the manipulation of service offerings, and WSOI in general. For example, we plan experiments comparing WSOL and languages using custom-made SLAs. We want to more precisely determine benefits, usability, and limits of dynamic manipulation of WSOL service offerings.

While the WSOL language is relatively complete and stable, we also have some items for future work in this area [3]. 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. Likewise, 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. World Wide Web Consortium (W3C): Web Services Description Requirements. W3C Working Draft 28 October 2002. On-line at: http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028/ (2002)
2. Tosic, V., Ma, W., Pagurek, B., Esfandiari, B.: On the Dynamic Manipulation of Classes of Service for XML Web Services. Research Report SCE-03-15, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, June 2003. On-line at: http://www.sce.carleton.ca/netmanage/papers/TosicEtAlResRepJune2003.pdf (2003)
3. Tosic, V., Pagurek, B., Patel, B. Esfandiari, B., Ma, W.: Management Applications of the Web Service Offerings Language (WSOL). Proc. of CAiSE'03 (Velden, Austria, June 2003). Lecture Notes in Computer Science (LNCS), No. 2681. Springer-Verlag (2003) 468-484
4. Tosic, 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. 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)
7. 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)
8. 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)
9. 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
10. 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)

11. The DAML Services Coalition: DAML-S: Semantic Markup for Web Services. WWW page for DAML-S version 0.7. Oct. 2, 2002. On-line at: http://www.daml.org/services/daml-s/0.7/daml-s.html (2002)
12. 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
13. Brose, G.: Securing Web Services with SOAP Security Proxies. Proc. of ICWS'03 (Las Vegas, USA, June 2003), CSREA Press (2003) 231-234
14. The Axis Development Team: Axis Architecture Guide, Version 1.0. Apache Axis WWW page. On-line at: http://cvs.apache.org/viewcvs.cgi/~checkout~/xml-axis/java/docs/architecture-guide.html (2003)