# On the Dynamic Manipulation of Classes of Service 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
{vladimir, weima, bernie, babak} @ sce.carleton.ca

**Abstract.** Classes of service are a mechanism for differentiation of service and quality of service (QoS) that incurs less overhead than custom-made Service Level Agreements (SLAs), user profiles, and other alternatives. For their formal representation for XML (Extensible Markup Language) Web Services, we have developed the Web Service Offerings Language (WSOL). A service offering in WSOL is a formal description of one class of service of a Web Service. It contains various constraints (functional, QoS, access rights), management statements (e.g., prices, monetary penalties, and management responsibilities), and reusability constructs (determining static relationships between service offerings). One Web Service can be associated with multiple service offerings. Dynamic (i.e., run-time) relationships between service offerings are specified outside WSOL service offerings, in a special format. In addition to the WSOL language, we are developing the Web Service Offerings Infrastructure (WSOI) that addresses monitoring and accounting of WSOL service offerings and dynamic adaptation of Web Service compositions using manipulation of service offerings. Five mechanisms for the dynamic manipulation of service offerings are explored: switching (initiated by the consumer or the provider Web Service), deactivation, reactivation, deletion, and creation of service offerings. WSOL relationships between service offerings are particularly useful for these mechanisms. From analytical studies and practical experiments with dynamic adaptation scenarios involving WSOL and alternative approaches, we conclude that manipulation of classes of service is simpler and faster than recomposition of Web Services and re-negotiation of SLAs. While it has limitations, it can be a useful additional lightweight dynamic adaptation approach.

## 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]. Hereafter, we use the term 'Web Service' as a synonym for the term 'XML Web Service'.

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. The goal of Web Service technologies is a standard platform, based on XML, for distributed application-to-application (A2A) and business-to-business (B2B) integration.

A Web Service can provide several ports. Each port implements a particular port type, which is collection of one or more operations. Operations can be input-output, input-only, output-only, or output-input. Consequently, they contain input, output, and/or fault messages. Different protocols for messaging are supported, most notably SOAP. Communication between Web Services can be synchronous or asynchronous; based on Remote Procedure Calls (RPCs) using XML or on the exchange of XML documents. While Web Services can be used for providing services to human end users, their true power is leveraged through **compositions** (orchestrations, choreographies, flows, networks) of Web Services. Hereafter, by a **consumer** (requester, client) of a Web Service *A* we assume another Web Service that is composed with *A* and collaborates with it, not an end user (human) using *A*. One Web Service can serve many different consumers, possibly at the same time. On the other hand, we refer to *A* as the **provider** (supplier) Web Service. The composed Web Services can be distributed over the network, run on different platforms, implemented in different programming languages, and provided by different vendors.

At the start of our research—when SOAP, WSDL, and UDDI just appeared—we have noted the need for significantly extending Web Service technologies with better support for the specification of management information and for Web Service Management (WSM) and Web Service Composition Management (WSCM). For example, WSDL defines specification of functionality (messages, operations, port types), access methods, and location of Web Services. However, it does not support specification of classes of service, various constraints (guarantees or requirements), management statements, Service Level Agreements (SLAs) and other contracts between Web Services. Explicit, precise, and unambiguous specification of such information is crucial for management activities [2, 3, 4]. Further, appropriate infrastructures for monitoring of Web Service and, particularly, for dynamic adaptation of Web Service compositions were missing.

We were particularly intrigued that the very important concept of classes of service was missing from Web Service technologies. In certain circumstances, discussed in the paper, it can be useful for a Web Service to offer several different classes of service to consumers. In addition, dynamic (i.e., run-time) manipulation of classes of service can be used for both Web Service Management and Web Service Composition Management. Consequently, the focus of our research is enabling Web Services to provide and specify multiple classes of service and to perform management activities, such as monitoring and dynamic adaptation, with these classes of service.

For the formal specification of classes of service, various types of constraint and management statements, we have developed the **Web Service Offerings Language (WSOL)**, compatible with and complementary to WSDL. For the monitoring of classes of service specified in WSOL and for the manipulation of classes of service, we have developed the **Web Service Offerings Infrastructure (WSOI)**. In this

paper, we present our recent results related to the dynamic adaptation mechanisms based on the manipulation of WSOL classes of service. We have designed algorithms and protocols, designed and partially implemented appropriate management infrastructure support, and compared our dynamic adaptation mechanisms with alternatives using analytical studies and practical experiments. We find that that manipulation of classes of service using WSOL and WSOI is simpler and faster than the re-composition of Web Services and the re-negotiation of SLAs. While it has limitations, it can be a useful additional lightweight dynamic adaptation approach.

Recently, important results on the management of Web Services and Web Service compositions have been achieved in parallel with our work on WSOL and WSOI. For the XML specification of custom-made SLAs between Web Services, HP has developed the Web Service Management Language (WSML) [3, 5], while IBM has developed the Web Service Level Agreements (WSLA) language [4, 6]. Both languages are accompanied by appropriate management infrastructures [5, 6]. One of the main distinctive characteristics of our research compared to WSML and WSLA is that WSOL is based on the concept of a class of service, instead of the more demanding concept of a custom-made SLA. In addition, our WSOI contains explicit support for the dynamic adaptation of Web Service compositions using manipulation of classes of service.

This paper is organized as follows. In this section, we have introduced our research by defining background terminology, summarizing motivation, stating the paper topic and thesis, and differentiating from key related work. In the next section, we discuss the benefits of classes of service for Web Services and define the concept of a service offering. We give a brief overview of constructs in WSOL in Section 3. In Section 4, we describe how WSOI supports monitoring of classes of service specified in WSOL. In particular, we explain how we have extended the Apache Axis open-source SOAP engine with WSOI. Sections 5, 6, and 7 represent the core of the paper. In Section 5, we discuss five mechanisms for the dynamic manipulation of classes of service: switching (initiated by the consumer or the provider Web Service), deactivation, reactivation, deletion, and creation of service offerings. WSOL and WSOI support for these dynamic adaptation mechanisms is examined in Section 6. WSOL support is the specification of static and dynamic relationships between classes of service, while WSOI support includes special data structures, implementations of algorithms and protocols, and definitions of special management port types. An important part of our work is comparison of our dynamic adaptation mechanisms with alternatives, such as re-composition of Web Services. In Section 7, we explain how we perform such comparisons analytically and using experiments with prototype implementations. Section 8 contains an overview of some recent related works. We summarize the conclusions and directions for future work in Section 9.

## 2 Classes of Service for XML Web Services

In our work, by a '**class of service**' we mean a discrete variation of the complete service and quality of service (QoS) provided by one Web Service. We discuss classes

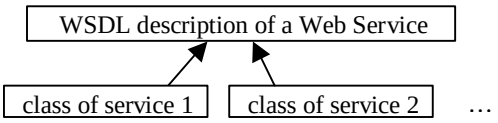



**Fig. 1.** Multiple Classes of Service for One Web Service

of service at the level of Web Services, not at the level of particular constraints or guarantees (e.g., response time) that are part of the overall service and QoS.

Classes of service of one Web Service refer to the same WSDL description (see Figure 1), 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, etc. The concept of classes of service also supports different capabilities, rights, and needs of potential consumers of the Web Service, including the power and type of devices on which they execute. Further, different classes of service may imply different utilization of the underlying hardware and software resources and, consequently, have different prices. Additionally, different classes of service can be used for different payment models, such as pay-per-use and subscription-based. To summarize, a Web Service with multiple classes of service can be used in different circumstances and by a wider range of consumers. Therefore, providing multiple classes of service enables the broadening of the market segment of a Web Service. It also enables the Web Service to better balance limited underlying resources and the price/performance ratio.

Providing classes of services is not the only possible way to customize constraints and management statements that a Web Service offers to its consumers. There are various alternatives, including custom-made Service Level Agreements (SLAs), user profiles, parameterization, and separate ports. However, the practice of telecommunication service provisioning shows that classes of service have relatively low overhead and complexity of management. One the goals of our research was to study mechanisms that have lower run-time overhead of management activities for Web Services [7, 8]. We wanted to accommodate relatively simple consumer Web Services and to support reduction of management overhead for provider 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. Consequently, we have decided to concentrate our research on classes of service. We are aware that they are not a complete replacement for all alternatives and that even the overhead of classes of service can be too high for some circumstances. However, even if some of the alternative approaches were more appropriate for particular circumstances, classes of service could be a useful addition and complement.

In our research, we use the term '**service offering**' to refer to the formal representation of a single class of service of one Web Service. Consequently, a service offering is a combination of formal representations of various constraints and management statements that determine the corresponding class of service. It can also be viewed as one simple contract or one SLA between the supplier Web Service, the consumer, and eventual management third parties. Hereafter, we will almost exclu-

sively use the term 'service offering', although in some cases we could have also used the term 'class of service of a Web Service'.

We specify service offerings separately from the WSDL description of the Web Service. While some constraints (particularly functional) rarely change during run-time, other constraints (particularly QoS constraints and prices/penalties) can be changed during run-time to better fit the execution circumstances. The separation of service offerings from WSDL descriptions enables that, if needed, service offerings can be deactivated, reactivated, created, or deleted dynamically without any modification of the underlying WSDL file.

In our work, consumers open sessions with provider Web Services. Sessions enable grouping of management information, e.g., for the measurement or calculation of periodic QoS metrics and the evaluation of QoS constraints. A Web Service can suggest different service offerings to different classes of consumer and maybe even several service offerings to the same consumer. Inside one session, only one service offering is used at a time. However, the consumer or the provider can initiate a change of service offerings, called switching, and other dynamic adaptation mechanisms discussed later in the paper. Some consumers might be allowed to open multiple parallel sessions with the same provider Web Service.

## 3 The Web Service Offerings Language (WSOL)

The **Web Service Offerings Language (WSOL)** is our language for the formal specification of classes of service, various constraints, and management statements for Web Services. The syntax of WSOL is defined using XML Schema, in a way compatible with WSDL 1.1. A WSOL file references one or more WSDL files and adds information that is not present in WSDL files.

The main **categories of constructs** in WSOL are:

1. service offerings,
2. constraints,
3. (management) statements,
4. reusability constructs, and
5. service offerings dynamic relationships.

We summarize the main characteristics of these categories of constructs in this section. An overview of WSOL can be found in [9, 7], while precise syntax, illustrative examples, and detailed discussions are given in [10].

As already stated, a WSOL **service offering** is the formal representation of one class of service and contains formal definitions of various constraints and management statements, as well as different reusability constructs. Definitions of WSOL service offerings can be long and complex. Therefore, in Figure 2 we have shown only some example parts of the definition of one service offering, *SO1*. This service offering contains the QoS constraint *QoScons2* and the management responsibility statement *MangResp1*. The other constraints and statements in this service offering are left out for brevity.

```
<wsol:serviceOffering name = ″SO1″ service = ″buyStock:
buyStockService″ 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: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>
```

**Fig. 2.** Parts of an Example WSOL Service Offering

Every WSOL **constraint** contains a Boolean expression that states some condition to be evaluated. Boolean expressions in constraints can also contain arithmetic, date/time/duration, and some simple string expressions. They can also contain calls to operations, which can be implemented by the provider Web Service, the party performing the evaluation of the constraint, or some external entity. The constraints can be evaluated before and/or after invocation of operations or periodically, at particular date/time instances. WSOL supports the formal specification of functional constraints, quality of service (QoS) constraints, and access rights. Functional constraints (pre-, post-, and future-conditions [7]) describe valid inputs and results of operations. QoS constraints describe properties such as performance, reliability, and availability. WSOL QoS constraints contain specifications of what QoS metrics are monitored, as well as when and by what entity. However, definition of QoS metrics (i.e., how they are measured or computed) is done in external reusable and extensible ontologies [11]. QoS constraints usually describe QoS guarantees. However, QoS constraints for output-input operations and some periodic QoS constraints can be used for the specification of QoS requirements. Access rights specify conditions under which any consumer using the current service offering has the right to invoke a particular operation. They are used in WSOL for differentiation of service. WSOL

can be extended with the formal specification of additional types of constraint using the XML Schema mechanisms.

A WSOL **statement** is any construct, other than a constraint, that states some important management information about the represented class of service. WSOL enables the formal specification of statements about management responsibility, subscription prices, pay-per-use prices, and monetary penalties to be paid if constraints are not met. WSOL can be extended with the formal specification of additional types of management statement (e.g., policies) using the XML Schema mechanisms.

Apart from definitions of constraints and management statements, WSOL service offerings can contain various **reusability constructs**. They are discussed in more detail in [12]. Most importantly, service offerings can be defined as extensions of other service offerings. The extending service offering contains all WSOL items (constraints, statements, reusability constructs) as the extended service offerings, as well as some additional WSOL items. Further, several constraints and/or statements can be gathered into a WSOL constraint group (CG). Then, constraints, statements, and constraint groups already defined elsewhere can be simply included in another service offering. Next, constraint group templates (CGTs) are parameterized constraint groups. They can be instantiated many times with different parameter values. These and other WSOL reusability constructs determine **static relationships between WSOL service offerings**. These relationships show similarities and differences between service offerings and do not change during run-time.

In addition, WSOL enables specification of dynamic relationships between service offerings. They can change during run-time, e.g., after dynamic creation of a new service offering. We use the term '**service offerings dynamic relationship** (SODR)' for such a relationship. One service offerings dynamic relationship states what class of service is an appropriate replacement if particular constraints from some other class of service cannot be met. Such relationships should not be built into definitions of service offerings, to avoid frequent modification of these definitions. Service offerings dynamic relationships are specified in a special XML format outside definitions of service offerings (often in separate files) to make their evolution independent from the evolution of other characteristics of a service offering. This format was discussed in [9, 10]. Static and, particularly, dynamic relationships between service offerings are very useful for the mechanisms for the manipulation of service offerings discussed later in the paper. In addition, they enable easier selection and negotiation of service offerings.

To verify the WSOL syntax, we have developed a **WSOL parser** called 'Premier' [10]. Its implementation is based on the Apache Xerces XML Java parser. This parser produces a DOM (Document Object Model) tree representation of WSOL files and reports eventual syntax errors and some semantic errors. We have also designed Java classes and XML description files that will be the results of the compilation of WSOL files. We have not yet finished a code generator that creates the designed Java classes and XML description files from DOM trees produced by our WSOL parser. A prototype WSOL compiler would then be a combination of the 'Premier' WSOL parser and this code generator.

## 4 Monitoring of WSOL Service Offerings

We are researching two groups of management applications of WSOL: Web Service Management (WSM) and Web Service Composition Management (WSCM) [9]. We concentrate our research of Web Service Management on monitoring of WSOL service offerings. This includes the evaluation of WSOL constraints, metering and calculation of used QoS metrics, and accounting of executed operations and evaluated WSOL constraints. On the other hand, in the Web Service Composition Management (WSCM) area we are primarily interested in dynamic adaptation mechanisms based on the manipulation of WSOL service offerings. To support both Web Service Management and Web Service Composition Management using WSOL, we are developing the **Web Service Offerings Infrastructure (WSOI)**. WSOI is the management infrastructure for WSOL. In this section, we summarize the monitoring of WSOL service offerings and how WSOI supports them. In the following sections, we discuss the dynamic adaptation mechanisms based on the manipulation of WSOL service offerings, how WSOI supports them, and how they relate to other possible mechanisms for dynamic adaptation of Web Services and Web Service compositions.

Different parties can be involved in the monitoring of a WSOL service offering: the provider, the consumer, and **management third parties** – software modules (e.g., Web Services) independent from the provider and the consumer [9]. Management third parties [4] can be used to perform monitoring and management actions that cannot be performed by the provider or the consumer. They can also be used when the consumer and the supplier do not trust each other, but both trust some independent and well-known Web Service Management entity. Some management third parties can be specialized for the evaluation of WSOL constraints, for the metering and calculation of used QoS metrics (or only particular groups of QoS metrics), or for the accounting and billing activities. Other management third parties might be used for a combination of these activities. While management third parties are important to achieve manageability, flexibility, and trust, they introduce additional complexity, overhead, and delays. These two opposite sets of issues can be balanced by concentrating monitoring and management actions into one or few management third parties.

For one service offering, only one party (a third party, the provider, or, in rare cases, the consumer) acts as a special management party – the **accounting party**. This party performs all accounting and billing activities for this service offering.

We are primarily researching cases where management third parties act as **SOAP intermediaries**. In other words, these management third parties intercept SOAP messages between the consumer and the provider, perform their management tasks, and piggyback their management results into SOAP headers [9]. WSOL also supports management parties that act as **probes**. Such management parties can send their results to other management parties using some agreed-upon management operations, which will be discussed later in the paper. Alternatively, these management parties can provide their results through operations that are invoked in appropriate WSOL constraints, using the WSOL external operation call mechanism.

The part of WSOI that performs monitoring of WSOL constraints is based on extensions of **Apache Axis** (Apache eXtensible Interaction System) [13], a popular open-source SOAP engine implemented in Java. A SOAP engine is an application 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 alter the processed SOAP message, e.g., add/remove headers. It can also perform some other message processing, e.g., measurement of QoS metrics or evaluation of constraints. An Axis **chain** is an ordered, pipelined collection of handlers. Since the Chain class is a subclass of the Handler class, every Axis chain is also a handler. Handlers exchange information through the **message context** data structure. It contains information about the request message, the response message, and a bag of properties. The message context properties determine how handlers process the message and can be modified by handlers.

The modular, flexible, and extensible Axis architecture enables implementing the support for monitoring of WSOL service offerings as a set of additional handlers and handler chains plugged into Axis. In WSOI, specialized Axis handlers perform WSOL-related metering and calculation of QoS metrics, evaluation of constraints, and accounting activities. Hereafter, we refer to these handlers as '**WSOI-specific handlers**'. We have designed these handlers so that a WSOL compiler can generate them automatically from WSOL files. However, since our prototype WSOL compiler is not yet fully implemented, we have manually implemented some of these handlers in our WSOI prototype.

We have studied several ways to design WSOI-specific handlers that evaluate WSOL constraints. One possible approach is that WSOL compiler creates generic expression-evaluation handlers that can be used for many different WSOL constraints. However, to reduce run-time overhead, we have chosen that that a WSOL compiler generates a specialized handler for every WSOL constraint, except for constraints that are results of instantiation of a WSOL constraint group template (CGT). For every constraint that is specified inside a WSOL constraint group template, a WSOL compiler generates a parameterized WSOI-specific handler class. It also creates a separate object of this handler class whenever the constraint group template is instantiated with some concrete parameter values.

There are also several possible designs for WSOI-specific handlers that measure or calculate QoS metrics. We have adopted that a WSOL compiler generates these handlers from ontological definitions of QoS metrics [11]. However, several application management instrumentation technologies, such as the Java Management Extensions (JMX) and the Desktop Management Interface (DMI), already exist. If such an instrumentation technology is already supported by some management party and its results can be related to QoS metrics used in WSOL service offerings, a WSOL compiler could generate appropriate calls to this infrastructure. We have left this issue for future research.

A WSOL compiler also generates special XML files describing what WSOI-specific handlers are used for a particular context (i.e., a particular operation, service

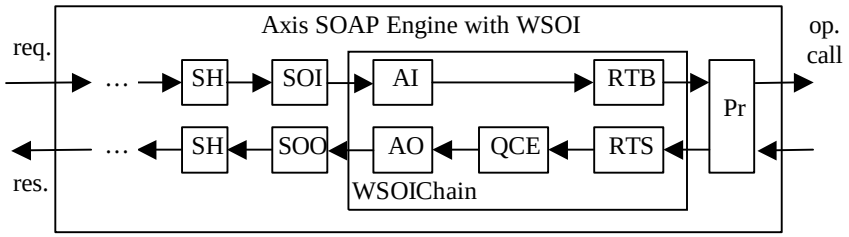



**Fig. 3.** Provider-side Axis SOAP Engine with WSOI

offering, and management party) and in what order. While a WSOL compiler might use a limited number of precedence rules for determining the order of the handlers, it seems that there is a possibility of errors. Therefore, we allow human Web Service administrators to generate or modify these XML files. We have developed the format for these XML files and the corresponding data structure inside WSOI. However, in our current prototype implementation of WSOI we have manually filed internal WSOI data structures with some data, while their loading from XML files is left for future work. We have also left the distribution of relevant results of a WSOL compiler (i.e., classes for WSOI-specific handlers and XML descriptions of how they are used) to management third parties for future research. Currently, we perform such distribution manually.

We transport WSOL-related information, such as results of measurement or calculation of QoS metrics and evaluation of constraints and price/penalty statements, between WSOI-specific handlers in special properties of the message context. As already mentioned, for the transport of WSOL-related information between different management parties we use SOAP headers. Special WSOI-specific handlers perform translations between these two formats. In addition to message context properties, WSOI also uses some additional data structures (often implemented as hashtables) for storing additional management information. For example, these data structures store what service offering is used in which session, what service offerings are active or deactivated, what is the history of unsatisfied and satisfied constraints, and what is the billing balance for a particular session. Many of these data structures are used for the dynamic adaptation mechanisms described in the following sections.

Let us now illustrate how WSOI-specific handlers are used for the monitoring of WSOL service offerings. In Figure 3, we have shown an example configuration of handlers inside the provider-side Axis SOAP engine extended with WSOI. In this example, the provider Web Service measures response time, evaluates a QoS constraint limiting this response time, and performs accounting. At the beginning of processing of an input (request) message, some Axis handlers standard for all Web Services are executed. These handlers are shown in Figure 3 as '…'. For every Web Service that supports WSOL, the WSOI-specific handlers WSOISessionHandler (SH), ServiceOfferingInput (SOI)/ServiceOfferingOutput (SOO), and the chain WSOIChain are executed. For processing input messages, WSOISessionHandler reads the session information from a SOAP header and writes it into the message

context. Similarly, ServiceOfferingInput reads WSOL-related information from SOAP headers and writes it into appropriate message context properties.

The main part of an Axis engine extended with WSOI is WSOIChain. It contains code that examines what is the operation invoked and what is the service offering used and dynamically constructs the chain of appropriate WSOI-specific handlers. For these decisions, WSOIChain uses internal data structures that are loaded from XML files generated by a WSOL compiler. In Figure 3, the first handler in WSOI-Chain for input message is AccountingInput (AI), which records the request message. Then, the ResponseTimeBegin (RTB) handler stores into message context the start time for measuring response time. After this, the standard Axis handler, Provider (Pr), is executed. It is outside WSOIChain. It dispatches the call to the Java object implementing the requested operation of the Web Service. This Java object returns its results back to the Provider handler. After Provider, handlers in the WSO-IChain process the output (i.e., response) message. First, the handler Response Time Stop (RTS) stores into message context the stop time for measuring response time, as well as the difference between this stop time and the start time stored by RTB. Next, the QoS constraint limiting response time is evaluated in the handler QoS Constraint Evaluation (QCE). This handler stores its results into the message context. Finally, the Accounting Output (AO) handler uses the information from the message context to calculate prices and eventual penalties to be paid. This information is also stored into the message context. After WSOIChain, ServiceOfferingOutput (SOO) and WSOISessionHandler (SH) process the output message. ServiceOfferingOutput performs the conversion in the opposite direction from ServiceOfferingInput. That is, it reads WSOL-related information from message context properties and writes into SOAP headers. For processing output messages, WSOISessionHandler reads the session information from message context and writes it into a SOAP header. Next, Axis handlers standard for all Web Services are executed.

While the evaluation of periodic constraints differs from the example illustrated in Figure 3, it is also supported by WSOI. Timer, a special active object in WSOI, initiates evaluation of periodic constraints and measurement or calculation of periodic QoS metrics. It invokes the WSOIChain object, which creates a chain of appropriate handlers and executes them. The results of such an evaluation, measurement, or calculation can be stored locally for future processing. They can also be reported to other management parties in a special notification SOAP message.

The described monitoring of WSOL service offerings inside an Axis engine extended with WSOI incurs some run-time overhead. We have performed some experiments to check this overhead. In particular, we have measured time delays and Java Virtual Machine (JVM) memory usage for the same Web Services running Axis with and without WSOI. The memory overhead introduced by WSOI was about 5% of the total memory consumed by Tomcat, Axis, and Java implementations of the involved Web Services. The response time increased about 25%. In our opinion, this is an acceptable overhead.

## 5 Mechanisms for the Dynamic Manipulation of Service Offerings

Another application area of WSOL is the management and dynamic adaptation of Web Service compositions. We are particularly interested in dynamic adaptation without breaking an existing relationship between a provider Web service and its consumer. To achieve this goal, we are exploring management and dynamic adaptation mechanisms that are based on the manipulation of classes of service, particularly WSOL service offerings. The five main mechanisms that we study are switching between service offerings, deactivation, reactivation, and deletion of existing service offerings, and creation of new service offerings. These mechanisms can be used between operation invocations that are part of the same session.

**Dynamic switching between service offerings** is changing which service offering a consumer uses. Either a consumer or a provider Web Service can initiate it. In the latter case, the consumer is asked for confirmation. The consumer can initiate it to dynamically adapt the service and/or QoS it receives without searching for another Web service. The provider can initiate it to gracefully upgrade or degrade its service and/or QoS in case of changes. For example, when a financial analysis Web Service is a consumer of a stock notification Web Service, switching between service offerings can be initiated by either of them to adapt to a turbulent stock market.

Since switching between service offerings is the basic dynamic adaptation mechanism in our research, let us illustrate it with an example. We will observe a consumer-initiated switching scenario, shown in Figure 4, in which one independent third party, denoted *A*, performs all management and accounting activities. The provider Web Service *P* has at least two service offerings *SO1* and *SO2*. At the beginning of the observation, the consumer *C* uses *SO1*. However, *C* also knows about other active service offerings of *P* that it can use, particularly *SO2*. At some time, *C* decides that it would be better if it used *SO2* instead of *SO1*. Therefore, *C* sends the accounting party *A* the 'switchSO' message containing the name of *SO2* (step 1 in Fig. 4). After the receipt of this message, *A* blocks and queues further requests from *C* to *P* in the given session (step 2). However, the requests that were received by *A* before the switching request are processed using *SO1*. When *A* finishes processing of all these requests, it sends *P* the 'switchSO' message containing the name of *SO2* (step 3). *P* checks whether the switching is possible, e.g., whether *SO2* is still active and *C* has the right to use it (step 4). Let us assume that the switching is possible, so *P* sends *A* the 'switchingOK' message (step 5). Then, *P* performs initialization of its internal activities and data structures related to *SO2* and finalization of activities and data structures related to *SO1* (step 6). In parallel, *A* also performs initialization and finalization of its activities and data structures, after it receives the 'switchingOK' message from *P* (step 7). When everything is ready for *C* to use *SO2*, *A* sends *C* the 'switchingOK' message (step 8). If *A* has queued any requests from *C*, it unblocks and processes them using *SO2* (step 9). Afterwards, *C* uses *SO2*.

We have also developed solutions for more complex scenarios and for the handling of special cases. For example, when more than one management third party is involved in *SO1* and/or *SO2*, *A* has to send them 'initAndFin' messages as part of step 7 discussed above. Also, the provider P may reject the consumer's request for

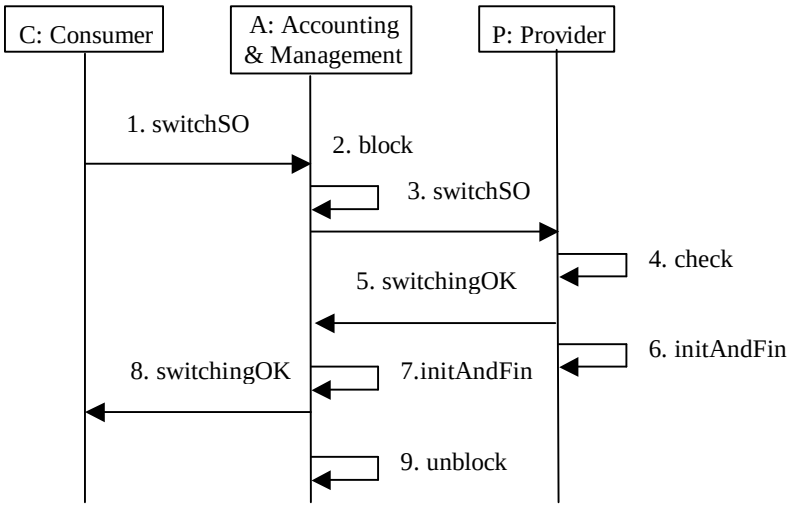



**Fig. 4.** Consumer-initiated Switching between Service Offerings

switching between service offerings, but suggest an alternative replacement service offering. In addition, when the accounting party for *SO2* is different from the accounting party for *SO1*, additional steps are needed to initialize the new accounting party and to forward it the requests queued at the old accounting party. On the other hand, when the provider Web Service performs all management and accounting activities, switching between service offerings becomes simpler.

**Deactivation and reactivation of service offerings** is used by a provider Web service when changes in operational circumstances affect what service offerings it can provide to its consumers. Some service offerings cannot be used in all circumstances. For example, it is sometimes impossible to achieve high QoS or it is dangerous to provide service offerings with low security. An example of changed circumstances is unexpected fluctuation in the QoS provided by Web Services used by the provider. Another example is some temporary disturbance of the communication between involved parties, e.g., due to mobility. This dynamic adaptation mechanism supports both graceful degradation and seamless service upgrades and expansions.

When a change of circumstances occurs, a provider Web Service can **dynamically deactivate service offerings** that cannot be supported in the new circumstances. The essential issue is what to do with consumers using the deactivated offering. We have developed support for handling such cases. In principle, the provider initiates switching of service offerings (e.g., to a service offering with less QoS, or with higher QoS and price) and asks the affected consumers for confirmation. Consumers can accept the suggested replacement service offering, initiate switching to another replacement service offering they prefer, or close the session with the provider. In many situations (e.g., when trust is important), it is better for an affected consumer to accept the

replacement service offering from the same provider than to search for another provider Web Service. If there is no appropriate replacement service offering to which consumers can be switched, the provider can initiate creation of new service offerings. When this is also not possible, other approaches to dynamic adaptation, e.g., re-composition of Web Services, have to be used.

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

If the change of circumstances is permanent, so the probability of future reactivation of a deactivated service offering is zero or very low, the provider Web Service can decide to **dynamically delete a service offering**. For example, if implementation of a provider Web Service is dynamically upgraded to improve performance, some of its service offerings with lower QoS might become redundant and can be deleted. Another example is when a management third party used in some service offering goes out of business. Deletion permanently removes support for a service offering. Only deactivated service offerings can be deleted.

**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, or in the execution environment. To some limited extent, it can also be performed on demand of important consumers. It then becomes a substitute for negotiation of a custom-made contract or SLA between Web Services.

This dynamic adaptation mechanism is needed to enable further flexibility, customizability, and adaptability of Web Services and Web Service compositions. While the dynamic adaptation mechanisms discussed above handle changes that are to some extent anticipated, the creation of new service offerings can be used for unanticipated changes. Not all circumstances of run-time operation (particularly, QoS) and not all consumer needs can be predicted in advance. In addition, Web Services and Web Service compositions can evolve (e.g., be upgraded) dynamically. Creation of new service offerings provides some support for evolution with minimal disruption of the operation and for propagation of changes to co-operating Web Services. For example, when a financial analysis Web Service uses a stock notification Web Service and the implementation of the stock notification is dynamically upgraded, then new service offerings of the stock notification Web Service can be created. To reflect these changes, new service offerings of the financial analysis Web Service can also be created. This propagates upgrade benefits from the stock notification Web Service to consumers of the financial analysis Web Service, e.g., decision support systems.

Note that the creation of new service offerings is not the creation of new Web Service characteristics described in WSDL files, such as operations and ports. It is the creation of new sets of constraints (e.g., updated QoS constraints) and management statements (e.g., updated prices and monetary penalties) for the existing WSDL description of a Web Service. It can accompany, but not completely replace, dynamic creation of new WSDL files, e.g., during dynamic evolution of Web Services.

Dynamic creation of new service offerings can be non-trivial and incur non-negligible overhead. It cannot be performed arbitrarily due to various possible con-

flicts. For example, QoS constraints cannot be set arbitrarily because of the limitations of used resources (including other Web Services), mutual dependencies of QoS parameters, and other issues. Detection and resolution of such conflicts can be very complex. Creating new service offerings might consume considerable time and resources of the Web Service. Therefore, we are researching only simple and limited creation of new service offerings as variations of existing service offerings. While we are concentrated on provider-initiated creation of service offerings, we also leave the possibility of consumer-initiated creation.

One possible way to dynamically create new service offerings is to use existing WSOI-specific handlers, but in different circumstances. For example, a stock notification Web Service might have two operations: 'stockValue' returns value of one stock symbol, while 'multipleStockValues' returns values for an array of stock symbols. Let us assume that service offering *SO1* contains a response time guarantee for 'stockValue', but not for 'multipleStockValues'. Then, dynamic creation of a new service offering *SO2* that contains response time guarantees for both operations is relatively simple and straightforward. It requires only an update of the descriptions which of these handlers are used in the given circumstances and in what order.

Another, even simpler, way to dynamically create a new service offering *SO3* from *SO1* is to simply strengthen the response time guarantee for 'stockValue', e.g., from '2 seconds' to '1 second'. A WSOL compiler generates parameterized WSOI-specific handlers from parameterized constraints defined inside WSOL constraint group templates (CGTs). Consequently, if the response time guarantee for 'stockValue' is defined inside a WSOL constraint group template, then the dynamic creation of *SO3* would not even require changes to descriptions of the order of used handlers. To enable monitoring of *SO3*, a change of the appropriate handler parameter would be enough.

We have envisioned, but not yet developed, a Java API (Application Programming Interface) that would enable generation of WSOL files from internal WSOI data structures. This API would be used in the process of dynamic creation or deletion of service offerings.

Apart from the above-mentioned dynamic adaptation mechanisms, other mechanisms related to the manipulation of service offerings can be studied. An important group of mechanisms is **deactivation, reactivation, deletion, and creation of service offerings dynamic relationships** (SODRs). We are working on these mechanisms primarily in the context of deactivation, reactivation, deletion, and creation of service offerings. After a service offering is deactivated, reactivated, or deleted, the corresponding service offerings dynamic relationships have to be deactivated, reactivated, or deleted. When a new service offering is created, new service offerings dynamic relationships can be created, while some old ones might be deactivated or even deleted. However, the relationship between the manipulation of service offerings dynamic relationships and the manipulation of involved service offerings can be more complex. For example, assume the stock notification Web Service provides three service offerings and at least one service offerings dynamic relationship. The service offerings dynamic relationship states that if service offering *SO3* is deactivated because the response time guarantee for the operation 'stockValue' cannot be

kept, then the appropriate replacement service offering is *SO2*. After several deactivations (and subsequent reactivations) of *SO1* due to response time guarantee for 'stockValue', the provider Web Services (or some entity managing it) notices that the vast majority of consumers prefer service offering *SO1* as a replacement for *SO3*. Then, the new service offerings dynamic relationship can be created and the old one can be deactivated.

Another pair of dynamic adaptation mechanisms related to service offerings is **allowing and disallowing particular consumers or classes of consumer to use some service offerings**. We have left such security-related mechanisms for future research.

The presented dynamic adaptation mechanisms are primarily initiated by the provider Web Service or an external entity (software or human administrator) managing the provider. Consumers might be allowed to initiate switching and, in exceptional cases and only for some consumers, creation of service offerings and manipulation of service offerings dynamic relationships. They should not be allowed access to the other discussed dynamic adaptation mechanisms. In addition, some provider Web Services might allow only particular consumers or classes of consumer to initiate switching of service offerings, e.g., due to security or performance reasons. Such restrictions could be permanent or limited to special circumstances. Furthermore, not all of these dynamic adaptation mechanisms have to be supported by a Web Service. For example, some Web Services might not implement creation and deletion of service offerings and service offerings dynamic relationships. This means that in such cases even the provider cannot initiate all dynamic adaptation mechanisms.

## 6 WSOL and WSOI Support for the Manipulation of Service Offerings

Now that we have presented the dynamic adaptation mechanisms based on the manipulation of service offerings, let us discuss how WSOL and WSOI support them.

The crucial WSOL language support for these mechanisms is the specification of various relationships, both static and dynamic, between service offerings. As mentioned earlier, WSOL contains a special format for the description of service offerings dynamic relationships (SODRs), which can change during run-time. One such dynamic relationship states what is the appropriate replacement service offering if some constraints from the current service offering were not satisfied and/or cannot be satisfied in the future. These relationships are essential for switching (particularly provider-initiated), deactivation, and reactivation of service offerings.

On the other hand, WSOL also contains a number of reusability constructs [12] that determine static (i.e., development-time) relationships between service offerings. These relationships show similarities an differences between service offerings. Three very important static relationships between service offerings are extension (single inheritance) of service offerings, inclusion of the same constraint, and instantiation of the same constraint group template. These and other static relationships between

service offerings and reusability constructs are very useful for the dynamic creation of service offerings.

WSOI supports the dynamic adaptation mechanisms based on the manipulation of service offerings in several different ways. First, it contains data structures storing necessary information. Second, it implements algorithms and message exchange protocols for these mechanisms. Third, it defines several special port types that contain operations relevant for monitoring and manipulation of WSOL service offerings. These data structures, algorithm and protocol implementations, and management port types are not based on Apache Axis. However, some WSOI-specific data structures are used both for monitoring and for manipulation of WSOL service offerings and updated by WSOI-specific Axis handlers. We summarize the WSOI support for the manipulation of WSOL service offerings in the following paragraphs. While we have designed and partially implemented the major elements of this infrastructure, we are still working on its improvement and full prototype implementation.

WSOI-specific data structures store information essential for determining whether manipulation of service offerings is necessary and what is the appropriate replacement service offering. For example, when constraint is evaluated, the result of this evaluation is stored into a special data structure that keeps track of satisfied and unsatisfied constraints in this session. Other data structures store descriptions of service offerings dynamic relationships, the information about what service offering is used in which session, and the information about what service offerings are active or deactivated. The provider Web Service uses information from all these data structures to determine whether deactivation and/or switching of the currently used service offering is appropriate and to what replacement service offering to switch the affected consumers. Other management parties can also keep some of these data structures, such as the one storing what service offering is used in a particular session. Some other data structures are characteristic for accounting parties, although the provider and consumer can also implement them. For example, accounting parties can keep a billing balance as the sum of all prices and penalties that have to be paid. They can also keep history of prices and monetary penalties incurred in a particular session.

The algorithms for the manipulation of service offerings contain program logic deciding whether, what, how, and when the manipulation of service offerings should be performed. They use the data structures discussed above. For example, the algorithm for deactivation of service offerings first checks the history of unsatisfied and satisfied constraints to determine whether deactivation is appropriate. Then, it uses several data structures discussed in the previous paragraph to determine to which service offering to switch. Next, it invokes the algorithm and protocol for the provider-initiated switching of service offerings for affected consumers. Further, it handles eventual special cases, such as the possibility that some consumers reject the suggested replacement service offering. After all these activities are performed, this algorithm invokes the algorithm for deactivation of affected service offerings dynamic relationships.

The protocols for the manipulation of service offerings deal with the coordination of management parties to achieve the manipulation. For example, consumer-initiated

switching of service offerings shown in Figure 4 uses one such a protocol to coordinate the consumer, the provider, and the accounting party.

The majority of algorithms and protocols for the manipulation of service offerings are relevant for provider Web Services. However, consumers can also implement some algorithms and protocols, e.g., related to consumer-initiated switching of service offerings. If the used accounting party is third-party, it also participates in the switching of service offerings, both provider- and consumer-initiated. Consequently, it has to support the protocols used for switching. In addition, provider Web Services can outsource the decision-making related to the dynamic adaptation mechanisms to some external Web Service Composition Management entities. If such an entity is used, it has to implement relevant dynamic adaptation algorithms and protocols. In our research, we are interested in management activities without human intervention. Therefore, we want to implement all relevant decisions inside our dynamic adaptation algorithms and protocols. However, we leave the possibility that humans are consulted in some exceptional cases.

The operations used for monitoring and manipulation of service offerings are exposed to external entities through special management ports. For example, a provider Web Service can support the special operation for consumer-initiated switching of service offerings. We have defined several WSDL port types for the monitoring and manipulation of WSOL service offerings, as well as signatures of operations in these port types. Most of these port types can be implemented by provider Web Services, while some port types are also for other management parties. However, a WSOL-enabled Web Service (e.g., acting as a provider) might support only some of these port types and/or only some operations within a particular port type. Due to security reasons, only selected entities outside the Web Services implementing these port types are allowed to invoke these management operations. For example, only the consumer or the accounting party is allowed to invoke the provider operation 'switchSO' used in the scenario from Figure 4. Since Web Services define only endpoints and not their implementation, different Web Services (e.g., hosted by the same SOAP engine) implementing the same port type can also share the actual implementation of the operations in this port type. This reduces the overhead of supporting these port types. The advantage of defining management port types instead of separate management Web Services is easier discovery.

Let us now provide an overview of the port types we have defined:

First, the ServiceOfferingsInformation port type contains operations that provide information about available service offerings. It should be implemented by a provider Web Service. If a special independent broker Web Services is used to find, compare, and select service offerings, it can also implement some of these operations. A very important example of operations from this port type is a group of several similar operations that return information about active service offerings that a consumer is allowed to use and can choose from. Some of these operations return a WSOL file with descriptions of service offerings, while some return a URI (Uniform Resource Identifier) to such a WSOL file. These operations are particularly useful at the beginning of a session between the consumer and the provider Web Service. Similarly, this port type contains operations that return information (e.g., a WSOL file or a URI

to it) about relevant service offerings dynamic relationships. Another example is the group of operations that return up-to-date information what service offerings are active or deactivated.

Second, the ServiceOfferingsManagement-Provider port type contains operations used by a provider Web Service to initiate or participate in the discussed mechanisms for the manipulation of service offerings. An example is the operation 'switchSO' used in the scenario from Figure 4. Additionally, it contains an operation used at the beginning of a session to choose one service offering from the list of available service offerings. The program logic for this operation is similar to (but significantly simpler than) to the logic for switching of service offerings. Operations for opening and closing sessions between the consumer and the provider can be part of this port type or another management port type with operations for session management.

Third, ServiceOfferingsManagement-Consumer contains operations used by a consumer to participate in the discussed mechanisms for the manipulation of service offerings. An example is the operation that starts the scenario from Figure 4.

Fourth, ServiceOfferingsManagement-AccParty contains operations used by an accounting party to participate in the manipulation of service offerings. An example is the operation 'switchSO' used in the scenario from Figure 4.

Fifth, ServiceOfferingsManagement-MgmtParty contains operations used by any party performing monitoring of WSOL service offerings. These operations enable management parties to participate in the discussed mechanisms for the manipulation of service offerings. An example is the operation 'initAndFin', used in the scenario from Figure 4. This operation performs initialization of internal activities and data structures related to the new service offering and finalization of activities and data structures related to the old service offering.

Sixth, the ServiceOfferingsDynamicRelationshipsManagement port type contains operations used by a provider Web Service to initiate or participate in the manipulation of service offerings dynamic relationships. An example is the operation 'deactivateSODR' that deactivates a particular service offerings dynamic relationship.

Seventh, the ServiceOfferingsNotification port type should be supported by all parties involved in the monitoring and manipulation of WSOL service offerings. Operations from this port type are used to exchange WSOL-related management information, e.g. results of QoS measurements and constraint evaluations, between management parties. While the majority of WSOL-related management information is exchanged using SOAP headers, operations for the explicit exchange of this information are also needed. For example, after a management party measures a periodic QoS metric and/or evaluates a periodic QoS constraint, it can use an operation from this port type implemented by another management party to inform it about the results. Providers and consumer can implement another operation that is used by an independent accounting party to inform about the values of metered or calculated QoS metrics, evaluated WSOL constraints, and their monetary consequences.

Eight, the ServiceOfferingsSecurity port type contains operations for security management of service offerings. Only a provider Web Service should implement this port type. Two examples are the operations for allowing and disallowing particu-

lar consumers or classes of consumer to use certain service offerings. We have left the study of such security-related issues for future work.

Ninth, ServiceOfferingsComparisons contains operations for determining static relationships between service offerings. These operations can be implemented by provider Web Services and/or by special independent brokers used to find, compare, and select service offerings. For example, one operation checks whether one service offering is an extension of another service offering. Another operation returns names of all available service offerings that are extensions of the given service offering. Yet another operation lists equivalencies and differences between two service offerings. This port type can contain many other operations.

## 7 Manipulation of Service Offerings vs. Alternatives

Apart from the above mechanisms based on the manipulation of service offerings, other approaches to dynamic adaptation of Web Service compositions exist. For example, many Web Services do not describe any class of service or SLA for their consumers. When a consumer starts using such a provider Web Service, but determines that the QoS is not satisfactory, it can stop using this Web Service and search for another provider Web Service implementing the same WSDL port types. We call such approach to dynamic adaptation 'switching between (provider) Web Services'. A similar approach to dynamic adaptation is 're-composition of Web Services'. In this approach, some entity (e.g., Web Service Composition Management software or human administrator) manages a Web Service composition. Often, an explicit description of a Web Service composition in an appropriate language, such as the Business Process Execution Language for Web Services (BPEL4WS), exists. When this management entity determines that one Web Service in the composition does not perform well enough, it breaks down the Web Service composition and creates a new composition using some replacement Web Service. Switching between Web Services is a special case of re-composition of Web Services in which the consumer acts as a management entity. When the consumer and the provider Web Service can negotiate a custom-made SLA (or a similar contract), dynamic adaptation can be achieved with the 're-negotiation of SLAs'. The creation of a new service offering is a special, limited, case of the latter approach to dynamic adaptation.

Manipulation of service offerings deals with a limited number of service offerings and involves communication only with already known provider Web Service and management third parties. Consequently, it seems as a simpler, faster, and more lightweight (in terms of less run-time overhead) approach to dynamic adaptation than re-negotiation of SLAs and re-composition of Web Services. However, to verify this observation, we have been performing analytical studies and practical experiments involving different dynamic adaptation approaches.

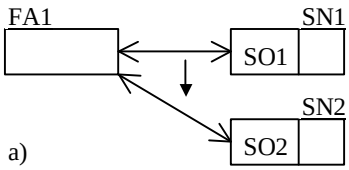
Our analytical studies concentrate on estimating and comparing delays introduced by different dynamic adaptation mechanisms. We have decided to model delays introduced by dynamic adaptation mechanisms with the number of exchanged SOAP messages. From our experience, internal operations performed by parties involved in

dynamic adaptation are relatively simple and fast compared to generating, transmitting, and processing of SOAP messages. While the delay of transmitting messages over the Internet depends on many factors, such as proximity of communicating nodes, we have assumed that delays for all exchanged SOAP messages are relatively similar and can be treated as equal. Consequently, we have compared only the number of SOAP messages exchanged in different dynamic adaptation scenarios.

For example, the scenario shown in Figure 4 introduces the delay of 4 SOAP messages. If instead of this scenario, *C* decides to switch to another provider Web Service *P'*, it has to close the session with *P* and open a new session with *P'*. In case the same accounting party is used for both *P* and *P'*, the delay introduced by this switching between Web Services is at least 6 messages. However, the number of exchanged messages in the latter scenario can be significantly higher, e.g., because selection of a new service offering from *P'* might require additional messages.

We have also generated formulae that determine the number of exchanged SOAP messages for different dynamic adaptation mechanisms, both those involving manipulation of classes of service and their alternatives. These formulae take into consideration issues such as the number of management third parties and whether the old and new service offering use the same accounting party. We have verified these formulae on a number of example scenarios. These formulae also showed us that the delay of our dynamic adaptation mechanisms based on the manipulation of service offerings linearly increases with the number of involved management third parties and does not depend on the number of supported service offerings.

In addition to analytical studies, we have developed experiments to examine delay and run-time overhead in comparable scenarios involving our dynamic adaptation mechanisms and their alternatives. A simple example experiment is explained below. For these experiments, we have set up a test-bed environment with several Web Service compositions. In this environment, Web Services use Apache Axis executing over Apache Tomcat and run on different computers in a local laboratory network. Some of these Web Services support WSOL and WSOI, while some do not. In addition, we have set up a test UDDI directory for experiments in which discovery of Web Services is performed dynamically. While we have already completed some experiments, we are currently performing additional ones. Further, we also plan to run some experiments for comparing overhead incurred by WSOL and the competing languages (WSML or WSLA), when used in similar circumstances.



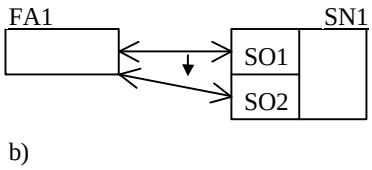




**Fig. 5.a.** Switching between Web Services

**Fig. 5.b.** Switching between Service Offerings

Let us now discuss a simple experiment when a financial analysis (FA) Web Services invokes a stock notification (SN) Web Service to find out current values of different stock symbols. In other words, the financial analysis Web Service is a consumer of the stock notification Web Service. In this experiment, all Web Services are located on the same computer and use Axis extended with WSOI. We will compare the switching between Web Services with the switching between service offerings.

In the first case, shown in Figure 5.a, there are two stock notification Web Services, *SN1* and *SN2*, implementing the same WSDL port type, but using different service offerings. *SN1* uses *SO1*, while *SN2* uses *SO2*. A financial analysis Web Service *FA1* uses *SN1* and its *SO1*. However, after some time, *SO1* is no longer appropriate for *FA1*. Therefore, *FA1* closes the session with *SN1* and opens a session with *SN2*. After this switching, *FA1* continues to use *SN2* and its *SO2*. In reality, *FA1* would use a UDDI directory or WSIL (Web Services Inspection Language) to find out that *SN2* provides *SO2*. However, in this simple example, we have hard-coded this information into *FA1*. This first case represents simple switching between Web Services.

We compare the above case with the case that uses switching between service offerings, shown in Figure 5.b. In this second case, *SN1* provides two service offerings, *SO1* and *SO2*, and *FA1* first uses *SN1* with *SO1*. *FA1*, *SO1*, *SO2*, and the WSDL description of *SN1* are the same as in the first case. When the need for dynamic adaptation arises, *FA1* switches from using *SO1* to using *SO2* without closing the session with *SN1*. After the switching, *FA1* uses *SN1* and its *SO2*.

After conducting this experiment a number of times and averaging the results, we have concluded that in this simple experiment the switching between service offerings is about 18% faster than the switching between Web Services. It also consumes less memory (about 4% of the total memory consumed by Tomcat, Axis, WSOI, and Java implementations of the involved Web Services). In more complex experiments, when Web Services execute on different computers and/or switching between Web Services requires access to a UDDI directory, the advantages of dynamic adaptation mechanisms based on the manipulation of service offerings are significantly greater.

Our analytical studies and practical experiments support our initial observation that manipulation of service offerings is simpler, faster, and incurs less overhead than the re-composition of Web Services and the re-negotiation of SLAs. In addition, it provides additional flexibility and enhances robustness of the relationship between

a provider Web Service and its consumer. This robustness is important when the consumer trusts the current provider, but does not yet trust alternative providers.

However, compared to the re-composition of Web Services, manipulation of service offerings has limitations. Service offerings of one Web Service differ only in constraints and management statements, which might not be enough for adaptation. Further, appropriate alternative service offerings cannot always be found or created. Therefore, manipulation of service offerings is a complement to, and not a complete replacement for, the re-composition of Web Services. On the contrary, it can be either a complement to or a lightweight replacement for the re-negotiation of SLAs, because they have similar limitations.

Consequently, we suggest that a management system for dynamic adaptation of Web Service compositions integrates the manipulation of service offerings and the re-composition of Web Services. When a need for dynamic adaptation arises, such system would first try to find a replacement service offering from the same provider Web Service. Only when this is not possible, the system would try to find a replacement provider Web Service and perform re-composition. In some cases (e.g., to achieve uninterrupted service), the used provider Web Service could supply a temporary replacement service offering while the consumer searches for another, more appropriate, Web Service. Other approaches to dynamic adaptation, such as re-negotiation of SLAs, could also be integrated into such management system to address cases when they are appropriate. A possible approach to classification and integration of different approaches to dynamic adaptation of distributed component compositions (including Web Service compositions) are discussed in [14].

## 8 Related Work

Our work on WSOL draws from the considerable previous work on differentiated classes of service and formal representation of various constraints in other areas, such as [15-19]. We were also influenced by works on run-time software evolution, such as [20], although we do not research architecture-based adaptation. At the beginning of our research, there was no relevant work of this kind for Web Services. In parallel with our research, several related works emerged.

The most important related works are the two recent languages for the formal XML-based specification of custom-made SLAs for Web Service: the **Web Service Level Agreements (WSLA)** [4, 6] from IBM and the HP work on the formal specification of Web Service SLAs [3, 5]. The latter work seems to be part of the HP's **Web Service Management Language (WSML)**. 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. These languages specify more detail for QoS constraints than WSOL and specify custom-made SLAs, not classes of service. In these aspects, they are more powerful than WSOL. It seems that this results in higher run-time overhead than the overhead of the simpler WSOL. On the other hand, WSOL also has advantages. In addition to QoS constraints and price/penalty statements, it enables formal specification of func-

tional constraints, access rights, various reusability constructs, and service offerings dynamic relationships. It is can be extended with support for additional types of constraint and management statement. Both WSLA and WSML are accompanied by appropriate management infrastructures [5, 6]. These infrastructures are more powerful, but also more complex, than WSOI. However, only WSOI contains built-in support for the dynamic adaptation mechanism based on the manipulation of service offerings. To conclude, while both WSLA and WSML are very good languages with adequate management infrastructures, they do not address all the issues that WSOL and WSOI do. The simpler and more lightweight WSOL and WSOI might be a better choice for Web Services for which additional run-time overhead is an important issue. Web Services in mobile and ubiquitous computing are one example.

Another recent related work is **WS-Policy** [21] – a general framework for the specification of policies for Web Services. A policy can be any property of a Web Service or its parts, so it corresponds to WSOL concepts of a constraint and a management statement. WS-Policy is only a general framework, while the details of the specification of particular categories of policies will be defined in specialized languages. The only such specialized language currently developed is WS-SecurityPolicy. WS-PolicyAssertions can be used for the formal specification of functional constraints, but the contained expressions can be specified in any language. It is not clear whether and when some specialized languages for the specification of QoS policies, prices/penalties, and other management issues will be developed. Another set of issues is where, when, and how are WS-Policy policies monitored and evaluated. WS-Policy has a number of good features, such as flexibility, extensibility, and reusability. However, it does not have the concept of a class of service nor the support for specification, monitoring, and manipulation of classes of service. Some other advantages of our research are specification of static and dynamic relationships between classes of service, explicit support for management applications in WSOI and WSOL, formal specification of various constraints and management statements, unified representation of expressions, and wider range of reusability constructs.

Apart from these recent works that are closely related to and partially competing with WSOL and WSOI, there are several other resent works that recognize the importance of the formal specification of various constraints, SLAs, and contracts for Web Services and special types of Web Service. The DAML-S (DAML-Services) [22] community works on semantic descriptions of Web Services, including specification of some functional and some QoS constraints. However, these specifications are not precise and detailed enough to be usable for the actual monitoring and management activities. The OGSA (Open Grid Services Architecture) [23] community also recognizes the need for formal specification of constraints, SLAs, and contracts. However, a Grid Service is a very special Web Service and it is still not clear how the future results from the OGSA community will relate to general Web Services. The notion of WSEL (Web Services Endpoint Language) has been mentioned in the literature [24], but with no detailed publication to date. One of the goals stated for WSEL was the specification of some constraints, including QoS, for Web Services. In addition, several research projects related to the specification and/or management of QoS for Web Service have been started recently.

In summary, our work on WSOL and WSOI is the only one that provides detailed support for the specification, monitoring, and manipulation of classes of service for Web Services. Compared with the related languages, WSOL has advantages in expressive power, lower run-time overhead, and support for management applications [8].

## 9 Conclusions and Future Work

Providing multiple classes of service empowers an XML Web Service to broaden the range of possible consumers and usage circumstances and to better balance limited underlying resources and the price/performance ratio. Our Web Service Offerings Language (WSOL) and Web Service Offerings Infrastructure (WSOI) enable specification, monitoring, and manipulation of classes of service for Web Services to the extent that is not provided by related works.

As part of WSOI, we have developed support for switching, deactivation, reactivation, deletion, and creation of service offerings. This WSOI support contains appropriate data structures, implementations of algorithms and protocols, and special management port types. It complements the WSOL support consisting of explicit specification of various static and dynamic relationships between service offerings.

We have performed analytical studies and practical experiments to compare these dynamic adaptation mechanisms with alternatives, such as re-composition of Web Services. These studies have supported the observation that manipulation of service offerings is simpler, faster, and more lightweight approach to dynamic adaptation than re-composition of Web Services and re-negotiation of SLAs. In addition, these dynamic adaptation mechanisms provide additional flexibility and enhance robustness of the relationship between a provider Web Service and its consumer. The main limitation of these mechanisms is the fact that appropriate replacement service offerings cannot be always found or created dynamically. Therefore, a Web Service Composition Management system should combine manipulation of service offerings with more powerful dynamic adaptation approaches.

While we have designed and partially implemented the main parts of WSOI, we still have to implement some parts of the prototype and maybe to extend and improve its design in some aspects. For example, we are still working on support for deactivation, reactivation, deletion, and creation of service offerings dynamic relationships. Likewise, we currently have only rudimentary support for creation of service offerings. 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 our work.

While WSOL is relatively complete and stable, we also have some items for future work in this area. The major one is the full implementation of a WSOL compiler. Likewise, a Java API for the generation of WSOL files would be beneficial. In addition, we have not yet addressed security issues and integration of WSOL files into UDDI directories. Related to the latter issue of discovery, we believe that WSOL

static and dynamic relationships between service offerings can be very useful for comparing similar service offerings and Web Services in the process of their negotiation and selection. However, we still have to research this area.

Let us finish the paper with a discussion of possible wider implications of our research. Several authors (e.g., [25]) have predicted that a future distributed computing platform will integrate and extend technologies currently developed for Web Services, Grid Computing, the Semantic Web, Peer-to-Peer (P2) systems, pervasive computing, and mobile computing. We believe that in such environment the importance of classes of service for Web Services and the mechanisms for their manipulation will increase. In the Open Grid Services Architecture, Web Services are used not only for representing software, but also as abstractions for hardware and communication resources. For such "implementations" of Web Services, issues related to QoS, access rights, prices and penalties, resource utilization, and price/performance ratio can be very significant. Since we have suggested definition of QoS metrics in ontologies that are outside particular WSOL service offerings, we also see our work as compatible with Semantic Web technologies. Further, Web Services with classes of service can form peer-to-peer networks. In such a case, manipulation of service offerings enables managing their compositions without an external Web Service Composition Management entity and even without explicit descriptions of Web Service compositions. Since classes of service are a lightweight approach to customization of service and QoS and the manipulation of service offerings is a lightweight approach to dynamic adaptation, we see them as suitable for resource-constrained devices in pervasive and mobile computing. In addition, manipulation of service offerings can be used for handling temporary disturbances, which might occur relatively often in mobile computing.

## 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., Pagurek, B., Patel, K.: WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services. Res. Rep. OCIECE-02-06. Ottawa-Carleton Institute for Electrical and Computer Engineering. Nov. 15, 2002. On-line at: http://www.sce.carleton.ca/netmanage/papers/TosicEtAlResRepNov2002.pdf (2002)
3. 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)
4. 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)
5. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA Monitoring for Web Services. In Proc. of the 13th IFIP/IEEE International Workshop on Distrib-

uted Systems: Operations and Management, DSOM 2002 (Montreal, Canada, Oct. 2002). Lecture Notes in Computer Science (LNCS), No. 2506. Springer-Verlag (2002) 28-41
6. Dan, A., Franck, R., Keller, A., King, R., Ludwig, H.: Web Service Level Agreement (WSLA) Language Specification. In Documentation for the Web Services Toolkit, Version 3.2.1. Aug. 9, 2002. International Business Machines Corporation (IBM) (2002)
7. Tosic, V., Pagurek, B., Patel, K.: WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services. Res. Rep. OCIECE-02-06. Ottawa-Carleton Institute for Electrical and Computer Engineering. Nov. 15, 2002. On-line at: http://www.sce.carleton.ca/netmanage/papers/TosicEtAlResRepNov2002.pdf (2002)
8. Tosic, V., Patel, K., Pagurek, B.: WSOL – A Language for the Formal Specification of Classes of Service for Web Services. To be publ. in Proc. of ICWS'03 - The First International Conference on Web Services (Las Vegas, USA, June 2003)
9. Tosic, V., Pagurek, B., Patel, B. Esfandiari, B., Ma, W.: Management Applications of the Web Service Offerings Language (WSOL). To be publ. in Proc. of the 15th Conference On Advanced Information Systems Engineering - CAiSE'03 (Velden, Austria, June 2003). Springer-Verlag, Lecture Notes in Computer Science (LNCS)
10. 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)
11. Tosic, V., Esfandiari, B., Pagurek, B., Patel, K.: On Requirements for Ontologies in Management of Web Services. In Proc. of the Workshop on Web Services, e-Business, and the Semantic Web – WES at CAiSE'02 (Toronto, Canada, May 2002). Lecture Notes in Computer Science (LNCS), No. 2512. Springer-Verlag (2002) 237-247
12. Tosic, V., Patel. K., Pagurek, B.: Reusability Constructs in the Web Service Offerings Language (WSOL). To be publ. in Proc. of the Workshop on Web Services, e-Business, and the Semantic Web – WES at CAiSE'03 (Velden, Austria, June 2003). Revised extended version published as: Res. Rep. SCE-03-14, The Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. May 2003. On-line at: http://www.sce.carleton.ca/netmanage/papers/TosicEtAlRepMay2003.pdf (2003)
13. 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)
14. Tosic, V., Pagurek, B., Esfandiari, B., Patel, K.: On Various Approaches to Dynamic Adaptation of Distributed Component Compositions. Res. Rep. OCIECE-02-02. Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE). June 2002. On-line at: http://www.sce.carleton.ca/netmanage/papers/TosicEtAlResRepJune2002.pdf (2002)
15. Aimoto, T., Miyake, S.: Overview of DiffServ Technology: Its Mechanisms and Implementation. IEICE Trans. Inf. & Syst., Vol. E83-D, No. 5 (May 2000) IEICE (2000) 957-964
16. Kristiansen L.: (ed.) Service Architecture, Version 5.0. TINA-C (Telecommunications Information Networking Architecture Consortium) specification. (June 16, 1997) On-line: http://www.tinac.com/specifications/documents/sa50-main.pdf (1997)
17. Beugnard, A., Jezequel, J.-M., Plouzeau, N., Watkins, D.: Making Components Contract Aware. Computer, Vol. 32, No. 7 (July 1999) IEEE (1999) 38-45
18. Mckee, P., Marshall, I.: Behavioural Specification using XML. In Proc. of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems - FTDCS'99, (Cape Town, South Africa, Dec. 1999) IEEE Computer Society Press (1999) 53-59
19. Jacobsen, H.-A., Karamer, B. J.: Modeling Interface Definition Language Extensions. In Proc. Technology of Object-Oriented Languages and Systems - TOOLS Pacific 2000 (Sydney, Australia, November 2000) IEEE Computer Society Press (2000) 241-252

20. Oreizy, P., Medvidovic, N., Taylor, R. N.: Architecture-Based Software Runtime Evolution. In Proc. of the International Conference on Software Engineering 1998 - ICSE'98 (Kyoto, Japan, Apr. 1998) ACM Press (1998) 177-186
21. 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)
22. 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)
23. Foster, I., Keselman, C., Nick, J. M., Tuecke, S.: Grid Services for Distributed Systems Integration. Computer, Vol. 35, No. 6 (June 2002) IEEE –CS (2002) 37-46
24. Ferguson, D. F.: Web Services Architecture: Direction and Position Paper. In Proc. of the W3C Workshop on Web Services – WSWS'01 (San Jose, USA, Apr. 2001) W3C. On-line at: http://www.w3c.org/2001/03/WSWS-popa/paper44 (2001)
25. Milenkovic, M., Robinson, S. H., Knauerhase, R. C., Barkai, D., Garg, S., Tewari, V., Anderson, T. A., Bowman, M.: Toward Internet Distributed Computing. Computer, Vol. 36, No. 5 (May 2003) IEEE –CS (2003) 38-46