



Management applications of the Web Service Offerings Language (WSOL)

Vladimir Tosic*, Bernard Pagurek, Kruti Patel, Babak Esfandiari, Wei Ma

Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, Ont., Canada K1S 5B6

Abstract

We discuss Web Service Management (WSM) and Web Service Composition Management (WSCM) applications of the Web Service Offerings Language (WSOL) and how the language supports these applications. WSOL is a language for the formal specification of classes of service, various constraints (functional constraints, Quality of Service—QoS, and access rights), and management statements (prices, monetary penalties, and management responsibilities) for Web Services. Describing a Web Service in WSOL, in addition to the Web Services Description Language, enables monitoring, metering, accounting, and management of Web Services. Metering of QoS metrics and evaluation of constraints can be the responsibility of the provider Web Service, the consumer, and/or one or more mutually trusted third parties (SOAP intermediaries or probes). Further, manipulation (switching, deactivation, reactivation, deletion, or creation) of classes of service can be used for dynamic (run-time) adaptation and management of Web Service compositions. To demonstrate the usefulness of WSOL for WSM and WSCM, we have developed a corresponding management infrastructure, the Web Service Offerings Infrastructure (WSOI). WSOI enables monitoring of WSOL-enabled Web Services and dynamic manipulation of their classes of service.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Web Service; Description language; Class of service; Web Service Management; Web Service Composition Management

1. Introduction and motivation

The World Wide Web Consortium defines a **Web Service** as ‘a software application identified by a Uniform Resource Identifier (URI), whose interfaces and bindings are capable of being defined, described and discovered by Extensible Markup Language (XML) artifacts and [which] supports direct interactions with other software applications using XML-based messages via

*Corresponding author. Tel.: +1 519 858 1828; fax: +1 613 520 5727.

E-mail addresses: vladat@computer.org (V. Tosic), bernie@sce.carleton.ca (B. Pagurek), kruts.patel@lycos.com (K. Patel), babak@sce.carleton.ca (B. Esfandiari), weima@sce.carleton.ca (W. Ma).

URL: <http://www.sce.carleton.ca/~vladimir/WSOLpublications.htm>.

Internet-based protocols' [1]. The term 'XML Web Service' is sometimes used instead of the term 'Web Service'. A Web Service can provide several ports (URI endpoints). Each port implements a particular port type (interface), which is a collection of one or more operations. The main Web Service technologies are XML, the SOAP protocol for XML messaging, the Web Services Description Language (WSDL) language, and the Universal Description, Discovery, and Integration (UDDI) directory.

Web Service technologies are intended for application-to-application integration. They can be used for business-to-business (B2B) integration and/or for Enterprise Application Integration within companies. Consequently, their true power is leveraged through **compositions** (choreographies and orchestrations [2]) of Web Services. In a Web Service composition, a Web Service can play the roles of a **consumer** (client, requester) and a **provider** (supplier). For example, when a financial analysis Web Service FA1 uses operations of a stock notification Web Service SN1, then FA1 is the consumer and SN1 is the provider. A Web Service can at the same time be a consumer of many Web Services and a provider of many other Web Services. Hereafter, we assume that a consumer is a Web Service, not a human end user. The composed Web Services can be distributed over the Internet, run on different platforms, implemented in different programming languages, and provided by different businesses. Communication between Web Services can be synchronous or asynchronous; based on Remote Procedure Calls XML or on the exchange of XML documents.

Management of Web Services and their compositions is needed to ensure regular operation, to attain or surpass the guaranteed service and quality of service (QoS), to accommodate change, and to keep track of the consumed resources. For example, management has to be performed if a used Web Service becomes unavailable.

Web Service Management (WSM) is the management of a particular Web Service or a group of Web Services within the same domain of responsibility, such as the same business. It consists of the monitoring of the operation of the Web Service

and the control of the Web Service to meet the guaranteed service and QoS. Monitoring of a Web Service includes measurement and calculation of relevant QoS metrics, evaluation of various conditions (requirements and guarantees), calculation of prices and monetary penalties, and accounting of executed operations, measured/calculated QoS metrics, evaluated constraints, and monetary amounts to be paid. The monitoring of a Web Service can be performed by the provider, the consumer, and/or one or more **management third parties** [3]—entities (e.g., Web Services) independent from the provider and the consumer. Management third parties can be used to perform monitoring and management actions that the provider and the consumer cannot execute. They can also be used when the consumer and the provider do not trust each other. For example, when a consumer does not trust that its provider would accurately measure achieved QoS, these measurements can be outsourced to one or several management third parties trusted by both the provider and the consumer. While the use of management third parties reduces processing overhead placed on the provider and the consumer, their number should not be high because they introduce additional compositional complexity and communication overhead.

On the other hand, we define the **Web Service Composition Management (WSCM)** as the management of which Web Services are composed and how they interact. Interaction between the composed Web Services can be explicitly and formally described in various contracts. A contract is any formal agreement between the provider, the consumer, and, potentially, management third parties. For example, a contract can contain descriptions of provided operations, guarantees of maximum response time, prices, and/or legal responsibilities. Consequently, WSCM consists of the selection of Web Services to be composed, the selection and possibly negotiation of contracts between these Web Services, the monitoring of contract fulfillment, the modification of contracts that are no longer appropriate, and possibly the re-selection of Web Services to adapt to changes. The monitoring of contract fulfillment combines the monitoring of the composed Web Services.

To completely describe interactions between Web Services for WSM and WSCM, different information about service and QoS has to be specified. WSDL version 1.1 [4] is used for specification of port types, operations, messages, access methods (messaging protocols used), and location of ports of the Web Service. The new WSDL version 2.0 [5] brings several improvements, but not pertinent to contributions of this paper. While this WSDL information is mandatory for using Web Services, it is not enough for management applications. For example, it is also important to prescribe the flow of information and control between the composed Web Services. Several languages for such description of Web Service compositions were developed, the most popular of which is the Business Process Execution Language for Web Services (BPEL4WS). However, neither WSDL nor Web Service composition languages provide specification of what QoS metrics to measure or calculate, which requirements and guarantees to evaluate, when to perform measurement of QoS metrics and evaluation of conditions, what are the prices and potential monetary penalties to be paid, and what parties perform particular management activities. Explicit, precise, and unambiguous specification of such management information is essential for monitoring and management activities [3,6]. For example, QoS metrics and QoS guarantees are valuable for performance management, while prices and penalties are important for accounting management. In addition to the management of Web Services and their compositions, the formal specification of such information is beneficial for selection of most appropriate Web Services because QoS, price, and manageability can be important competitive advantages on a global market of Web Services implementing the same WSDL port types.

When a provider Web Service collaborates with a large number of different consumers in parallel, it often has to provide different levels of QoS to accommodate different characteristics and needs of the consumers. Measured QoS metrics, prices, and management parties and the other management information can also differ between the consumers.

A considerable body of work on formal specification of technical contracts and various management information, as well as customization of service and QoS exists in telecommunications and software engineering. Service-level agreements (SLAs), classes of service, profiles, and policies are often used in telecommunications and distributed systems for this purpose. In addition, several telecommunication architectures, including Diff-Serv (Differentiated Services), also provide customization of service and QoS. In software engineering, particular conditions (requirements and guarantees) that have to be evaluated are specified in the form of constraints. For example, several languages for the formal specification of QoS metrics and constraints for CORBA (Common Object Request Broker Architecture), such as QML (Quality Modeling Language) [7], were developed. Often, a distinction is made between functional (behavioral) constraints and QoS (extra-functional, non-functional) constraints. Functional constraints define conditions that a functionally correct operation invocation must satisfy, while QoS constraints describe properties such as performance, reliability, and availability. The need for the unified formal specification of various constraints and comprehensive technical contracts for software components was elaborated in [8]. As discussed in [9], the results of these prior works could not have been applied to Web Services directly. Some of the characteristics of Web Services are Internet-wide and B2B dynamic composition, heterogeneity of implementations and execution platforms, and the use of XML and WSDL.

To enable explicit and formal specification of classes of service, various categories of constraints, and management statements for Web Services, we have developed the **Web Service Offerings Language (WSOL)** [9–12]. Our goal for WSOL was to support both WSM and WSCM activities. While management is a critical business activity, it can incur significant overhead. Since we wanted to provide management even for simple Web Services, we have studied management solutions with relatively low run-time overhead. The most important of them is the concept of a **class of service**, which is a discrete variation of the

complete functionality and QoS provided by one Web Service. Since we wanted to extend existing Web Service technologies, WSOL is an XML-based language compatible with and complementary to WSDL version 1.1, but separate from it. A WSOL file references one or more WSDL files and describes additional management information that is not present in WSDL files. Consequently, dynamic (run-time) creation, deletion, deactivation, and reactivation of WSOL specifications can be performed without any modification of the referenced WSDL files. Such updates are needed relatively frequently during the management of Web Services and their compositions. As we will explain, none of the recent related works on the specification of custom-made SLAs, policies, QoS, semantic descriptions, and agreements for Web Services addresses the same WSM and WSCM issues as WSOL. The main examples of such works, developed in parallel with WSOL, are Web Service Level Agreement (WSLA) [3,13], Web Service Management Language (WSML) [6,14], SLAng [15], WS-Policy [16], UX [17], DAML-Services (DAML-S) [18], and WS-Agreement [19]. While WSOL has limitations, it also has a set of important unique characteristics and advantages.

To demonstrate and further explore the use of WSOL for the management of Web Services and their compositions, we have developed the corresponding management infrastructure—the **Web Service Offerings Infrastructure (WSOI)** [20,21]. WSOI enables monitoring of Web Services and dynamic manipulation of classes of service described in WSOL.

In this paper, we discuss and illustrate WSM and WSCM applications of WSOL and how the language supports these applications. In this section, we introduced the general area of our research and motivated our work. In the next section, we describe the main concepts and constructs in WSOL. In Section 3, we discuss applications of WSOL for WSM. In Section 4, we examine how manipulation of classes of service described in WSOL can be used for WSCM. A brief overview of the most important recent related works is presented in Section 5. In Section 6, we summarize how WSOL supports management applications, recap other conclusions, and outline

directions for future work. Throughout the paper, we use bolded text to emphasize definitions of important concepts and summaries of main characteristics of our work. Names of XML elements in the WSOL grammar and values of data tuples are written between < and > characters, while names of attributes in the WSOL grammar, values of constants, and terminological phrases are written between ‘ and ’ characters.

2. The Web Service Offerings Language (WSOL)

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

1. service offering (SO),
2. constraint,
3. management statement,
4. reusability element, and
5. service offerings dynamic relationship (SODR).

We summarize the main characteristics of these categories of constructs in the following five subsections. Further information can be found in our other publications about WSOL. In particular, precise syntax defined using XML Schema, illustrative examples, and discussion of the WSOL language constructs for an older version of WSOL are given in [10], while some recent improvements and additions in the new WSOL version 1.1 are presented in [12]. Other WSOL and WSOI publications [9,11,20–22] elaborate several additional aspects of our work.

Fig. 1 is a Unified Modeling Language (UML) class diagram illustrating the main WSOL constructs and their relationships. To avoid overcrowding, we have not depicted that constraints, statements, and reusability elements can also be specified inside WSOL files, but outside particular SOs. In addition, we have not listed all supported types of constraint, statement, reusability element, and expression.

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 Document Object Model (DOM) tree representation of WSOL files and reports syntax errors and some semantic

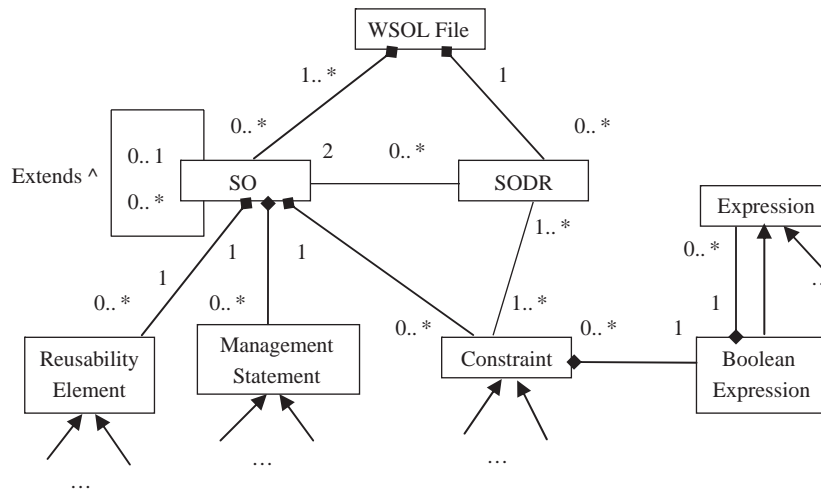


Fig. 1. Partial UML class diagram for WSOL concepts.

errors, if any. We have also designed Java classes that will be the results of the compilation of WSOL files. As will be explained in Section 3.2, the majority of these classes are WSOI-specific handlers—WSOI modules that perform WSOL-related measurement and/or calculation of QoS metrics, evaluation of constraints, and accounting activities. We have not yet finished a code generator that creates the designed Java classes 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.

2.1. Service offerings

The central concept in WSOL is a class of service. Classes of service of one Web Service refer to the same WSDL description, but differ in constraints and management statements. The practice of telecommunication service provisioning showed that classes of service have relatively low overhead and complexity of management compared to the more powerful alternatives, such as custom-made SLAs and profiles. Even in situations when classes of service are not the best choice for customization of service and QoS, they can be a useful addition and complement.

In WSOL, we use the term **'service offering' (SO)** to refer to the formal representation of a

single class of service of one Web Service. Consequently, a WSOL SO contains formal representations of various constraints and management statements that determine the corresponding class of service. It can also contain appropriate reusability elements. The specification of contractual legal aspects is currently not addressed by WSOL. Hereafter, we will almost exclusively use the term 'service offering', although in some cases we could have also used the term 'class of service of a Web Service'.

When a consumer invokes a provider's operation, it is necessary to determine which SO is used for this invocation. Further, for the measurement or calculation of periodic QoS metrics, the evaluation of QoS constraints, the accounting of subscription periods, and some other monitoring activities, it is necessary to perform grouping of management information from several consumer's invocations of the provider. Several mechanisms can be used for association between operation invocations and SOs and/or for grouping of management information. For simplicity, we have chosen that in our work consumers open sessions with providers. A provider can suggest multiple SOs to its consumer, but the consumer can use only one of them at a time in one session. On the other hand, a provider Web Service can have in parallel many open sessions, in some cases even several sessions with the same consumer. Inside a

session, the consumer can choose a SO to be used, as will be discussed in Section 4.1. Further, the consumer or the provider can initiate a change of SOs, called switching, and other dynamic adaptation mechanisms discussed in Section 4.2.

Let us now illustrate the benefits of SOs with an e-business example. A financial analysis Web Service consumes one or several stock notification Web Services and supplies results of its analyses, such as recommendations, to different consumers. It can provide its results on request from a consumer, but a consumer can also subscribe for periodic reports from the financial analysis Web Service. The former is an example of the pay-per-use business model, while the latter is an example of the subscription model. SOs could accommodate different classes of consumer, e.g., consumers that require a slightly different emphasis or depth of the financial analysis. Also, the SOs could differ in verbosity of results, the rate of unsolicited notification to consumers, in priority of notification of significant market disturbances, in the guaranteed response time, etc. SOs would differ in price and play an important role in balancing of the resources used by the Web Service when it processes requests from a large number of different consumers. Examples of these resources are processing power, threads, consumed memory, and used stock notification Web Services. The resources are limited and their use incurs some costs. For example, the used stock notification Web Services from other businesses have to be paid for, probably according to the received level of service and QoS. To summarize, a Web Service with multiple SOs can be used in different circumstances and by a wider range of consumers. Providing multiple SOs also enables the Web Service to better balance limited underlying resources and the price/performance ratio.

Fig. 2 shows an example WSOL SO. Descriptions of SOs are usually long and complex, so Fig. 2 contains only example parts that will be referenced in this section.

Important management information is conveyed in the attributes of the WSOL `<serviceOffering>` element. The 'service' applicability domain attribute specifies for which Web Service the SO should be used. The value of this attribute is a qualified

name whose namespace part refers to a WSDL file, while local part refers to the name of a Web Service defined in this WSDL file. In Fig. 2, the service offering SO2 is defined for the buyStock-Service Web Service. This Web Service is described in the WSDL file <http://www.buyStockProvider.ca/buyStockFile.wsdl>. For the majority of constraints, statements, reusability constructs, and QoS metrics inside a SO, the applicability domain is determined with three attributes: 'service', 'portOrPortType', and 'operation', according to the rules presented in [11].

The 'extends' attribute specifies the name of the SO the given SO extends through single inheritance. In Fig. 2, the service offering SO2 extends some service offering SO1 defined in the same WSOL file.

The 'accountingParty' attribute determines the **accounting party**—a special management party responsible for keeping track of the use of the provider Web Service and management third parties, as well as what constraints were satisfied and what were not. Further, it calculates prices and penalties to be paid. In addition, the accounting party is responsible for evaluation of all constraints for which management responsibility is not specified explicitly through management responsibility statements. Every SO has exactly one accounting party. The provider, a management third party, or, in rare cases, the consumer can play the role of the accounting party. In Fig. 2, the accounting party for SO2 is the provider (supplier) Web Service.

There are several other optional attributes of the `<serviceOffering>` element [12], not illustrated in Fig. 2. These attributes are used to specify duration of the validity period or expiration time of the SO, subscription duration and price, and whether the provider should ask for consumer confirmation before switching from this WSOL SO.

2.2. Constraints and expressions

In WSOL, every **constraint** contains a Boolean expression that states some condition to be evaluated. The constraints can be evaluated before and/or after invocation of operations or at

```

<wsol:WSOLdefinitions ... xmlns: buyStock =
  " http://www.buyStockProvider.ca/buyStockFile.wsdl " ... >
...
<wsol:serviceOffering name = " SO2 "
  service = " buyStock: buyStockService "
  extends = " tns: SO1 "
  accountingParty = " WSOL-SUPPLIERWS " >
  <wsol:constraint xsi:type=" qosSchema: qosConstraint "
    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: statement xsi:type= " managementResponsibilitySchema:
  managementResponsibility " name = " MangRespl " >
  <wsol:supplierResponsibility scope = " tns: AccRght1 " />
  <wsol:consumerResponsibility scope = " tns: Precond3 " />
  <wsol:independentResponsibility scope = " tns: QoScons2 "
    entity = " http://www.someThirdParty.com " />
  </wsol:managementResponsibility>
</wsol:serviceOffering>
</wsol:WSOLdefinitions>

```

Fig. 2. Parts of an example WSOL service offering.

particular date/time instances. WSOL enables the formal specification of:

1. **Functional constraints.** These constraints define conditions that a functionally correct operation invocation must satisfy. They usually check some characteristics of message parts of the invoked operation. WSOL enables specification of pre-, post-, and future-conditions, as well as invariants. The novel concept of a future-condition is introduced to model conditions evaluated some time after the provider finishes execution of the requested operation and sends
2. **QoS (non-functional, extra-functional) constraints.** These constraints check whether the monitored QoS metrics are within specified limits. They can be checked for a particular operation invocation or periodically, at specified times.
3. **Access rights.** An access right specifies conditions under which any consumer using

results to the consumer. It enables specification of operation effects that cannot be easily expressed with post-conditions. An example is delivery confirmation for goods bought using Web Services.

the current SO has the right to invoke a particular operation. For example, an access right can limit the time of day when an operation can be invoked. It can also limit the number and/or frequency of invocations. Access rights are used in WSOL for service differentiation. On the other hand, specification of conditions under which a particular consumer (or a class of consumers) may use a SO and other security issues are outside the scope of WSOL.

For the specification of QoS constraints, WSOL uses external ontologies of QoS metrics and measurement units. Such ontologies contain precise definitions of how the QoS metrics are measured and/or calculated and how the QoS metrics (or the measurement units) relate to each other. These external ontological definitions can be reused for different Web Services. We have summarized requirements for such ontologies in [22]. Ideally, appropriate standardization bodies would develop such ontologies and make them well known. Such standardization is useful because existence of multiple ontologies for the same concepts reduces interoperability. However, since standardization process is long and tedious, it is practical to allow that other interested parties can also define and publish ontologies to be used until standards are completed. Once a QoS metric is defined in an external ontology, a WSOL file declares its use through the ‘metricType’ attribute of QoS constraints.

WSOL constraints are defined using the <constraint> element, which is independent of particular types of constraint. The ‘type’ attribute of the <constraint> element refers to an XML schema defining a particular type of constraint. We have defined XML schemas for the above-mentioned types of constraint. Using the XML Schema mechanisms, additional types of constraint can be defined. An example WSOL constraint, the QoS constraint QoScons2, is shown in Fig. 2. This QoS constraint contains a comparison of a measured QoS metric ResponseTime and the constant ‘0.3s’. The values for the applicability domain attributes ‘service’, ‘portOrPortType’, and ‘operation’ indicate that

this QoS constraint is evaluated for every operation of the ‘buyStockService’ Web Service, because this is the Web Service for which the containing service offering SO2 is evaluated. Further explanation of WSOL constants such as ‘WSOL-ANY’ and ‘WSOL-EVERY’ can be found in [10,11]. The value ‘WSOL_INTERNAL’ for the ‘measuredBy’ attribute of the QoS states that the entity that evaluates this QoS constraint also measures the response time QoS metric.

Boolean expressions in constraints can contain standard Boolean operators, references to operation message parts of type Boolean or other WSOL Boolean expressions, and comparisons of arithmetic, string, date/time, or duration expressions. WSOL also supports checking operation message parts that are arrays of any data type using quantifiers ‘ForAll’ and ‘Exists’. Arithmetic expressions can contain standard arithmetic operators, arithmetic constants, and references to operation message parts or WSOL expressions of numeric data types. WSOL provides only basic built-in support for string and date/time/duration expressions.

In addition, it is possible to perform operation calls in any expression. The called operations can be implemented by the Web Service for which the constraint is evaluated, by another Web Service, or by the management entity evaluating the given constraint. In the latter case, although these external operations are described with WSDL, they are invoked using internal mechanisms, without any SOAP call.

Different types of constraints have similar syntax and require similar monitoring infrastructure. Further, management statements, such as prices/penalties and management responsibility statements, relate to all constraints and not a particular type of constraint, such as QoS. Consequently, there is less overhead in supporting one language for various types of constraint, such as WSOL, than several separate languages for QoS constraints, functional constraints, access rights, and possibly other constraints and management statements. This topic was discussed in more detail in [9].

2.3. Management statements

A WSOL **statement** is any construct, other than a constraint, that states important management information about the represented class of service. For the formal specification of statements, WSOL contains the general `<statement>` element. It is analogous to the general `<constraint>` element discussed in the previous subsection. The ‘type’ attribute of the `<statement>` element refers to the XML schema defining a particular type of statement. We have defined XML schemas for the following types of management statement:

1. pay-per-use price statements,
2. monetary penalty statements, and
3. management responsibility statements.

Additional types of statement can be supported in WSOL by defining XML schemas for them.

A **pay-per-use price statement** states the monetary amount a consumer using the particular SO has to pay for invoking particular operation. **Penalty statements** specify the monetary amount that the provider Web Service has to pay to a consumer if the consumer invokes some operation and the Web Service does not fulfil all constraints in the SO. A penalty can be stated for not meeting all constraints for an operation and/or for not meeting a particular constraint. Currency units used in price and penalty statements are defined in external ontologies. A **management responsibility statement** specifies what entity (the provider, the consumer, or a management third party) has the responsibility for checking a particular constraint, a constraint group (CG), or the complete SO. As will be discussed in Section 3.1, these management parties act as SOAP intermediaries. Due to the specifics of accounting parties, it is not possible to use a management responsibility statement to specify the accounting party for a SO. Fig. 2 shows an example WSOL management responsibility statement, `MangResp1`. In this example, the provider (supplier) Web Service evaluates the access right `AccRght1`, the consumer evaluates the pre-condition `Precond3`, and the third party evaluates the QoS constraint `QoScons2`.

2.4. Reusability elements

When multiple classes of service, SLAs, or other comprehensive contracts are specified, there is often a lot of similar information that differs in some details. For example, SLAs that a telecommunication service provider has with its customers often have similarities. Analogously, two classes of service for the same Web Service can be the same in many elements, but differ only in response time and price. Expressive reusability constructs in WSOL enable easier specification of new SOs from existing SOs of the same Web Service or other Web Services. They can also be helpful in determining similarities and differences between SOs in the process of selection of Web Services and SOs.

WSOL files can contain several special **reusability elements**:

1. the definition of constraint groups (CGs),
2. the inclusion of already defined constraints, statements, or CGs,
3. the definition of constraint group templates (CGTs),
4. the instantiation of CGTs, and
5. the declaration of operation calls.

A CG is a named set of constraints, statements, and reusability elements (including nested CGs). The inclusion element enables reusing constraints, statements, and CGs that have been previously defined. A CGT is a parameterized CG. In a definition of a CGT, parameters are declared and used within the contained constraints. In an instantiation of a CGT, concrete values are supplied for all parameters. A declaration of an operation call is used to enable referencing results of the same operation call in several related constraints.

Further, WSOL defines several **reusability attributes**. Most importantly, the ‘extends’ attribute can be used to specify extension (single inheritance) of SOs, CGs, and CGTs. Reusability constructs in WSOL were discussed in detail in [11].

2.5. Service offerings dynamic relationships (SODRs)

Dynamic (run time) **relationships between SOs** are those that can change during run-time, e.g., after dynamic creation of a new class of service. For example, one such dynamic relationship can state what class of service could be an appropriate replacement if a particular constraint from some other class of service cannot be met. On the other hand, **static** (development-time) **relationships between SOs** show similarities and differences between SOs that do not change during run time. Two important examples of such relationships are extension (single inheritance) of SOs and instantiation of the same CGT with different parameter values. Both static and dynamic relationships between SOs are useful for easier selection and negotiation of SOs. In addition, dynamic relationships are crucial for the mechanisms for dynamic adaptation of Web Service compositions discussed in Section 4.2.

In WSOL, static relationships between SOs are expressed with the reusability elements and attributes, such as extension, inclusion, and instantiation. They are built into definitions of SOs. Conversely, WSOL defines a special construct, the **service offerings dynamic relationship (SODR)**, for dynamic relationships. SODRs are specified outside definitions of SOs, often in separate files. In this way, a change in a SODR does affect descriptions of SOs.

A SODR can be specified as a triple $\langle \text{SO1}, \text{S}, \text{SO2} \rangle$ or as a triple $\langle \text{SO1}, \text{E}, \text{SO2} \rangle$ [12]. Here:

1. SO1 is the used SO,
2. (a) S is the set of constraints from SO1 that are not satisfied;
(b) E is a Boolean expression relating unsatisfied constraints from SO1, and
3. SO2 is the appropriate replacement SO.

If the format $\langle \text{SO1}, \text{S}, \text{SO2} \rangle$ is used, then SO2 is the appropriate replacement for SO1 if all constraints from S are unsatisfied. If the format $\langle \text{SO1}, \text{E}, \text{SO2} \rangle$ is used, then SO2 is the appropriate replacement for SO1 if the expression E is true. The former format is simpler and easier

to monitor during run time, so it was the only format for SODRs in early versions of WSOL [10]. In the new WSOL version 1.1 we have added the latter format, which is more powerful and can be used instead of the older format.

3. Using WSOL in Web Service Management

We concentrate our research of WSM on monitoring of WSOL SOs. This includes measurement and calculation of used QoS metrics, evaluation of WSOL constraints, and accounting of executed operations and evaluated WSOL constraints. In this section, we illustrate monitoring of WSOL SOs with an example configuration of management parties and discuss how our WSOL implements monitoring activities inside a management party.

3.1. Web Service monitoring with several management parties

Management third parties that are specified inside WSOL management responsibility statements, as well as independent accounting parties, act as **SOAP intermediaries**. In other words, they intercept SOAP messages between the consumer and the provider, perform their management tasks, and piggyback their management results into SOAP headers.

Fig. 3 shows an example configuration of management third parties as SOAP intermediaries. In this example, only QoS constraints are evaluated and distinction is made between the accounting party, the QoS metering party, and the QoS constraint evaluation party. In this figure, the arrows represent passing of information and control between different management parties, achieved through sending of SOAP messages. When a consumer submits a request for executing a provider's operation, the management third parties are organized as SOAP intermediaries for the request, as well as the response message. The request from the consumer to the provider first goes through the accounting party, which logs that the request has been made. The request is then forwarded to the QoS metering party, which

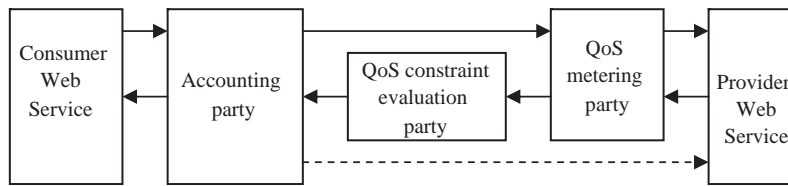


Fig. 3. An example configuration of management third parties as SOAP intermediaries.

performs necessary activities. For example, for metering response time the QoS metering party logs the time that will be considered as the beginning time of the operation invocation. Next, the request message is forwarded to the provider Web Service, which performs the requested operation and sends the response message. The response message first goes through the QoS metering party. In the response time measurement example, the QoS metering party logs the time that will be considered as the ending time of the operation invocation and subtracts from it the logged beginning time. One way to transport to the QoS constraint evaluation party the information about the measured QoS metrics is to use special SOAP headers as additions to the original response message from the provider. The QoS constraint evaluation party receives the response message and the information about the measured QoS metrics and evaluates appropriate constraints. It forwards to the accounting party the response message together with the information whether the evaluated constraints were satisfied or not and appropriate details if some constraints were violated. The accounting party logs the received management information, calculates prices and/or penalties to be paid, and forwards the response message and management information to the consumer. It can also notify the provider with appropriate details, particularly if some QoS constraints were not satisfied. This can help the provider to adapt its behavior to meet guarantees for future operation requests. The additional SOAP message from the accounting party to the provider shown in Fig. 3 is optional. This is because there are a couple of other ways to transport management information from the accounting party back to the provider. One of them is piggybacking the next SOAP message from

the consumer to the provider, while another is periodic sending of reports.

The routing of SOAP messages between the consumer and the provider through SOAP intermediaries can be implemented in several ways. In our WSOI prototype, we have used a simple approach in which the route is stored in a SOAP header and the WSOI code in the consumer invokes the accounting party instead of the provider. The accounting party extracts the route from the SOAP header and invokes the next SOAP intermediary. The process is repeated until the route is completed.

There are other Web Service monitoring scenarios that can be accommodated with WSOL. For example, WSOL has two features that can reduce the run-time overhead of monitoring activities, at the cost of less management information. First, WSOL constraints checked before and/or after operation invocations can be evaluated **occasionally**, only for some randomly chosen invocations (on average: 1 in n). This is specified with the optional attribute 'evalPeriod' of QoS constraints. For example, when this attribute is set to 5, this means that this QoS constraint is checked, on average, for one operation invocation out of five. To maximize usefulness of such constraint evaluations, the provider must not know in advance for which operation invocation the QoS metrics will be measured and/or the constraint will be evaluated. This is achieved when an independent accounting party randomly chooses between operation invocations and monitoring is performed only by third parties. To make up for cases when unsatisfied constraints are not checked, monetary penalties for not meeting such constraints should be relatively high.

On the other hand, some QoS constraints can be evaluated **periodically** at particular date/time

instances, independently from particular operation invocations. This is useful for checking QoS metrics such as average response time and throughput. The accounting party initiates calculation of periodic QoS metrics and evaluation of periodic QoS constraints.

In addition, WSOL supports management third parties that act as **probes**. Some QoS metrics, such as availability, can be measured using probing instead of message interception [3]. WSOL supports this by modeling probing entities as separate Web Services that provide results of their measurements through operations of agreed-upon port types (interfaces). These operations can be invoked in appropriate QoS constraints in WSOL SOs using the WSOL operation call mechanism. Alternatively, the probing entities can send their results to other management parties using operations of the special notification port type, discussed in Section 4.1.

3.2. Web Service monitoring inside WSOI

A management party performs its Web Service monitoring activities using WSOI. WSOI modules can be classified into modules that are used for monitoring of WSOL SOs, modules used for dynamic manipulation of WSOL SOs, and modules used for both purposes [21]. The part of WSOI that performs monitoring of WSOL SOs is based on extensions of **Apache Axis** (Apache eXtensible Interaction System) [23,24], 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 several good features that we found crucial for WSOI. Here we emphasize the two most important. First, Axis has a modular, flexible, and extensible architecture based on configurable chains of pluggable SOAP message processing components, called handlers. Such architecture enables implementing monitoring of WSOL SOs using a set of additional handlers and handler chains that are plugged into Axis easily. Second, Axis defines a SOAP message processing node that can be used for provider Web Services, consumer Web Services, and SOAP intermedi-

aries. Consequently, it can be used for all WSOL management parties.

An Axis **handler** [23,24] can process input, output, and/or fault SOAP messages. For example, it can perform measurement of QoS metrics. A handler 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. There are three types of chain in Axis. A transport chain performs processing related to the transport of SOAP messages. A global chain performs other processing applicable to all Web Services. A service chain performs processing characteristic for a particular Web Service. Axis handlers exchange information through an instance of the **MessageContext** class, which contains information about the request message, the response message, and a set of properties. Axis handlers can use the MessageContext properties for decisions related to message processing. They can also modify these properties.

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**'. 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 proof-of-concept prototype of WSOI. Some design decisions for WSOI-specific handlers were discussed in [20].

In different invocation contexts, different WSOI-specific handlers are used. An **invocation context** is determined by the full name of the invoked operation (containing Web Service, port, and operation names), the name of the SO, the name of the management party in which this handler is used, and whether the processed message is request, response, or fault. In WSOL files, constraints and management statements are associated with different invocation contexts through applicability domain attributes, containment of constraints and management statements inside SOs, management responsibility statement,

and the ‘accountingParty’ attribute of SOs. We have developed a special XML file format, called the Web Service Offering Descriptor (WSOD), for describing the order of WSOI-specific handlers used in a particular invocation context. WSOD files can be generated by a WSOL compiler. In addition, we allow human Web Service administrators to generate or modify WSOD files. WSOI loads this ordering information from WSOD files into internal data structures.

The **WSOIChain** class is the crucial WSOI module for the monitoring of WSOL-enabled Web Services. It contains code that dynamically constructs the chain of appropriate WSOI-specific handlers for the given invocation context, according to the contents of the internal data structures loaded from WSOD files. In this way, different WSOI-specific handlers are used for different WSOL SOs and/or different operations. An instance of the WSOIChain class is a sub-chain of the Axis’ service chain.

Several WSOI-specific handlers are used by all management parties. For efficiency reasons, they are implemented outside the WSOIChain chain. One of them is WSOISessionHandler (SH) that performs session management activities. Further, the ServiceOfferingInput (SOI) and ServiceOfferingOutput (SOO) exchange the management information between MessageContext properties and SOAP headers. While MessageContext properties are used to transport WSOL-related management information between WSOI modules, SOAP headers are used to transport this information between management parties.

MessageContext properties only store monitoring information for the latest invocation. Therefore, WSOI additionally stores descriptions of WSOL constructs and the actual measured or computed data into several **WSOI-specific data**

structures, often implemented as hashtables. For example, these data structures store information about which SO is used in a session, the lists of active and deactivated SOs, the history of unsatisfied and satisfied constraints, the billing balance for a particular session, and the history of prices and monetary penalties incurred in a session. Most of these data structures are also used for the mechanisms for manipulation of SOs described in Section 4.2. Further information about these data structures and other WSOI modules is given in [21].

Let us now illustrate how WSOI-specific handlers are used for the monitoring of WSOL-enabled Web Services. In Fig. 4, we have shown an example configuration of handlers inside the provider-side WSOI. The example is analogous to the example in Fig. 3, but in Fig. 4 all shown modules are parts of the provider Web Service. In other words, the provider Web Service measures response time, evaluates a QoS constraint limiting response time, and performs accounting. In this figure, the arrows represent passing of information and control between internal modules, achieved through local procedure calls. The Transport-Listener (TL) component of Axis receives the request (input) SOAP message and passes it to the standard Axis handler called Deserializer (DS), which creates a MessageContext instance used by other handlers. After DS, several other standard Axis handlers used for all Web Services are executed [24]. Due to space limits and the fact that these handlers are not crucial for the illustration of WSOI, they are not shown in Fig. 4. Instead, their position is indicated with ‘...’. For every Web Service that supports WSOL, the WSOI-specific handlers SH, SOI or SOO, and the chain WSOIChain are executed. For processing input messages, SH reads the session

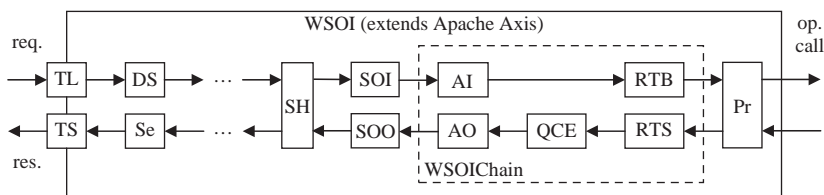


Fig. 4. An example configuration of handlers and message processing inside provider-side WSOI.

information from a SOAP header and writes it into a MessageContext property. Similarly, SOI reads WSOL-related information from SOAP headers and writes it into appropriate MessageContext properties. The first handler in WSOIChain for input message is AccountingInput (AI), which records the request message. Then, the ResponseTimeBegin (RTB) handler stores into a MessageContext property 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 Pr handler. After Pr, handlers in WSOIChain process the response (output) message. First, the handler ResponseTimeStop (RTS) stores into a MessageContext property 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 QoSConstraintEvaluation (QCE). This handler stores its results into a MessageContext property. Finally, the Accounting-Output (AO) handler uses the information from MessageContext properties to calculate prices and/or penalties to be paid. This information is also stored into other MessageContext properties. After WSOIChain, SOO and SH read WSOL-related information from MessageContext properties and write them into SOAP headers. Next, several standard Axis handlers used for all Web Services [24] are executed. These handlers are not shown in Fig. 4, but their position is indicated with ‘...’. The last of these handlers is Serializer (Se), which packs information from the MessageContext instance into SOAP. The TransportSender (TS) component of Axis sends the response SOAP message to the consumer. The information about the measured response time, the evaluated constraint, and, possibly, prices or penalties to be paid are in the SOAP header.

When for another SO and/or for another operation different QoS metrics have to be measured, different WSOL constraints evaluated, and/or different prices or penalties calculated, then only the contents of the WSOIChain changes

appropriately. The configuration of all other handlers remains the same.

The measurement or calculation of periodic QoS metrics and evaluation of periodic constraints differs from the example in Fig. 4. As already mentioned, it is initiated by the accounting party. If the provider acts as the accounting party, the periodic monitoring activities are initiated by **Timer**, a special active module in WSOI. Timer invokes WSOIChain, which creates and executes a chain of appropriate WSOI-specific handlers. The results of such measurement, calculation, and/or evaluation can be stored locally for future processing. They can also be reported to other management parties in a special notification SOAP message.

We have performed a number of **experiments** to demonstrate that WSOL and WSOI can be used for monitoring of Web Services and to estimate the overhead that such monitoring places on Web Services. The experiments compare average response time and average Java Virtual Machine (JVM) memory usage when Web Services are hosted with Axis and when they are hosted with WSOI. Tosic et al. [21] describe one such experiment in which the use of WSOI increased response time 15% and memory usage 4%, compared to response time and memory usage when only Axis was used. In our opinion, this is an acceptable increase, because it is significantly low for many consumers and many circumstances. Further, in the mentioned experiment the provider and the consumer Web Services executed on different computers in a local network. When Web Services are distributed over the Internet, the network delay is higher, so the relative WSOI increase of the total response time is lower.

4. Using WSOL in Web Service Composition Management

4.1. Using WSOL for configuring Web Service compositions

Let us now explain how a consumer who already knows about the appropriate provider Web Service can choose one of the provider’s SOs.

This process is part of the configuration of Web Service compositions.

For monitoring and manipulation of WSOL SOs, we have defined WSDL signatures of a number of operations grouped into several **Service Offering Management (SOM) port types** [20,21]. For example, the SONotification port type defines operations for notifications and other exchange of WSOL-related management information between management parties during monitoring of WSOL-enabled Web Services. Several characteristics of the SOM port types should be noted. First, the definition of well-known (ideally: standardized) management operations significantly reduces complexity of management activities. Several other works, such as WSLA [13], Open Grid Services Architecture (OGSA) [25], and WS Manageability [26] also define management operations for Web Services. Second, the advantages of defining management port types instead of separate management Web Services are easier discovery and easier support for only some SOM port types and operations in them. While WSLA defines separate management Web Services, WSOL, WS Manageability, and OGSA define management ports. The concept of management interfaces (port types) was also advocated and explored in several earlier works on the management of distributed services, e.g., [27]. Third, a management party (e.g., a provider) might implement only some of these management port types and/or only some operations within a particular port type. Fourth, only selected external entities are allowed access to the majority of operations from SOM port types. These entities are primarily the other management parties in the composition, but potentially also special management software or human administrators. Fifth, some or all Web Services of the same vendor (e.g., Web Services using the same SOAP engine instance) can share the actual implementation of SOM port-type operations. This is because Web Services do not describe implementation. In such cases, the overhead of supporting the SOM port types is lower.

In the process of setting up a SO to be used between a consumer and a provider, several operations from SOM port types are used. At the very beginning, the consumer has to open a

session with the provider by invoking the provider's implementation of the openSession() operation from the SessionMgmt port type. This operation creates a new session and returns the session ID (identifier) to the consumer. When the consumer invokes the listActiveSOsForMe() operation from the SOInfo port type, the provider returns a WSOL file containing descriptions of all active SOs that this consumer is allowed to use. To compare SOs, the consumer can invoke operations from the SOComparisons port type. These SOComparisons operations can be implemented by the consumer, the provider, or some specialized helper Web Service. When the consumer decides which SO is most appropriate, it invokes the startWithSO() operation of the SOM-Prov port type.

Note that the negotiation between the consumer and the provider is very simple—the provider suggests several existing SOs and the consumer chooses one of them. While this is less powerful than negotiation of custom-made SLAs, it requires much simpler program code for decision making. In addition, selection of SOs can be done much faster than negotiation of custom-made SLAs, primarily because of the smaller number of exchanged SOAP messages. To conclude, while this approach has limitations, it also has advantages because of its **relative simplicity, speed, and low overhead**. As will be mentioned in the next subsection, we also allow dynamic creation of new SOs in exceptional cases.

In the next subsection, we will mention SOM port types used for the manipulation of WSOL SOs, while further information about all SOM port types can be found in [20,21].

4.2. Using WSOL for dynamic adaptation of Web Service compositions

In the WSCM area, we are investigating management and dynamic adaptation of Web Service compositions without breaking an existing relationship between a provider Web service and its consumer. For example, the adaptability of the relationship between the stock notification Web Service and the financial analysis Web Service might be a very valuable feature in a turbulent stock market. To achieve this goal we are

exploring mechanisms for the manipulation of WSOL SOs. The crucial WSOL language support for these mechanisms is the specification of dynamic relationships (SODRs) between service offerings, discussed in Section 2.5. The main five **dynamic manipulation mechanisms** we have studied are [20]:

1. switching (initiated by the consumer or the provider Web Service),
2. deactivation,
3. reactivation,
4. deletion, and
5. creation of SOs.

These mechanisms can be used between operation invocations inside one session. We illustrate the benefits of these mechanisms by extending the e-business example with provider stock notification and consumer financial analysis Web Services, introduced in Section 2.1.

Dynamic switching between SOs means changing which SO a consumer uses. Either a consumer or a provider can initiate it. In the latter case, the consumer is notified about the change. In addition, if the ‘autoManipulation’ attribute of the old SO is set to ‘False’, the consumer is asked for confirmation of the switching before it is executed. The consumer can initiate switching to dynamically adapt the service and/or QoS it receives without searching for another provider. For example, depending on the analysis of the current situation in a stock market, the financial analysis Web Service could want to dynamically switch between different SOs of the stock notification Web Service it consumes. Fig. 5a shows a case when the financial analysis Web Service FA1 switches from service offering SO1 to service offering SO2 of the same stock notification Web Service SN1. In this

example Web Service composition, FA1 is the consumer, while SN1 is the provider. In [20], we discussed a more detailed example of consumer-initiated switching between SOs and illustrated it with a UML sequence diagram. The provider can initiate switching to gracefully upgrade or degrade its service and/or QoS in case of changes. For example, after dynamic creation of a new SO, the stock notification Web Service can suggest some of its consumers to switch to the new SO. Switching between SOs is the basic adaptation mechanism in our research.

Deactivation of SOs is used by a provider Web service when changes in operational circumstances affect what SOs it can provide to consumers. Some SOs cannot be used in all circumstances. For example, it is sometimes impossible to achieve high QoS or it is dangerous to provide SOs 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. When such change of circumstances occurs, a Web service can dynamically and automatically deactivate SOs that cannot be supported in the new circumstances. The affected consumers are switched to an appropriate replacement SO and notified about the change. In addition, if the ‘autoManipulation’ attribute of the deactivated SO is set to ‘False’, the consumers are asked for confirmation of the switching to the replacement SO. If there is no appropriate replacement SO, the provider can initiate creation of new SOs. When this is also not possible, an alternative provider Web Service has to be sought. This dynamic adaptation mechanism supports both graceful degradation and seamless service upgrades and expansions.

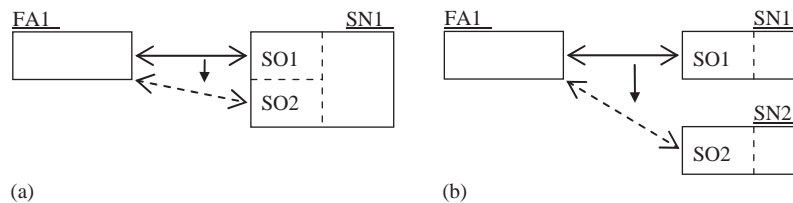


Fig. 5. (a) Switching between SOs; and (b) Switching between Web Services.

The deactivated SO 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 SOs. 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 SO**. For example, if implementation of the stock notification Web Service is dynamically upgraded to improve performance, some of its SOs with lower QoS might become redundant and can be deleted. Another example is when a management third party used in some SO goes out of business. Deletion permanently removes support for an SO. Only deactivated SOs can be deleted.

Dynamic creation of new SOs 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. While the dynamic adaptation mechanisms discussed above handle changes that are to some extent anticipated, the creation of new SOs can be used for unanticipated changes. For example, when the implementation of the stock notification Web Service is dynamically upgraded, then new SOs of this Web Service can be created. To reflect these changes, new SOs 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 creation of new SOs is not creation of new functionality or new functional interfaces, but creation of new sets of constraints and management statements for the existing functionality. However, dynamic creation of new SOs 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 SOs as variations of existing SOs, based on WSOL reusability elements and attributes. While we concentrate on provider-initiated creation of SOs, we also leave the possibility of consumer-initiated creation in special cases. In the latter cases,

dynamic creation of a new SO can be viewed as a substitute for negotiation of a custom-made technical contract or SLA between Web Services because it involves matching consumer's preferences with provider's capabilities.

Manipulation of SOs can be followed by appropriate deactivation, reactivation, or creation of SODRs. The mechanisms for dynamic manipulation of WSOL SOs were discussed in detail and illustrated with further examples in [20].

For the implementation of these dynamic adaptation mechanisms, we have developed appropriate algorithms and protocols. The **algorithms** for the manipulation of SOs contain program logic deciding whether, what, how, and when the manipulation of SOs should be performed. In most cases, these algorithms are executed in provider-side WSOI, particularly in the special **SOMgmtDecisions module**, when certain conditions are met, without explicit external intervention. They use the WSOI-specific data structures that are also used for Web Service monitoring. For example, the algorithm for deactivation of SOs first checks the history of unsatisfied and satisfied constraints to determine whether deactivation is appropriate. Then, it uses several data structures, including the information about the currently used SO and descriptions of active SODRs, to determine to which SO to switch. Next, it invokes the algorithm and protocol for the provider-initiated switching of SOs for affected consumers. Further, it handles possible special cases, such as the possibility that some consumers reject the suggested replacement SO. After all these activities are performed, this algorithm invokes the algorithm for deactivation of affected SODRs. On the other hand, the **protocols** for the manipulation of SOs govern the coordination of management parties to achieve the manipulation. The operations that participate in these protocols are defined in SOM port types SOM-Prov, SOM-Cons, SOM-AccP, SOM-MgmtP, and SODRMgmt.

To conclude, WSOI implements the discussed mechanisms for manipulation of WSOL SOs with several data structures, the SOMgmtDecisions module, and several modules that implement SOM port types. Unlike the Web Service monitoring

modules discussed in Section 3.2, these modules are not based on Apache Axis. The details of these modules were presented and illustrated in [21].

We have performed a number of analytical studies and practical experiments comparing the manipulation of SOs with alternative approaches to dynamic adaptation of Web Service compositions. The main alternatives are re-composition of Web Services and re-negotiation of SLAs. Fig. 5b illustrates switching between Web Services, which is a special case of re-composition. In this example, the financial analysis consumer Web Service FA1 is the consumer. SN1 and SN2 are two stock notification providers implementing the same WSDL port types, but providing different service offerings SO1 and SO2. To achieve dynamic adaptation, FA1 switches from SN1 with the service offering SO1 to SN2 with the service offering SO2. Our **analytical studies** [20] compared the number of exchanged SOAP messages in different dynamic adaptation approaches. This is because internal operations performed by parties involved in dynamic adaptation are relatively simple and fast compared to the generation, transmission, and processing of SOAP messages. On the other hand, in our **experiments** [20,21] we measured average delay and average JVM memory usage.

These experiments and analytical studies showed that manipulation of SOs is generally **simpler, faster, and incurs less run-time overhead** than the re-composition of Web Services and the re-negotiation of SLAs. Further, it provides additional flexibility and enhances robustness of the relationship between a provider Web Service and its consumer. This robustness is important, e.g., 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 SOs has **limitations**. SOs of one Web Service differ only in constraints and management statements, which might not be enough for adaptation. Further, appropriate alternative SOs cannot always be found or created. Therefore, manipulation of SOs 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 relatively similar limitations.

Consequently, we suggest that a management system for dynamic adaptation of Web Service compositions integrates the manipulation of SOs and the re-composition of Web Services. The first step in such dynamic adaptation is to try to find a replacement SO from the same Web Service. If this is not possible, the second step is to try to find a replacement Web Service and perform re-composition. In some cases, the used provider Web Service can supply a temporary replacement SO 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.

5. Related work

The most important related works to WSOL are two languages for the formal XML-based specification of custom-made SLAs for Web Service: the WSLA [3,13] from IBM and HP's WSML [6,14]. These languages do not directly support the concept of classes of service. SLAs in these two languages contain QoS guarantees and management information such as management responsibilities and prices. Since WSOL SOs also specify QoS constraints (guarantees and requirements), they can be viewed as simple SLAs or other technical contracts. Let us outline some of the differences between WSOL SOs and SLAs in WSLA and WSML. Custom-made SLAs have to explicitly specify information about consumers, which is implicitly assumed in WSOL. In addition, SLAs in WSLA and WSML contain more detail for QoS constraints than WSOL SOs. For example, they contain detailed description of management action guarantees, while WSOL only describes monetary penalties to be made. Similarly, their description of schedules that is more detailed than information provided in WSOL periodic QoS constraints. In these aspects, WSLA

and WSML are more powerful than WSOL. Further, WSLA and WSML define used QoS metrics within SLAs, while WSOL outsources such definition to external ontologies. It seems that all this results in higher run-time overhead of WSLA and WSML than the overhead of the simpler WSOL. On the other hand, WSOL SOs can contain formal specification of functional constraints and access rights, which are not specified in SLAs. Another advantage of WSOL is the broad set of powerful reusability constructs [11]. Both WSLA and WSML are oriented towards management applications in inter-enterprise scenarios. It seems that they assume existence of some measurement and management infrastructure at both ends. This is a different assumption from the one that we have adopted for WSOL. These languages are accompanied by appropriate management infrastructures. These infrastructures are more powerful, but also more complex, than our WSOL. To conclude, while both WSLA and WSML are very good languages for their domain and purpose, they do not address all the issues that WSOL does.

Another language that can be used for specification of SLAs for Web Services is SLAng [15]. SLAng enables specification of SLAs on the Web Service level and several lower levels, so it has broader scope than WSLA, WSML, and WSOL. However, the definitions of QoS metrics are built into SLAng schema, so SLAs have predefined format. Since the current version of SLAng lacks flexibility and power, it seems less well suited for Web Services than WSLA, WSML, and WSOL. The work on a management infrastructure for SLAng is in progress.

WS-Policy [16] is 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 expres-

sions 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 information 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, some of the advantages of WSOL are the explicit support for management applications, built-in support for various categories of constraints and management statements, unified representation of expressions, wider range of reusability constructs, and specification of classes of service and relationships between them.

Several other recent works also 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 [18] community works on semantic descriptions of Web Services, including specification of some functional and some QoS constraints. However, these specifications are not yet precise and detailed enough for the monitoring and management activities. While DAML-S has the concept of a service profile, there is no concept of a class of service and no SODRs. The OGSA [25] community also recognizes the need for formal specification of contracts, SLAs, and constraints. It has been working on WS-Agreement, previously known as OGS-Agreement [19]. However, a Grid Service is a very special Web Service and it is not yet clear how the future results from the OGSA community will relate to general Web Services. In addition, several research projects related to the description and discovery of QoS for Web Services, such as the UX [17] extension of UDDI, have been started recently. However, these works are not yet accompanied by research of management-related issues and appropriate management infrastructures. On the other hand, several recent papers and commercial products focus on particular areas of WSM and WSCM, often performance or security. However, these management solutions cannot be transferred or generalized to other management areas. In addition, they do not address comprehensive description of management information for Web Services.

6. Conclusions and future work

Describing a Web Service in WSOL, in addition to WSDL is useful for both WSM and WSCM applications.

The crucial WSOL support for WSM applications is the **formal and unambiguous specification of various types of constraint and management statement**, in a format that can be used for automatic generation of constraint-checking code. WSOL describes for Web Services the QoS metrics to measure or calculate, the constraints (requirements and guarantees) to evaluate, when and where and, to some extent, how to perform these monitoring activities, and what are the monetary consequences of meeting or not meeting the constraints. This management information is the basis for both Web Service monitoring and control of the Web Service to meet its guarantees. Different types of constraint and management statement are useful in different management areas. QoS constraints are particularly useful in performance management. Functional constraints are beneficial in fault management, particularly to determine whether a Web Service behaves correctly. Access rights limit access to operations and ports of a Web Service. While WSOL access rights are used primarily for service differentiation, they could also be a small part of a comprehensive security management solution for Web Services. Statements about pay-per-use prices and monetary penalties are valuable in accounting management, particularly billing. Management responsibility statements can be used in configuring Web Services and their compositions. While WSOL constraints and statements can be viewed as two categories of management policies, WSOL SOs formally represent classes of service and can be viewed as simple SLAs or technical contracts between providers, consumers, and potential management third parties.

Web Service monitoring activities can be performed by the provider Web Service, the consumer, and/or one or more mutually trusted third parties (SOAP intermediaries or probes.) Management third parties and the specific accounting party can be explicitly designated in WSOL files. The accounting party is stated in an attribute of the XML element for a SO. A management third

party that measures or calculates a QoS metric is indicated in an attribute of the declaration of use of this QoS metric. Management third parties that act as SOAP intermediaries are specified in management responsibility statements, while a party that acts as a probe is specified through an attribute of the appropriate operation call.

Other attributes of WSOL constructs are also relevant for management. The subscription and validity period attributes of SOs are useful for accounting management. The applicability domain attributes determine for which Web Service, port, and operation a WSOL construct or a QoS metric should be monitored. An attribute of QoS constraints can be used to indicate their occasional evaluation, while child elements of periodic QoS constraints are used to specify evaluation times.

The crucial WSOL language support for dynamic adaptation of Web Service compositions based on the manipulation of SOs is the **specification of relationships between SOs**. In particular, WSOL contains the built-in format for the specification of SODRs. These relationships are essential for switching (particularly provider-initiated), deactivation, and reactivation of SOs. On the other hand, WSOL reusability elements and attributes determine static relationships between SOs. They can be used for dynamic creation of new SOs.

The work on WSOL is accompanied with the development of WSOI and the research of the mechanisms for the dynamic manipulation of SOs. WSOI demonstrates **feasibility and usefulness** of using WSOL for management applications. It enables monitoring of WSOL-enabled Web Services and dynamic manipulation of their SOs. For Web Service monitoring, we have extended the Apache Axis open-source SOAP engine with WSOI-specific modules, data structures, and management ports. To support dynamic manipulation of SOs, we have developed appropriate algorithms, protocols, and management port types and built into WSOI modules, data structures, and ports for their implementation.

The use of classes of service for customization of Web Services has limitations. Similarly, the dynamic adaptation of Web Service compositions based on the manipulation of classes of service has

limitations. However, the main competitive advantages of using classes of service and their manipulation for management activities are **relative simplicity, speed, and low run-time overhead**. In many situations, application of our results can be more practical than utilization of solutions based on custom-made SLAs or other alternatives. One example of such a situation is when a provider serves a very large number of consumers (e.g., hundreds or more) in parallel. Another example situation is when the provider executes in resource-constrained environments, such as embedded and/or mobile systems. Both classes of service and the mechanisms for their dynamic manipulation can be used as lightweight complements and additions to the more powerful alternatives.

WSOL and WSOI address **specification, monitoring, and manipulation of classes of service for Web Services**. These issues have not yet been researched by related works. Several recent related works—particularly WSLA, WSML, and WS-Policy—address management-related issues that partially overlap with WSOL. In some aspects, they are more powerful than WSOL. On the other hand, some of the unique characteristics and advantages of WSOL are [9]: support for classes of service and their static and dynamic relationships, formal specification of various types of constraint and statement, a diverse set of reusability constructs, features reducing run-time overhead, and support for both WSM and WSCM applications. Further, WSOL can be extended with support for additional types of constraint and management statement using XML Schema mechanisms, without any changes to the current WSOL grammar. Unfortunately, it seems that we currently do not have resources to push WSOL through a standardization process. Nevertheless, our results from the work on WSOL, WSOI, and the mechanisms for manipulation of WSOL SOs can be **integrated** into future standards and platforms for WSM and WSCM. In particular, we advocate integration of good features from WSLA, WSML, WS-Policy, and WSOL into a future standard for description of Web Services.

Our current work is mainly focused on the further development of WSOI, its prototype implementation, and research of the manipulation

of SOs. While our current WSOI prototype demonstrates the main concepts, the improved prototype will demonstrate the complete system. For example, we have not yet implemented the WSOI support for manipulation of SODRs. Likewise, WSOI currently has only rudimentary support for creation of SOs. We plan further experiments comparing WSOL and languages based on custom-made SLAs. Integration into WSOI of the actual control of Web Services to meet the specified constraints is also left for future work.

While WSOL can be improved in several ways, we consider the language relatively complete and stable. We have recently completed several improvements, described in [12], of the WSOL grammar and parser. For example, we have added specification of complex expressions into SOs dynamic relations, naming and inclusion of expressions, as well as definition of CGs and CGTs in which only some ('at least one' or 'exactly one') constraint have to be satisfied. We have also added additional management-related attributes to SOs and QoS constraints, improved specification of periodic QoS constraints and future-conditions, and made specification of statements more flexible and more consistent. The Premier WSOL parser was improved according to these WSOL grammar changes. We plan full compatibility with the new WSDL version 2.0. In addition, we are considering extending WSOL to achieve compatibility with BPEL4WS. The major WSOL issue related to WSOI is the full implementation of a WSOL compiler to enable automatic generation of WSOI-specific handlers from WSOL files. A Java API for the generation of WSOL files would also be beneficial.

We have left several important areas for future research. One of them is use of WSOL for selection of Web Services and their SOs. The process of comparison, negotiation, and selection of Web Services, their functionality, and particularly QoS is a very complex issue, without a simple and straightforward solution. QoS often cannot be easily compared as it can have many dimensions of difference. For example, when one SO constrains availability and does not limit response time, while another SO does the opposite, it is very difficult to

say which one is ‘better’. On the other hand, as the number of Web Services that offer similar functionality increases, the offered QoS, price, and manageability will become important competitive advantages. Consequently, the comprehensive description of Web Services in WSOL, particularly the explicit representation of static and dynamic relationships between WSOL SOs, can assist in selecting appropriate Web Services and SOs. Therefore, the development of appropriate WSOL-based comparison and selection algorithms and heuristics and their implementation in specialized Web Service brokers seem like promising research topics. A related open issue is integration of WSOL with the existing Web Service discovery and selection technologies, such as UDDI.

In addition, WSOL could be used with security technologies for Web Services. For example, different keys could be used for encryption of SOAP message body and various QoS measurements and constraint evaluation results, so that only relevant management parties would see them. Further research in this area is needed.

References

- [1] J.C. Schlimmer (Ed.), Web services description requirements, World Wide Web Consortium (W3C) working draft, on-line at: <http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028/>, October 28, 2002.
- [2] C. Peltz, Web services orchestration and choreography, *Computer*, IEEE-CS 36 (10) (2003) 46–52.
- [3] A. Keller, H. Ludwig, The WSLA framework: specifying and monitoring service level agreements for Web Services, *J. Network Systems Manag.* 11 (1) (2003).
- [4] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web services description language (WSDL) 1.1, World Wide Web Consortium (W3C) note, on-line at: <http://www.w3.org/TR/wsdl>, March 15, 2001.
- [5] R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer, S. Weerawarana (Eds.), Web services description language (WSDL), version 2.0, Part 1: core language, World Wide Web Consortium (W3C) working draft, on-line at: <http://www.w3.org/TR/2003/WD-wsdl20-20031110>, November 10, 2003.
- [6] A. Sahai, A. Durante, V. Machiraju, Towards automated SLA management for web services, Research Report HPL-2001-310 (R.1), Hewlett-Packard (HP) Laboratories, Palo Alto, on-line at: <http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf>, July 26, 2002.
- [7] S. Frolund, J. Koistinen, Quality of service specification in distributed object systems design, in: Proceedings of the Fourth USENIX Conference on Object-Oriented Technologies and Systems—COOTS '98, Santa Fe, USA, April 1998, USENIX.
- [8] A. Beugnard, J.-M. Jezequel, N. Plouzeau, D. Watkins, Making components contract aware, *Computer*, IEEE-CS 32 (7) (1999) 38–45.
- [9] V. Tosic, K. Patel, B. Pagurek, WSOL—a language for the formal specification of classes of service for web services, in: Proceedings of the 2003 International Conference on Web Services—ICWS'03, Las Vegas, USA, June 2003, CSREA Press, pp. 375–381.
- [10] K. Patel, XML Grammar and parser for the web service offerings language, M.A.Sc. Thesis, Carleton University, Ottawa, Canada, on-line at: <http://www.sce.carleton.ca/netmanage/papers/KrutiPatelThesisFinal.pdf>, January 30, 2003.
- [11] V. Tosic, K. Patel, B. Pagurek, Reusability constructs in the web service offerings language (WSOL), in: Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web (WES) at CAiSE'03, Velden, Austria, June 2003, LNCS, No. 2681, Springer, Berlin, pp. 468–484.
- [12] K. Patel, B. Pagurek, V. Tosic, Improvements in WSOL grammar and “premier” WSOL Parser, Res. Rep. SCE-03-25, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, on-line at: <http://www.sce.carleton.ca/netmanage/papers/PatelEtAlResRepOct2003.pdf>, October 2003.
- [13] H. Ludwig, A. Keller, A. Dan, R.P. King, R. Franck, 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.
- [14] A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, F. Casati, Automated SLA monitoring for web services, in: Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management—DSOM 2002, Montreal, Canada, October, Lecture Notes in Computer Science (LNCS), No. 2506, Springer, Berlin, 2002, pp. 28–41.
- [15] D.D. Lamanna, J. Skene, W. Emmerich, SLAng: a language for defining service level agreements, in: Proceedings of the Ninth IEEE Workshop on Future Trends in Distributed Computing Systems—FTDCS 2003, Puerto Rico, May 2003, IEEE-CS Press, pp. 100–106.
- [16] M. Hondo, C. Kaler (Eds.), Web services policy framework (WS-Policy), version 1.0, BEA/IBM/Microsoft/SAP, on-line at: <ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf>, December 18, 2002.
- [17] Z. Chen, C. Lianf-Tien, B. Silverajan, L. Bu-Sung, UX—an architecture providing QoS-aware and federated support for UDDI, in: Proceedings of the 2003 International Conference on Web Services—ICWS'03, Las Vegas, USA, June 2003, CSREA Press, pp. 171–176.

- [18] The DAML Services Coalition, DAML-S: semantic markup for web services, WWW page for DAML-S version 0.9, on-line at: <http://www.daml.org/services/daml-s/0.9/daml-s.html>, May 5, 2003.
- [19] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, M. Xu, Agreement-based grid service management (OGSI-Agreement), Version 0, Global Grid Forum, June 2003.
- [20] V. Tasic, W. Ma, B. Pagurek, B. Esfandiari, On the dynamic manipulation of classes of service for XML web services, in: Proceedings of the 10th Hewlett-Packard Open View University Association (HP-OVUA) Workshop, Geneva, Switzerland, July 2003, Hewlett-Packard.
- [21] V. Tasic, W. Ma, B. Pagurek, B. Esfandiari, Web service offerings infrastructure (WSOI)—a management infrastructure for XML web services, in: Proceedings of the NOMS—The IEEE/IFIP Network Operations and Management Symposium 2004, Seoul, South Korea, April 2004, IEEE, pp. 817–830.
- [22] V. Tasic, B. Esfandiari, B. Pagurek, K. Patel, On requirements for ontologies in management of web services, in: Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web at CAiSE'02, Toronto, Canada, May 2002, Lecture Notes in Computer Science (LNCS), No. 2512, Springer, Berlin, pp. 237–247.
- [23] The Axis Development Team, Axis architecture guide, 1.0 Version, The Apache Software Foundation, on-line at: http://archive.apache.org/dist/ws/axis/1_0/xml-axis-10.zip, October 7, 2002.
- [24] WebServices.Org, Introduction to axis, WWW resource, WebServices.Org, on-line at: <http://www.webservices.org/index.php/article/articleview/415/1/24/>, May 28, 2002.
- [25] I. Foster, C. Keselman, J.M. Nick, S. Tuecke, Grid services for distributed systems integration, *Computer, IEEE-CS 35 (6) (2002) 37–46*.
- [26] M. Potts, I. Sedukhin, H. Kreger, E. Stokes, Web service manageability—specification (WS-Manageability), version 1.0, IBM/CA/Talking Blocks submission to the OASIS Web Services Distributed Management TC, September 2003.
- [27] M. Sloman, Management issues for distributed services, in: Proceedings of the IEEE Second International Workshop on Services in Distributed and Networked Environments—SDNE '95 Whistler, Canada, June 1995, IEEE-CS Press, New York, pp. 52–59.