WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services

Vladimir Tosic, Bernard Pagurek, Kruti Patel

Research Report OCIECE-02-06
November 2002

WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services

Vladimir Tosic, Bernard Pagurek, Kruti Patel
Department of Systems and Computer Engineering, Carleton University
1125 Colonel By Drive
Ottawa, ON, K1V 6L8, CANADA
+1 (613) 520-2600, x3548
{vladimir, bernie, kpatel}@sce.carleton.ca

ABSTRACT

We are developing a language, Web Service Offerings Language (WSOL), for the formal specification of various constraints, management statements, and classes of service for Web Services. WSOL is an XML (Extensible Markup Language) notation compatible with WSDL (Web Services Description Language).

A service offering in WSOL is a formal description of one class of service of a Web Service. It contains formal representation of various constraints: functional (pre-, post-, and future-conditions), Quality of Service (QoS, a.k.a. non-functional, extra-functional), and access rights. It also contains management statements, such as statements about prices, monetary penalties, and management responsibilities. One Web Service can be associated with multiple service offerings. For easier specification of similar service offerings, WSOL enables specification of constraint groups (CGs) and constraint group templates (CGTs). We have also developed a format for representation of dynamic relationships between service offerings.

WSOL service offerings are simple contracts and SLAs (Service Level Agreements) between Web Services. Describing a Web Service in WSOL, in addition to WSDL, enables monitoring, metering, and management of Web Services. The Web Service, its consumer, or one or more designated third parties (usually SOAP message intermediaries) can meter QoS metrics and evaluate constraints in WSOL service offerings. Further, manipulation of service offerings can be used for dynamic adaptation and management of Web Service compositions. In addition, WSOL supports selection of a more appropriate Web Service and service offering for particular circumstances.

The main distinctive characteristics of WSOL, compared to recent related works, are its expressive power, features that reduce run-time overhead, and orientation towards management applications.

Categories and Subject Descriptors

K.6 [**Management of Computer and Information Systems**], D.m [**Miscellaneous**], H.m [**Miscellaneous**].

General Terms

Management, Measurement, Languages.

Keywords

Web Service, constraint, class of service, service offering, SLA, WSDL, WSOL, management of Web Services.

1. INTRODUCTION

In the last couple of years, there has been a lot of work on Web Service technologies. The W3C (World Wide Web Consortium) defines a **Web Service** as "a software application identified by a URI, 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" [12]. URI means 'Uniform Resource Identifier' and XML means 'Extensible Markup Language'. The three main Web Service technologies are the SOAP protocol for XML messaging, the WSDL (Web Service Description Language) description language, and the UDDI (Universal Description, Discovery, and Integration) directory. The ultimate goal of the standardization efforts related to Web Services is a standard platform, based on already widely used technologies like XML, for distributed application-to-application (A2A) and business-to-business (B2B) integration [3].

While Web Services can be used for providing services to human end users, their true power is leveraged through **compositions** (a.k.a. orchestrations, choreographies, flows, networks) of Web Services. Hereafter, by a **consumer** (a.k.a. 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 **supplier** (a.k.a. provider) 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.

Since Web Service technologies are relatively young, standardization bodies do not yet address a number of important issues. Our research group is researching issues related to management of Web Services and Web Service compositions. One part of this research is our work on **Web Service Offerings**

Language (WSOL) – an XML language for formal specification of various constraints, management statements, and classes of service for Web Services.

In this section, we have defined the general area of our work and terminology used. In Section 2, we describe the motivation and goals for the development of WSOL. Then, in Section 3, we discuss why we have made classes of service the central concept in WSOL and define the term 'service offering'. In Section 4, we discuss in detail WSOL language features and constructs. After that, we discuss possible applications of WSOL, primarily in the management area, in Section 5. We briefly review some recent related works in Section 6 and then summarize the distinctive benefits of WSOL in Section 7. In Section 8 we make general conclusions and summarize future work.

2. MOTIVATION AND GOALS

When SOAP, WSDL, and UDDI were first published, we examined them to see how they support management activities. It was easy to conclude that these technologies needed significant additions to better support management. We were particularly intrigued by the fact that WSDL does not support specification of various constraints, management statements, classes of service, SLAs (Service Level Agreements) and other contracts between Web Services. Explicit, precise, and unambiguous specification of such information is crucial for management activities.

Functional constraints, such as pre-conditions and post-conditions, are invaluable in determining whether a Web Service behaves correctly. Consequently, they are useful in fault management and, to some extent, configuration management. Formal and precise specification of Quality of Service (QoS, a.k.a. non-functional, extra-functional) constraints is the basis for monitoring and metering QoS metrics. It prescribes which QoS metrics to monitor, where and how to do this monitoring, how to eventually calculate aggregate QoS metrics, what the expected values of QoS metrics are, and eventually what to do if they are not met. Consequently, formal and precise specification of QoS metrics is particularly useful in performance management. Access rights, another category of constraints, limit access to operations and ports of a Web Service and can be one part of a comprehensive security management solution for Web Services. Statements about prices and monetary penalties that have to be paid are invaluable in accounting management. Grouping of various constraints and management statements into classes of service, SLAs, or other contracts between Web Services helps in better handling of the complexity of management information. Therefore, it is very useful for all management activities.

Since WSDL does not specify the information that is crucial for our research of Web Services and Web Service compositions, we have decided to develop our own XML language for this purpose. We have named this language Web Service Offerings Language (WSOL). As will be explained in the next section, we have chosen the concept of a class of service to model SLAs and contracts between Web Services in WSOL. We had several goals for WSOL:

1. Usability for monitoring, metering, and management of Web Services and Web Service compositions. Since our research group is interested in management of Web Services and Web Service compositions, we have envisioned WSOL as the basis for our research on Web Services.
2. Expressive power to enable reusable specifications. When classes of service, SLAs, or other comprehensive contracts between two parties are specified, there is often a lot of similar information that differs in some details. For example, two classes of service can be the same in many elements, but differ only in response time and price. Expressive mechanisms for reuse of specifications enable easier specification of new classes of service, SLAs, or contracts from existing ones. In addition, they can be very useful in determining similarities and differences between two classes of service, SLAs, or contracts.
3. Reduction of run-time overhead. While monitoring and management is a critical business activity, it can incur significant overhead. One of the goals of our work on WSOL has been study of mechanisms to reduce this overhead without disabling management activities. We did not assume that Web Services are provided by enterprises that already have complex management frameworks and/or application servers supporting management. While supplier Web Services must have some infrastructure behind them to support management activities, this need not be the case with their consumers. Consequently, we wanted WSOL to accommodate relatively simple consumer Web Services and to support reduction of management overhead for supplier Web Services.
4. Full compatibility with WSDL. WSDL is the standard for the specification of functionality, access methods, and location of Web Services. Our goal with WSOL was reuse of this information and additional specification of various constraints, management statements, and classes of service for Web Services. Further, we have envisioned WSOL as an optional language, separate from WSDL. 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. It is beneficial that WSOL specifications can be created, deactivated, or reactivated dynamically (i.e., during run-time) without modifying the referenced WSDL files. To achieve this, a WSOL specification has to reference the underlying WSDL specification without modifying it.

The desired relationship between WSDL and WSOL is shown in

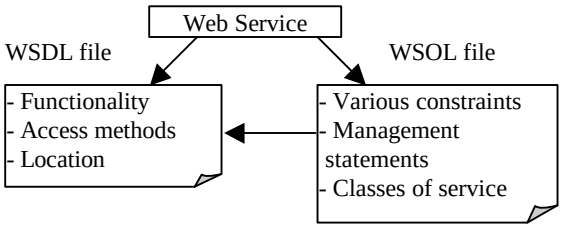

Figure 1. WSDL vs. WSOL.

Figure 1. WSOL files reference WSDL files and contain information that is not present in WSDL files.

3. SERVICE OFFERINGS

In certain circumstances, it can be useful for a Web Service to offer several different classes of service to consumers. By a '**class of service**' we mean a discrete variation of the complete

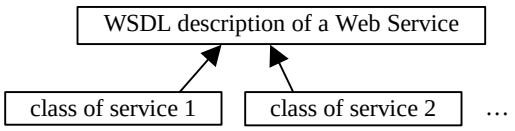

Figure 2. Multiple classes of service for one Web Service.

service and quality of service (QoS) provided by one Web Service. In other words, we discuss classes of service at the level of Web Services, not at the level of particular constraints (e.g., response time) that are part of the overall service and QoS of the Web Service.

Classes of service of one Web Service refer to the same functionality (i.e., WSDL description), but differ in constraints and management statements. For example, they can differ in usage privileges, service priorities, response times guaranteed to consumers, verbosity of response information, etc. The concept of classes of service also supports different capabilities, rights, and needs of potential consumers of the Web Service, including 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, like pay-per-use or 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 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 SLAs, profiles, parameterization, an separate ports. However, classes of service require relatively low overhead and complexity of management. Since one of our goals with WSOL was the study of mechanisms for the reduction of run-time overhead, we have decided to make classes of service the central concept in WSOL. 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.

The concept that one Web Service can have multiple classes of service is depicted in Figure 2.

We define a **service offering** as a 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 contract or one SLA between the supplier Web Service, the consumer, and eventual management third parties. A Web Service can offer multiple service offerings to a consumer, but a consumer can use only one of them at a time. WSOL service offerings are specified separately from the WSDL description of the Web Service. This enables dynamic creation, deactivation, and/or reactivation of service offerings without any modification of the underlying WSDL file.

Note that in WSOL service offerings are specified for the complete Web Service, not for separate ports. In other words, we discuss 'component-level service offerings (CLSOs)'. In WSOL, there is no need for a separate concept of a 'port-level service offering (PLSO)' that would contain constraints and management statements for a particular port. If needed, WSOL constraints and management statements for one port can be grouped into a constraint group (CG), as will be discussed later in this paper.

4. LANGUAGE FEATURES

The syntax of WSOL is defined using XML Schema. WSOL is currently compatible with WSDL 1.1, but we will make it compatible with WSDL 1.2 when the work on the latter is finalized.

WSOL is based on the following specification **constructs**:

- constraint,
- statement,
- constraint group (CG),
- constraint group template (CGT), and
- service offering.

4.1 Constraints and Expressions

In WSOL, every **constraint** is a Boolean expression that states some condition to be evaluated. The constraints can be evaluated before and/or after invocation of operations or at particular date/time instances. WSOL enables formal specification of:

1. **Functional constraints** (pre-, post-, and future-conditions). These constraints define conditions that a functionally correct operation invocation must satisfy. They usually check some characteristics of message parts of the invoked operation.
2. **QoS (a.k.a. non-functional, extra-functional) constraints**. These constraints describe properties such as performance, reliability, and availability. They check whether the monitored QoS metrics are within specified limits.
3. **Access rights**. An access right specifies conditions under which any consumer using the current service offering has the right to invoke a particular operation. If access is not explicitly allowed, it is forbidden. 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 consumer) may use a service offering and other security issues are outside the scope of WSOL.

WSOL constraints are defined using the <*constraint*> element, which is independent of particular types of constraints. The *type* attribute of the <*constraint*> element refers to the XML schema defining a particular type of constraints. We have defined XML schemas for the above mentioned types of constraints. Using the XML Schema mechanisms, additional types of constraints can be defined.

Figure 3 shows an example WSOL constraint, in this case a precondition. The <*constraint*> element contains attributes determining the type of constraint, name, and scope to which it applies. In this case, the constraint *C3* applies to a particular service, port, and operation. However, constraints and other

WSOL constructs, other than service offerings, can be defined for scopes that are more abstract. Some examples of such 'more abstract' scopes are 'a particular operation of a particular port type (but not a particular port)' and 'any operation of any port of any service'. The shown constraint contains a simple Boolean expression that compares an operation input parameter and a constant.

```
<wsol:constraint name="C3" xsi:type= "preConditionSchema:
preCondition" service="buyStock:buyStockService"
portOrPortType="buyStock:buyStockServicePort"
operation="buyStock:buySingleStockOperation">
  <expressionSchema:booleanExpression>
    <expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticVariable
avName="buyStock:buySingleStockRequest.quantity"/>
    </expressionSchema:arithmeticExpression>
    <expressionSchema:arithmeticComparator type=">"/>
    <expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticConstant>
        <expressionSchema:integerConstant value="0"/>
      </expressionSchema:arithmeticConstant>
  </expressionSchema:arithmeticExpression>
  </expressionSchema:booleanExpression>
</wsol:constraint>
```

Figure 3. An example WSOL pre-condition.

Boolean expressions in constraints can contain standard Boolean operators (AND, OR, NOT, IMPLIES, EQUIVALENT), references to operation message parts of type Boolean, and comparisons of arithmetic, string, date/time, or duration expressions. Arithmetic expressions can contain standard arithmetic operators (+, -, unary -, *, /, **), arithmetic constants, and references to operation message parts of numeric data types. WSOL provides only basic built-in support for string and date/time/duration expressions. However, it is possible to perform external operation calls in any expression. Here, 'external' means 'outside the Web Service for which the constraint is specified'. These external operations can be implemented by other Web Services or they can be implemented by the management entities 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. Note that WSOL does not support operation calls upon the same Web Service because there is no way to guarantee that they are side-effect free (i.e., that they do not change the state of the Web Service). Evaluation of constraints must be side effect free. WSOL also supports checking operation message parts that are arrays (of any data type) using quantifiers ForAll and Exists.

The concept of a future-condition is a novel concept, first introduced in WSOL. A future-condition is a Boolean expression evaluated some time after the supplier finishes execution of the requested operation and sends results to the consumer. This is different from a post-condition, which is evaluated when the supplier sends results to the consumer. In WSOL, one can specify that a future-condition should be evaluated: a) on a particular date/time; b) after a specified duration elapses from the completion of the invoked operation; c) periodically from one date/time to another date/time with specified interval duration and number of repeats. If a future-condition is not satisfied, operation invocation is considered invalid and the supplier has to pay some penalty. The concept of a future-condition enables specification of operation effects that cannot be easily expressed with post-conditions. This includes some effects that a Web Service operation can have in the physical world. An example is delivery confirmation for goods bought using Web Services.

For specification of QoS constraints, WSOL needs external ontologies of QoS metrics and measurement units. We have summarized requirements for such ontologies in [11]. In our current implementation of WSOL, we have simply assumed that ontologies of QoS metrics are collections of names with information about appropriate data types and measurement units. Similarly, ontologies of measurement units are simple collections of names without any additional information. A more appropriate definition of ontologies of QoS metrics, measurement units, as well as monetary units for price/penalty statements is planned for a future version of WSOL.

Note that one could argue for separate languages for different categories of constraints. For example, one could suggest one language for functional constraints, another language for QoS constraints and SLAs, and maybe a third language for access rights. However, there are benefits of describing various constraints and management statements in one language. These benefits are related to our goals of WSOL development, particularly reduction of incurred overhead. There is less overhead in supporting one language for various constraints than several separate languages. This is because syntax of different constraints is similar (Boolean expressions containing arithmetic and other expressions), while management statements (such as prices and penalties) relate to all constraints and not a particular category of constraints. Further, a unified language for various categories of constraints can reduce redundancies and potential incompatibilities that can occur when similar information is described in different ways. In addition, dependencies between different categories of constraints can occur (although such cases are probably rare). For example, one can state that the lowest response time for all operations is reserved for those consumers that have access to management operations. When all constraints and management statements are described in one language, suppliers can express such dependencies more easily. Finally, formal specification of various categories of constraints and management statements in one language enables definition of comprehensive classes of service, the benefits of which will be discussed in the next section.

4.2 Statements

A WSOL **statement** is any construct, other than a constraint, that states some important information about the represented class of service. WSOL enables formal specification of various statements: price/penalty statements, management responsibility statements, include statements, and declarations of external operation calls.

Price statements specify the price that a consumer using the particular service offering has to pay for successful use of the Web Service. Penalty statements specify the monetary amount that the 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 service offering. WSOL price/penalty statements support the subscription and the pay-per-use payment models, as well as their combinations.

Figure 4 shows an example price statement in WSOL. This price statement specifies pay-per-use price for a particular operation. The <*price*> element has attributes for name of the statement and for the scope to which it applies. The currency *CanadianDollar* used for this price is defined in an external ontology of monetary units.

```
<wsol:price name="Price1" service="buyStock:buyStockService"
portOrPortType="buyStock:buyStockServicePort"
operation="buyStock:buySingleStockOperation">
  <wsol:numberWithUnitConstant>
    <wsol:value>0.003</wsol:value>
    <wsol:unit type="currencyOntology:CanadianDollar"/>
  </wsol:numberWithUnitConstant>
</wsol:price>
```

Figure 4. An example WSOL price statement.

A management responsibility statement specifies what entity has management responsibility for checking a particular constraint, a constraint group, or the complete service offering. A management entity can be the supplier Web Service, the consumer, or an independent third party trusted by both the supplier and the consumer.

The <*include*> statement enables constraints, statements, or constraint groups to be reused across different service offerings, constraint groups, and/or constraint group templates. This powerful reuse mechanism will be discussed in more detail and illustrated with an example later in this paper.

Declaration of external operation calls enables results of the same external operation call to be used in several related constraints.

4.3 Constraint Groups

A **constraint group (CG)** is a named set of constraints and/or statements. A CG can also contain other CGs (including instantiations of CGTs, which will be discussed later). Arbitrary levels of nesting of CGs are allowed.

```
<wsol:CG name="CG7" service="buyStock:buyStockService"
portOrPortType= “WSOL-ANY” operation="WSOL-ANY">
  <wsol:CG name="CG8" service="buyStock:buyStock-Service"
portOrPortType="buyStock:buyStockServicePort"
operation="buyStock:buySingleStockOperation">
  …
  </wsol:CG>
  <wsol:constraint name="C7" xsi:type="preConditionSchema:
preCondition" service="buyStock:buyStockService"
portOrPortType="buyStock:buyStockServicePort"
operation="buyStock:buyMultipleStocksOperation">
  </wsol:constraint>
    …
  <wsol:include constructName="C3" resService="buyStock:
buyStockService"
resPortOrPortType="buyStock:buyStockServicePort"
resOperation="buyStock:buySingleStockOperation"
resName="C3inCG7"/>
</wsol:CG>
```

Figure 5. An example WSOL constraint group.

The WSOL concept of a CG has several benefits. First, a CG can be reused across service offerings as a unit. Second, it is possible to specify that all constraints from a CG are evaluated by the same management entity. Third, constraints in different CGs can have the same constraint name, so using CGs enables name reuse. Fourth, one can use CGs to define aspects of service offerings. For example, one can group all functional constraints for one port type into one CG, QoS constraints for the same port type into another CG, and access rights for this port type into a third CG. Fifth, as already mentioned, CGs can be used as port-level service offerings.

When a new CG is defined and some of the contained constraints and CGs have been already defined elsewhere, there is no need to define them again. They can simply be included into the new containing CG using the WSOL <include> statement. On the other hand, new constraints and CGs can also be defined inside a containing CG. A new CG can be defined as an extension of an existing CG, inheriting all constraints, statements, and nested CGs and defining some additional ones. Extension is, in fact, single inheritance of CGs. We have also studied multiple inheritance, but it is not part or the current version of WSOL. Benefits similar to multiple inheritance can be achieved in WSOL by including several existing CGs inside the new CG. If inside one CG two or more constraints of the same type (e.g., two pre-conditions) are defined for the same operation, they all have to be satisfied. This means that the Boolean AND operation is performed between such constraints.

Figure 5 illustrates the WSOL concept of a CG. The CG named *CG7* contains the nested CG named *CG8*, the constraint *C7*, and the <*include*> statement for inclusion of the constraint *C3* (defined in Figure 3). The details of *CG8* and *C7* are left out for brevity. Note that CG7 contains information that relates to different operations and ports of the same service. This is specified with the “WSOL-ANY” constant as value of attributes of the <*CG*> element. During inclusion, the scope of included constraints, statements, and CGs can be specialized (according to some rules that are out of scope of this paper). However, in the shown example there is no specialization of scope. Analogous specialization of scope during instantiation of CGTs will be shown in Figure 7.

4.4 Constraint Group Templates

A **constraint group template (CGT)** is a parameterized CG. At the beginning of a CGT, one defines one or more abstract CGT parameters, each of which has a name and a type. CGT parameters often have the type ‘numberWithUnit’, which requires additional information about the used measurement unit. Definition of parameters is followed by definition of constraints and nested CGs, in the same way as for CGs. Constraints inside a CGT can contain expressions with CGT parameters.

```
<wsol:CGT name="CGT2" service="WSOL-ANY"
portOrPortType="WSOL-ANY" operation="WSOL-ANY">
  <wsol:parameter name="maxResTime" dataType="wsol:
numberWithUnit" unit="QoSMeasOntology:millisecond"/>
  <wsol:constraint name="QoScons2" service="WSOL-ANY"
portOrPortType="WSOL-ANY" operation="WSOL-ANY">
    <expressionSchema:booleanExpression>
      <expressionSchema:arithmeticExpression>
        <expressionSchema:QoSmetric metricType="QoS-
MetricOntology:ResponseTime“ service="WSOL-ANY"
```

```
portOrPortType="WSOL-ANY" operation="WSOL-ANY“
measuredBy="WSOL_INTERNAL"/>
    </expressionSchema:arithmeticExpression>
    <expressionSchema:arithmeticComparator type="&lt;"/>
    <expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticVariable avName=
"tns:CGT2.maxResTime"/>
    </expressionSchema:arithmeticExpression>
  </expressionSchema:booleanExpression>
 </wsol:constraint>
</wsol:CGT>
```

Figure 6. An example constraint group template.

Figure 6 illustrates the WSOL concept of a CGT. The CGT *CGT2* and the QoS constraint *QoScons2* within it are defined for any operation of any port of any service. This CGT has one parameter *maxResTime*, of the data type *numberWithUnit* and measurement unit millisecond (defined in an external ontology of measurement units). The QoS constraint *QoScons2* states that the measured value of the QoS metric *ResponseTime* must be less than the value of the parameter *maxResTime.*

A CGT is instantiated when concrete values are supplied for all CGT parameters. The result of such instantiation is a new CG. A CGT can be instantiated inside a CG, a definition of another CGT, or a service offering. One CGT can be instantiated many times with different parameter values. For example, one can define a CGT with one parameter 'maxRT' and one constraint that the measured response time must be less than 'maxRT'. Then, this CGT can be instantiated with 'maxRT' parameter values 20 milliseconds, 50 milliseconds, 2 seconds, etc.

The concept of a CGT in WSOL is a very powerful specification mechanism. Many classes of service (and SLAs) contain constraints with the same structure, but with different constant values. In our opinion, it is an even more important specification concept than the single inheritance (i.e., extension) of CGs, CGTs, and service offerings. However, the WSOL concept of a CTG also has some limitations. First, CGT definitions must not be nested. In other words, one must not define a CGT inside another CGT. Next, since constraints inside a CGT may contain expressions with CGT parameters, these constraints must not be included inside other CGTs, CGs, or service offerings. Further, WSOL supports single inheritance (i.e., extension) of CGTs, similarly to extension of CGs. However, a CGT extending some other CGT must not define additional CGT parameters. Only addition of new contained constraints, statements, and CGs is allowed.

4.5 Syntax of Service Offerings

Syntactically, a WSOL **service offering** is similar to a CG. It is a set of constraints, statements, and CGs (including instantiations of CGTs) that all refer to the same Web Service. Further, WSOL supports single inheritance (extension) of service offerings, similarly to single inheritance of CGs and CGTs. The rules discussed for CGs also apply for service offerings. An important exception is that service offerings must not be nested. We make a service offering a separate concept in WSOL to emphasize its special run-time characteristics. Most importantly, consumers can choose and use service offerings, not CGs. This is because CGs need not be complete and consistent from the usability view. For the same reason, dynamic relationships can be specified only for service offerings, not for CGs.

The accounting party is a special management party responsible for keeping track of the use of the supplier Web Service and management third parties, as well as what constraints were kept and what were not. In addition, the accounting party is the default management entity. In other words, it is responsible for evaluation of all constraints for which management responsibility is not specified explicitly through management responsibility statements. Due to its special purpose, the accounting party is specified through an attribute of the <*serviceOffering*> element and not through a <*managementResponsibility*> statement.

```
<wsol:serviceOffering name="SO1" service="buyStock:
buyStockService" accountingParty="WSOL-SUPPLIERWS">
 <wsol:instantiate CGTName="CGT2" resService=
"buyStock:buyStockService" resPortOrPortType="WSOL-EVERY"
resOperation="WSOL-EVERY" resCGName="CG5">
   <wsol:parmValue name="maxResTime">
    <wsol:numberWithUnitConstant>
     <wsol:value>300</wsol:value>
     <wsol:unit type="QoSMeasOntology:millisecond"/>
    </wsol:numberWithUnitConstant>
   </wsol:parmValue>
 </wsol:instantiate>
 …
</wsol:serviceOffering>
```

Figure 7. An example WSOL service offering.

Figure 7 shows an example WSOL service offering. The attributes of the <*serviceOffering*> element are *name*, *service*, and *accountingParty*. Since a service offering is always defined for a particular service, there is no need for the *portOrPortType* and *operation* attributes present in other WSOL constructs. In the given example, the service offering *SO1* contains an instantiation of the CGT *CGT2* shown in Figure 6. Since this CGT was defined for any operation of any port of any service, during instantiation a more specific scope is specified, along with the name of the result CG (*CG5*). The scope of the new constraint group is every operation of every port of the Web Service *buyStockService*. During instantiation, the abstract parameter *maxResTime* from *CGT2* is replaced with the constant value of 300 milliseconds. Therefore, every operation of every port of the Web Service *buyStockService* must have response time less then 300 milliseconds. Other parts of the service offering *SO1* are omitted for brevity.

4.6 Relationships between Service Offerings

One important issue in WSOL is how to represent relationships between service offerings. These relationships have to be specified for at least three purposes. The first one is to provide a more straightforward and more flexible specification of new service offerings. This is needed to specify relatively similar service offerings of one Web Service, as well as relatively similar service offerings of similar Web Services. The second purpose is to enable easier selection and negotiation of service offerings. The third purpose is to support dynamic adaptation of Web Service compositions based on the manipulation of service offerings, which we will briefly discuss in the next section.

Our study of this issue showed that a difference should be made between static and dynamic relationships between service offerings. Static relationships between service offerings are those that do not change during run-time. For example, two classes of service can instantiate the same template, or one class of service can be an extension of another class of service. Such relationships can be built into definitions of service offerings. These relationships are crucial for easier and more flexible specification of new service offerings from existing ones. In WSOL, static relationships between service offerings are modeled with the mechanism for reuse of specifications. In particular, single inheritance (extension) of service offerings is explicitly specified in WSOL. In addition, service offerings that instantiate the same CGTs and/or include the same constraints, statements, and CGs can be related to each other.

On the other hand, dynamic relationships between service offerings are those that can change during run-time, e.g., after dynamic creation of a new class of service. For example, one 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. Such relationships should not be built into definitions of service offerings, to avoid frequent modification of these definitions. Dynamic relationships between service offerings are useful for easier selection and negotiation of service offerings and for dynamic adaptation of Web Service compositions.

After research of several alternatives, we have decided to represent dynamic relationships between service offerings as triples <SO1, S, SO2> where:

- SO1 is a service offering;
- S is a set of constraints, statements, and CGs from SO1 that are not satisfied; and
- SO2 is the appropriate replacement service offering.

These triples are specified in a special XML format outside WSOL files to make their evolution independent from the evolution of other characteristics of a service offering.

5. APPLICATIONS

As argued in more detail in [9] and [10], WSOL can be used in several ways. We are particularly interested in management applications of WSOL. We strongly believe that appropriate specification of management information is the key for successful management activities. WSOL describes for Web Services the QoS metrics to monitor, the constraints to evaluate, as well as when (and to some extent: how) to perform particular management activities. Consequently, WSOL service offerings can be used for Web Service monitoring, metering, control, accounting, and billing. They are precise and complete enough to serve as simple contracts or SLAs between Web Services. Further, dynamic (i.e., run-time) manipulation of service offerings is a useful tool for management of Web Services and Web Service compositions.

When a consumer submits a request for executing a supplier's operation, the management third parties are organized as SOAP intermediaries for the request, as well as the eventual response message. An example configuration of management third parties as SOAP intermediaries is shown in Figure 8. Some QoS metrics, such as availability, can be measured using probing instead of message interception. WSOL supports this by

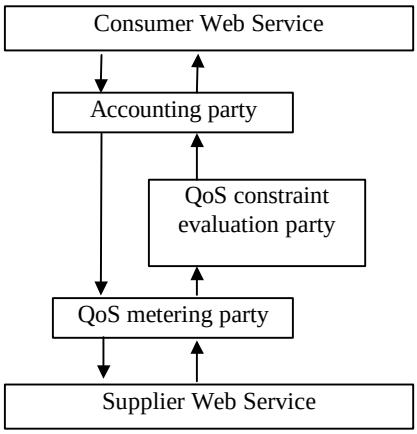

Figure 8. An example configuration of management third parties as SOAP intermediaries.

modeling probing entities as separate Web Services that provide results of their measurements through operations of some agreed-upon interfaces. These operations can be invoked in appropriate QoS constraints in WSOL service offerings, using the WSOL external operation call mechanism.

We are also researching management and dynamic adaptation of Web Service compositions without breaking an existing relationship between a Web service and its consumer. To achieve this goal we are exploring management and dynamic adaptation mechanisms that are based on the manipulation of service offerings in WSOL. Our dynamic adaptation mechanisms include switching between service offerings, deactivation/reactivation of existing service offerings, and creation of new appropriate service offerings. These mechanisms can be used between operation invocations that are part of the same transactions or session. We summarize them here, while more detail is given [9] and in a forthcoming publication. The crucial support for these mechanisms is specification of dynamic relationships between service offerings discussed in the previous section.

Dynamic switching between service offerings enables consumers to dynamically adapt the service they receive without the need to find another Web service. It also enables Web services to gracefully degrade or upgrade their service and QoS in case of changes.

Deactivation and reactivation of service offerings is used by a Web service in cases when changes in operational circumstances affect what service offerings it can provide to consumers. When a change of circumstances occurs, a Web service can dynamically and automatically deactivate service offerings that cannot be supported in the new circumstances. The affected consumers are switched to an appropriate replacement service offering and notified about the change. If there is no appropriate replacement service offering, an alternative supplier Web Service has to be sought. The deactivated service offering might be reactivated automatically at a later time after another change of circumstances and, eventually, the consumers can be automatically switched back to their original service offering and notified about the change.

Dynamic creation of new service offerings can be used when there has been a change in the Web Services implementation

(e.g., in case of dynamic versioning/evolution) or the execution environment. To some limited extent, it can also be performed after a demand of important consumers. It then becomes a substitute for negotiation of a custom-made contract or SLA between Web Services. Dynamic creation of new service offerings is often non-trivial and can incur significant overhead. Therefore, we use it only in exceptional circumstances.

Compared to finding alternative Web Services (i.e., re-composition of Web Service compositions), these three dynamic adaptation mechanisms enable faster and simpler adaptation and enhance robustness of the relationship between a Web Service and its consumer. Further, these capabilities are simple and incur relatively low overhead, while providing additional flexibility. However, compared to finding alternative Web Services, these dynamic adaptation mechanisms have 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 a replacement for, finding alternative Web Services. The first step in dynamic adaptation of a Web Service composition is to try to find a replacement service offering 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 fact, a Web Service can provide a temporary replacement service offering while the consumer searches for another, more appropriate, Web Service.

In addition to management applications, WSOL can be used in the process of selection of supplier Web Services (and their service offerings) that are best for particular circumstances. As the number of Web Services on the market that offer similar functionality increases, the offered QoS and price/performance ratio become important competitive advantages. Comprehensive WSOL descriptions of Web Services, help consumers to better choose service and QoS that they will receive and pay for.

6. 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 (e.g.,[2]). At the beginning of our research, there was no relevant work of this kind in the area of Web Services. In parallel with our research, several related works emerged.

The most important related works to WSOL are two recent languages for formal XML-based specification of custom-made SLAs for Web Service: WSLA (Web Service Level Agreements) [4] from IBM and the HP work on the formal specification of Web Service SLAs [8]. The latter work seems to be part of WSML (Web Service Management Language). SLAs in these two languages contain QoS constraints and some management information.

WSLA enables formal specification of contract parties, service definitions, and obligations (service level objectives and action guarantees) of the contract parties. 'Service definitions' in WSLA contain information about schedules, triggers, SLA parameters, metrics, operations, and operation groups. All these elements of SLAs can be described in detail, achieving precise description of what QoS metrics are measured, where and how they are measured, as well as how to compute aggregate (composite) metrics from raw measured metrics. In this aspect, WSLA captures more detail than WSOL, which leaves definition of QoS metrics to external ontologies. Further, QoS constraints in WSOL relate to service level objectives in WSLA, but WSOL has only implicit notion of action guarantees - payment of monetary penalties.

WSML enables formal and unambiguous specification of information about when SLAs should be evaluated, which inputs should be considered for evaluation, where are the measurements should occur, as well as what and how to evaluate. In addition, it is a flexible SLA formalization, fully compatible with WSDL and WSFL (Web Services Flow Language). However, WSML does not enable specification of management third parties. Further, WSML does not define the language for expressions to be evaluated. It is assumed that expressions will be written in some other mathematical languages, such as MathML. This means that the infrastructure for the evaluation of WSML constraints should also support these mathematical languages.

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. They specify more detail for QoS constraints than WSOL and specify custom-made SLAs, not classes of service. It seems that this results in higher run-time overhead than it is needed for the simpler WSOL. Both WSLA and WSML have some support for templates, but only at the level of and SLA, not its parts. They do not have support for inheritance of specifications and other support for reuse of specifications that WSOL has. Contrary to WSOL, these languages do not address formal specification of functional constraints, access rights, and other constraints. To conclude, while both WSLA and WSML are good language for their domain and purpose, they are not addressing all issues that WSOL does.

Another related work to WSOL is DAML-S (DAML-Services) [5]. This community works on semantic description of Web Services, including specification of functional and some QoS constraints. However, properties intended for formal specification of constraints in DAML-S are currently only placeholders for constraints, because one can use any kind of DAML object for them. It is expected that the DAML rule language, when it is developed, will be used for the formal specification of functional constraints in DAML-S. Also, properties for QoS constraints in DAML-S are defined somewhat ambiguously. They are not intended for actual monitoring and metering of QoS metrics and evaluation of QoS constraints, but only for more comprehensive description of DAML-S services. This is a major difference DAML-S and WSOL. DAML-S also enables a service to provide multiple service profiles, each describing functionality and various constraints. However, contrary to WSOL, DAML-S does not explicitly define the concept of classes of service that relate to the same functionality. Further, it does not address static and dynamic relationships between classes of service. Consequently, we find that WSOL has clear advantage in management of Web Services and Web Service compositions. DAML-S was originally developed as a language incompatible with the main three Web Service technologies (SOAP, WSDL, UDDI). The works towards this compatibility have started relatively recently.

Apart from these recent works that partially address similar issues to WSOL, there are other works that recognize the importance of the formal specification of various constraints, SLAs, and contracts for Web Services. For example, the notion of WSEL (Web Services Endpoint Language) has been mentioned in the literature [6], but with no detailed publication to date. One of the goals stated for WSEL was specification of some constraints, including QoS, for Web Services. In addition, the OGSA (Open Grid Services Architecture) [7] 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.

7. SUMMARY OF WSOL BENEFITS

The main benefits of WSOL, compared with the recent related works, are its expressive power, features that reduce run-time overhead, orientation towards management applications, and full compatibility with WSDL. These benefits are direct consequences of the goals we had for WSOL, discussed in Section 2.

The major features that demonstrate unique **expressive power of WSOL** are:

1. WSOL enables formal specification of different categories of constraints and management statements, as well as multiple classes of service for one Web Service.
2. WSOL has several built-in mechanisms for reuse of specifications:
   - CGTs,
   - single inheritance (extension) of service offerings, CGs, and CGTs,
   - the <*include*> statement,
   - grouping of constraints, statements, and nested CGs into CGs and CGTs, and
   - specialization of scope during inclusion and CGT instantiation.

These mechanisms enable easier specification of new service offerings from similar existing ones.

3. WSOL supports specification of both static and dynamic relationships between service offerings (although the latter are specified outside WSOL files).
4. WSOL is extensible because new types of constraints can be specified with the generic <*constraint*> element and because QoS metrics, measurement units, and monetary units are defined in extensible external ontologies.

The major features of WSOL that are aimed at the **reduction of run-time overhead** are:

1. The central concept in WSOL is class of service (represented by a service offering), instead of more demanding alternatives such as custom-made SLAs, user profiles, and others.
2. WSOL is one language for specification of various categories of constraints (functional constraints, QoS constraints, and access rights) and management statements. The overhead of this approach is less than the overhead when different languages are used for various categories of constraints.
3. Metering of QoS metrics and evaluation of WSOL constraints can be delegated to specialized third parties (SOAP intemediaries or probes). This reduces the run-time overhead at the supplier Web Service and its consumers. It is also appropriate if consumers do not fully trust suppliers.
4. Reasoning about WSOL service offerings, for example in the process of selection and negotiation of service offerings, can be delegated to specialized third parties.

Some of the WSOL features that **support management applications** are:

1. Constraints are specified formally and unambiguously, in a format that can be used for automatic generation of constraint-checking code.
2. Management statements are important parts of service offerings:
   - management responsibility statements, and
   - statements about prices and monetary penalties.
3. WSOL has language support for management third parties:
   - management responsibility statements,
   - the *monitoredBy* attribute of QoS metrics, and
   - management entities that are not SOAP intermediaries can be modeled with external operation calls.
4. WSOL has explicit support for accounting parties, due to their special characteristics.
5. WSOL supports specification of dynamic relationships between service offering. Although this specification is done outside WSOL files, it is dependent on naming conventions and other features of WSOL.

Due to these and other features, WSOL can be actually used for monitoring, metering, management, accounting, billing, and dynamic adaptation of Web Services and Web Service compositions. In addition to the WSOL language, we are developing the corresponding management infrastructure and management algorithms, which will be presented in more detail in a forthcoming publication.

8. CONCLUSIONS AND FUTURE WORK

Numerous practical benefits can result from providing multiple classes of service and formal specification of various constraints for Web Services. WSOL service offerings are simple contracts and SLAs between Web Services. Their specification and manipulation are useful in management of Web Services and Web Service compositions. Customization of service and QoS through classes of service has limitations. Similarly, dynamic adaptation mechanisms based on manipulation of classes of service have limitation. However, they incur relatively little overhead and require limited complexity of management. Consequently, we find that our solutions are appropriate in many non-trivial situations.

Formal specification of classes of service that contain various constraints and management statements was not addressed prior our work on WSOL. Several recent works partially address similar issues, such as formal specification of custom-made SLAs containing QoS constraints. However, the distinctive characteristics of WSOL are its expressive power, features for reduction of run-time overhead, support for management applications, and full compatibility with WSDL 1.1.

We are working intensively on tools for WSOL. Most importantly, we have developed a WSOL parser performing syntax checks and some semantic checks. Its implementation is based on the Apache Xerces XML Java parser. This parser will be discussed in detail in a forthcoming publication. We are also

looking at automatic generation of some constraint-checking code from WSDL and WSOL files. This is a complex issue. In this respect, we are exploring use of composition filters [1] and similar aspect-oriented approaches. This is because a constraint-checking SOAP intermediary in our approach can be related to a composition filter. Further, we plan to develop a Java API (Application Programming Interface) for easier generation of WSOL files, but this work is still in an early stage.

Our future work on WSOL will be oriented towards further development of WSOL tools and further research of management applications of WSOL. We plan full compatibility with WSDL version 1.2 when it becomes stable. In addition, we are considering extending WSOL to achieve compatibility with BPEL4WS and some other improvements of the WSOL language syntax. The improvement of formats for external ontologies (of QoS metrics, measurement units, and monetary units), and addition of some new categories of constraints and management statements (such as possible roles in patterns and coordination protocols) are important issues for future work, but they are not our current priority.

9. REFERENCES

[1] Bergmans, L., Aksit, M. Composing Crosscutting Concerns Using Composition Filters. *Comm. of the ACM*, Vol. 44, No. 10. (Oct. 2001), pp. 51-57, ACM.

[2] Beugnard, A., Jezequel, J.-M., Plouzeau, N., Watkins, D. Making Components Contract Aware. *Computer*, Vol. 32, No. 7 (July 1999), pp. 38-45.

[3] Curbera, F., Mukhi, N., Weerawarana, S. On the Emergence of a Web Services Component Model. In *Proc. of the WCOP 2001 workshop at ECOOP 2001* (Budapest, Hungary, June 2001). On-line at: http://www.research.microsoft.com/~cszypers/events/WCOP2001/Curbera.pdf

[4] Dan, A., Franck, R., Keller, A., King, R., Ludwig, H. Web Service Level Agreement (WSLA) Language Specification. In *Documentation for Web Services Toolkit, version 3.2.1* (August 9, 2002), International Business Machines Corporation (IBM).

[5] The DAML Services Coalition. *DAML-S: Semantic Markup for Web Services*. WWW page. (December 12, 2001) On-line at: http://www.daml.org/services/daml-s/2001/10/daml-s.html

[6] 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

[7] Foster, I., Keselman, C., Nick, J. M., Tuecke, S. Grid Services for Distributed Systems Integration. *Computer*, Vol. 35, No. 6 (June 2002), pp. 37-46, IEEE –CS.

[8] Sahai, A., Durante, A., Machiraju, V. Towards Automated SLA Management for Web Services. Research Report HPL-2001-310 (R.1), Hewlett-Packard Laboratories Palo Alto. On-line at: http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf

[9] Tosic, V., Pagurek, B., Esfandiari, B., Patel, K. On the Management of Compositions of Web Services. In *Proc. of the OOWS'01 (Object-Oriented Web Services 2001) workshop at OOPSLA 2001* (Tampa, Florida, USA, Oct. 2001). On-line at: http://www.research.ibm.com/people/b/bth/OOWS2001/tosic.pdf

[10] Tosic, V., Patel, K., Pagurek, B. WSOL - Web Service Offerings Language. In *Proc. of the Workshop on Web Services, e-Business, and the Semantic Web (WES) at CaiSE'02* (Toronto, Canada, May 2002). To be publ., Springer-Verlag, Lecture Notes in Computer Science (LNCS).

[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). To be publ., Springer-Verlag, Lecture Notes in Computer Science (LNCS).

[12] 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/