Reusability Constructs in the Web Service Offerings Language (WSOL) [Second Extended Revision]

Vladimir Tosic, Kruti Patel,
Bernard Pagurek

Research Report SCE-03-21
September 2003

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

Reusability Constructs in the Web Service Offerings Language (WSOL)

Vladimir Tosic, Kruti Patel, Bernard Pagurek

Department of Systems and Computer Engineering, Carleton University,
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada
{vladimir, bernie}@sce.carleton.ca, kruts.patel@lycos.com

Abstract. The Web Service Offerings Language (WSOL) is a novel language for the formal specification of classes of service, various types of constraint, and management statements for Web Services. Compared with recent competing works, WSOL has several unique characteristics. One of them is a diverse set of reusability constructs: definition of service offerings, definition of constraint groups, definition and instantiation of constraint group templates, extension, inclusion, specification of applicability domains, and declaration of operation calls. These constructs enable sharing parts of WSOL specifications between classes of service of different Web Services and development of libraries of reusable WSOL specifications. Consequently, they can help in alleviating heterogeneity of Web Services. In addition, reusability constructs are useful for easier development of new WSOL specifications from existing ones, for easier selection of Web Services and their classes of service, and for dynamic (run-time) adaptation of relationships between provider and consumer Web Services. An integration of WSOL reusability constructs into the works competing with WSOL would be beneficial.

1 Introduction and Motivation

The Web Service Description Language (WSDL) version 1.1 is the de-facto standard for the description of Web Services. However, WSDL does not enable the specification of constraints, management statements, and classes of service for Web Service. As discussed in [1], the specification of different types of constraint and management statements is necessary for the management of Web Services and Web Service compositions. In addition, classes of service are a simple and lightweight alternative to contracts, service level agreements (SLAs), and profiles [2]. Therefore, we have decided to develop our own language for the specification of classes of service, various types of constraint, and management statements for Web Service. We have named this language the **Web Service Offerings Language (WSOL)**.

When multiple classes of service are specified, there is often a lot of similar information that differs in some details. For example, classes of service that a telecommunication service provider offers 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. Defining common or similar parts of classes of service once and using these definitions many times simplifies the specification of new classes of service. Next, when it is explicitly stated that two classes of service share common parts, it is much easier to compare them. Such comparisons are useful in the process of selection and negotiation of Web Services and their classes of service. Further, when monitored classes of service have common elements, the overhead placed on the management infrastructure for the monitoring of Web Services, metering or calculation of quality of service (QoS) metrics, and evaluation of constraints, might be reduced. In addition, manipulation of classes of service can be used for simple dynamic (run-time) adaptation of Web Service compositions [1, 2]. Explicit specification of relationships between classes of service supports their comparison, as well as their manipulation.

For these reasons, we have built into WSOL a diverse set of **reusability constructs**. These constructs enable reuse of parts of WSOL specifications and easier comparisons of WSOL specifications, even when these WSOL specifications are specified for different Web Services. In this way, WSOL reusability constructs can help in alleviating heterogeneity of Web Services. They also model static relationships between classes of service (relationships that do not change during run-time) and thus support manipulation of classes of service.

WSOL was developed independently of and in parallel with several recent works that address issues somewhat similar to WSOL. However, these related works do not have such a diverse and rich set of reusability constructs. In our opinion, this is one of the advantages of WSOL [2]. We believe that integration of WSOL reusability constructs into the related works, as well as eventual future standards in this area, would be beneficial. Therefore, in this paper, we explain WSOL reusability constructs and their potential influences on related works. We assume that the reader is familiar with WSDL and the Extensible Markup Language (XML).

The paper is organized as follows. In this section, we have summarized the motivation for WSOL reusability constructs and the motivation for writing this paper. In the next section, we give a brief overview of WSOL and the most important related works. The core of the paper is Section 3, where we explain and illustrate WSOL reusability constructs. Section 4 discusses potential influences between reusability constructs in WSOL and in major related works. In the final section, we summarize how the WSOL reusability constructs are useful for Web Services and note items for future work.

Our other publications on WSOL discuss and illustrate different aspects of WSOL and its management infrastructure, the Web Service Offerings Infrastructure (WSOI). [1] and [2] present WSOL and WSOI, [3] compares WSOL and related works, while [4] contains detailed information about the WSOL syntax and its examples. (Note that WSOL was improved since the publication of [4]). Our research report [5] is an extended version of this paper and contains examples and additional details on the topics we discuss hereafter.

2 A Brief Overview of WSOL and the Related Work

The Web Service Offerings Language (WSOL) is a language for the formal specification of classes of service, various types of constraint, and management statements for Web Services. The syntax of WSOL is defined using XML Schema, in a way compatible with WSDL 1.1. WSOL descriptions of Web Services are specified outside WSDL files.

The crucial concept in WSOL is a **service offering (SO)**. A WSOL service offering is the formal representation of a single class of service of one Web Service [1, 2]. It can also be viewed as a simple contract or SLA between the provider Web Service, the consumer, and eventual management third parties. A Web Service can offer multiple service offerings to its consumers, but a consumer can use only one of them at a time in one session. A Web Service can have in parallel many open sessions, in some cases even several sessions with the same consumer. A WSOL service offering contains the formal definition of various types of constraint and management statements that characterize the represented class of service.

Every WSOL **constraint** contains a Boolean expression that states some condition (guarantee or requirement) to be evaluated. Boolean expressions in constraints can also contain arithmetic, date/time/duration, and some simple string expressions. The constraints can be evaluated before and/or after invocation of operations or periodically, at particular date/time instances. WSOL supports the formal specification of functional constraints (pre-, post-, and future-conditions), quality of service (QoS) constraints (describing performance, reliability, availability, and similar 'extra-functional' properties), and access rights (for differentiation of service).

A WSOL **statement** is any construct, other than a constraint, that states important management information about the represented class of service. WSOL enables the formal specification of statements about management responsibility, subscription prices, pay-per-use prices, and monetary penalties to be paid if constraints are not met. In addition, WSOL has extensibility mechanisms that enable definition of new types of constraint and management statement as XML Schemas.

Apart from definitions of constraints and management statements, service offerings can contain reusability constructs that discussed in detail later in this paper. WSOL reusability constructs determine static relationships between service offerings. These relationships do not change during run-time. One example of a static relationship is when a service offering extends another service offering. Another example is when two service offerings instantiate the same template. Further example is when two service offerings include the same constraints and/or statements. In addition, WSOL enables specification of dynamic (i.e., run-time) relationships between service offerings. These relationships can change during run-time. Dynamic relationships are specified outside definitions of service offerings in the format presented in [1, 4]. A dynamic relationship states what class of service could be an appropriate replacement if a particular group of constraints from the used class of service cannot be met.

We use the term '**WSOL item**' to refer to a particular piece (service offering, constraint, statement, or reusability construct) of a WSOL specification. Some WSOL items can contain other WSOL items.

To verify the WSOL syntax, we have developed a WSOL parser called 'Premier' [4]. To enable monitoring of WSOL-enabled Web Services, metering and calculation of QoS metrics, evaluation of WSOL constraints, accounting and billing, as well as dynamic adaptation of compositions including WSOL-enabled Web Services, we are developing the Web Service Offerings Infrastructure [1, 2].

Our work on WSOL draws from the considerable previous work on the differentiation of classes of service in telecommunications and on the formal representation of various constraints in software engineering. At the beginning of our research, there was no relevant work of this kind for Web Services. In parallel with our work on WSOL, several XML languages with somewhat similar goals have been developed. They are discussed in the remainder of this section. In addition, our work on reusability constructs in WSOL is based upon many works on reusability constructs in other languages, both programming languages and description languages. In the next section, we explain how we have studied these existing reusability constructs for inclusion into WSOL.

The IBM **Web Service Level Agreements (WSLA)** [5] and the HP **Web Service Management Language (WSML)** [6] are two powerful languages for the formal XML-based specification of custom-made SLAs for Web Service. SLAs in these two languages contain only QoS constraints and management information. While these two languages are more powerful than WSOL for the specification of QoS constraints, they do not address the formal specification of functional constraints, access rights, and other constraints. They do not have the concept of a class of service, which is simpler and easier to implement than an SLA. Consequently, the run-time overhead of supporting these languages is higher than the overhead of supporting WSOL. Further, these languages do not have constructs for the specification of relationships between SLAs, while WSOL provides specification of dynamic relationships between classes of service, as well as a diverse set of reusability constructs.

Another language that can be used for specification of SLAs for Web Services is **SLAng** [7]. SLAng enables specification of SLAs not only on the Web Service level, so it has broader scope than WSLA and WSML. 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 and WSML.

WS-Policy [8] is a general framework for the specification of policies for Web Services. A policy can describe any property of a Web Service or its parts, so it corresponds to WSOL concepts of a constraint and a statement. The detailed specification for particular categories of policies will be defined in specialized languages. Currently, specification details are defined only for security policies and to some extent for functional constraints, but not yet for QoS policies, prices/penalties, and other management issues. (Specification of functional constraints is supported, but the contained expressions can be specified in any language.) WS-Policy has a number of good features, such as flexibility, extensibility, and reusability. However, some advantages of WSOL are the explicit support for management applications, built-in support for various types of constraint and statement, unified representation of ex-

pressions, wider range of reusability constructs, and specification of classes of service and relationships between them.

DAML-S (DAML-Services) [9] is a work on semantic description of Web Services, including specification of functional and some QoS constraints. However, constraints in DAML-S are currently not specified in a precise, formal, and detailed notation, so they cannot be used for the actual monitoring, metering, and management. They are only placeholders for the future description of rules. They are specified for a more comprehensive service description, not for the actual control and management, which is the main application area of WSOL. While DAML-S has the concept of a service profile, there is no concept of a class of service.

The **Web Service Modeling Framework (WSMF)** [10] is a conceptual model for the development and description of Semantic Web enabled Web Services (SWWS). WSMF does not define concrete language syntax, but suggests extending existing Web Service languages, such as DAML-S, with missing concepts. WSMF defines four main types of element: ontologies, goal repositories, Web Service descriptions, and mediators. A goal describes objectives accomplished by Web Services and consists of pre- and post-conditions. Web Services descriptions in WSMF also contain pre- and post-conditions, as well as non-functional (QoS) properties. While WSMF is a conceptual model (currently without an implementation), WSOL is a concrete language. However, WSOL and WSMF have some similar concepts and WSOL could be extended with missing WSMF concepts.

To summarize, the main distinctive characteristics of WSOL, compared with the mentioned competing works, are: support for classes of service and their static and dynamic relationships, support for various types of constraint and statement, a diverse set of reusability constructs, features reducing run-time overhead, and support for management applications. A more detailed comparison between WSOL and some of these related works can be found in [3].

3 WSOL Reusability Constructs

A large number of reusability constructs was developed for programming languages, from jumps and loops in assembly languages to hyperspaces and other concepts in modern languages. Of course, not all of them are applicable for a description language like WSOL. Before we have started development of WSOL, we have examined various existing reusability concepts and their usability in the context of WSOL.

We had two partially conflicting sets of goals. On the one hand, we wanted to achieve reusability of WSOL specifications and comprehensive descriptions of relationships between service offerings, as discussed in Section 1. On the other hand, our approach in designing WSOL was to provide solutions that are relatively simple to use and implement and lightweight in terms of run-time overhead. (For this reason, we have used in WSOL classes of service instead of more demanding custom-made SLAs, one language for various types of constraint and management statement instead of several specialized languages, support for management third parties, constraints evaluated periodically, and random evaluation of constraints [3].) To achieve

simplicity and lightweightness of WSOL, we had to select a limited number of reusability constructs. Since WSOL was primarily developed for management of Web Services and their compositions, we were particularly interested in reusability constructs that support management applications, including dynamic adaptation [2].

We have studied a broad set of reusability constructs in programming and specification languages. For example, we have examined types, classes, single and multiple inheritance, polymorphism, overriding, restriction, sets and set operations (e.g., union, intersection, difference), scoping, domains, structs, templates, template instantiation, templatization, macros, include statements, subroutines, contracts, subcontracts, relationship tables, constraint dimensions, policies, roles, refinement, mixins, aspects, composition filters, hyperspaces, monitors, and views. In addition, we have examined three languages—Quality Interface Definition Language (QIDL) [11], Quality Modeling Language (QML) [12], and Quality Description Language (QDL) [13]—developed for the formal specification of QoS for Common Object Request Broker Architecture (CORBA) distributed objects, as well as reusability constructs in these languages.

We had no statistical data about the usage and usefulness of different reusability constructs in languages similar to WSOL. Therefore, we have tried to find some realistic practical examples that could justify inclusion of such constructs into WSOL. We have also tried to estimate how frequently such examples would occur. Further, we have tried to estimate the overhead of reusability constructs on WSOL and its tools. Another area that we have looked at was modeling of some constructs with other constructs, to reduce the total number of WSOL constructs. As already stated, simplicity and lightweightness of WSOL were important issues. After determining options and alternatives, we have analyzed their costs and benefits and determined priorities for reusability constructs in WSOL.

As a result of our study, WSOL now contains the following **reusability constructs**:

1. The definition of service offerings (SOs).
2. The definition of constraint groups (CGs).
3. The definition and instantiation of constraint group templates (CGTs).
4. The extension (single inheritance) of service offerings, constraint groups, and constraint group templates.
5. The inclusion of already defined constraints, statements, and constraint groups.
6. The specification of applicability domains.
7. The declaration of operation calls.

We explain and illustrate these WSOL constructs in the following seven subsections. Precise syntax definitions (for an earlier version of WSOL) and further examples are given in [4]. (We have recently made the syntax of WSOL statements more consistent and more versatile, but we have also kept the old syntax for backward compatibility reasons. The examples in this paper and our earlier publications use the old syntax, while the new syntax will be explained and illustrated in a forthcoming publication. Further, several recent conceptual improvements of WSOL mentioned in this paper will also be explained in detail and illustrated in a forthcoming publication.)

```
<wsol:serviceOffering name = "SO1" service = "buyStock:buyStockService"
accountingParty = "WSOL-SUPPLIERWS" >
  <wsol:constraint name = "QoScons2" xsi:type = "qosSchema:QoSconstraint"
service = "WSOL-ANY" portOrPortType = "WSOL-EVERY" operation =
"WSOL-EVERY" >
    <expressionSchema:booleanExpression>
      <expressionSchema:arithmeticWithUnitExpression>
        <expressionSchema:QoSmetric metricType =  "QoSMetricOntology:
ResponseTime" service = "WSOL-ANY" portOrPortType = "WSOL-ANY"
operation = "WSOL-ANY"            measuredBy = "WSOL_INTERNAL" />
      </expressionSchema:arithmeticWithUnitExpression>
      <expressionSchema:arithmeticComparator type = "&lt;" />
      <expressionSchema:arithmeticWithUnitExpression>
        <wsol:numberWithUnitConstant>
          <wsol:value> 0.3 </wsol:value>
          <wsol:unit type = "QoSMeasOntology:second" />
        </wsol:numberWithUnitConstant>
      </expressionSchema:arithmeticWithUnitExpression>
    </expressionSchema:booleanExpression>
  </wsol:constraint>
  …
  <wsol:managementResponsibility  name = "MangResp1" >
    <wsol:supplierResponsibility scope = "tns:AccRght1" />
    <wsol:consumerResponsibility scope = "tns:Precond3" />
    <wsol:independentResponsibility scope = "tns:QoScons2"    entity =
"http://www.someThirdParty.com" />
  </wsol:managementResponsibility>
</wsol:serviceOffering>
```

Fig. 1. Example Definition of a Service Offering (SO)

In the eighth subsection, we briefly discuss the definition of external ontologies of QoS metrics, measurement units, and currency units used in WSOL files. The definition of such ontologies is not a part of the WSOL language, but these ontologies can be reused across WSOL service offerings of different Web Services.

3.1 Definition of Service Offerings (SOs)

A WSOL **definition of a service offering (SO)** contains formal definition of new constraints and statements that categorize the class of service represented with this service offering. It can also contain reusability constructs discussed in the following subsections. For example, constraints and statements that were already defined elsewhere can be included in a service offering using the WSOL inclusion reusability

construct, discussed in Subsection 3.5. Further, a new service offering can be defined as an extension of an existing service offering, as discussed in Subsection 3.4. If inside one service offering two or more constraints of the same type (e.g., two preconditions) are defined for the same operation, they all have to be satisfied. This means that the Boolean 'AND' operation is performed between such constraints.

Every service offering has a name and exactly one accounting party. An accounting party is the management party responsible for logging all SOAP messages related to this service offering and for calculating monetary amounts to be paid [1]. We have recently added into WSOL optional specification of validity duration and/or expiration time of a service offering. If no validity duration or expiration time is specified, consumers can use the service offering until its deactivation or explicit switching of service offerings [2].

Figure 1 shows parts of an example definition of a service offering. In the given example, the service offering '*SO1*' is applied to the Web Service '*buyStockService*', as specified in the attribute '*service*' discussed in Subsection 3.6. Figure 1 shows three WSOL items – the service offering '*SO1*', the QoS constraint '*QoScons2*', and the management responsibility statement '*MangResp1*'. '*SO1*' contains '*QoScons2*' and '*MangResp1*'. Definitions of WSOL service offerings can be long and complex. We have omitted other WSOL items in '*SO1*' for brevity. More detailed explanation of this example is given in [1].

In most cases, a WSOL service offering is specified for a particular Web Service. Different consumers of this Web Service can use the same service offering, potentially in parallel. However, if a service offering contains constraints and statements that do not reference particular ports and operations, it can be provided by different Web Services. This feature enables development of libraries of reusable service offerings. When the concept of a 'service type' becomes standard in WSDL, it will be straightforward to update WSOL with the definition of a reusable service offering specified for a WSDL service type.

3.2 Definition of Constraint Groups (CGs)

WSOL has a special reusability construct for the **definition of constraint groups**. A **constraint group (CG)** is a named set of constraints and statements. It can also contain reusability constructs, including definitions of other constraint groups. The number of levels in such **nesting of constraint groups** is not limited. (Since a constraint group can contain not only constraints, but also statements and reusability constructs, the name 'item group' would be more appropriate. We have kept the name 'constraint group' for compatibility with early versions of WSOL.) Analogously to service offerings, a new constraint group can be defined as an extension of an existing constraint group. In the earlier versions of WSOL, the Boolean 'AND' operation was always applied between constraints in the same constraint group. However, we have recently added constraint groups in which only some constraints have to be satisfied (at least one constraint or exactly one constraint).

Figure 2 illustrates the definition of a constraint group in WSOL. The constraint group named '*CG7*' extends another constraint group '*CG6*'. In addition, it contains the definition of the constraint (post-condition) '*C7*', the definition of the nested constraint group named '*CG8*', an instantiation of the constraint group template '*CGT2*' (defined in Figure 3 and discussed in the next subsection), and the inclusion of the QoS constraint '*QoScons2*' defined in the service offering '*SO1*' (shown in Figure 1). The details of '*C7*' and '*CG8*' are omitted for brevity. We discuss the domain attributes '*service*', '*portOrPortType*', and '*operation*' in Subsection 3.6.

There are important syntax similarities between a WSOL service offering and a constraint group. Definitions of both service offerings and constraint group can contain constraints, statements, and reusability constructs (including definitions of constraint groups). These definitions also have some same domain attributes, discussed later in this paper. Both service offerings and constraint groups support extension. While definitions of constraint groups and service offerings have some syntax similarities, they also have semantic and syntax differences. The crucial semantic difference is that consumers can choose and use service offerings, not constraint groups. Service offerings are used for the definition of complete and coherent offerings to consumers. On the other hand, the main goal of constraint groups is the easier development of WSOL files. They are used for smaller sets of constraints and statements. These sets can, but need not, be complete from the consumer viewpoint. An important syntax consequence is that accounting party, validity duration, and expiration time can be specified only for a service offering, not a constraint group. Further, dynamic relationships can be specified only for service offerings, not for constraint groups. Another important syntax difference is that constraint groups can be nested, while service offerings must not be. In addition, while the Boolean 'AND' operation is always applied between constraints in a service offering, some other combinations of constraints can also be specified for constraint groups.

```
<wsol:CG name = "CG7" extends = "tns:CG6" service = "buyStock:
buyStockService" portOrPortType = "WSOL-MANY" operation = "WSOL-MANY"
>
  <wsol:constraint name = "C7" xsi:type = "postConditionSchema:postCondition"
service = "buyStock:buyStockService" portOrPortType = "buyStock:
buyStockServicePort" operation = "buyStock:buySingleStockOperation" >
    …
  </wsol:constraint>
  …
  <wsol:CG name = "CG8" service = "buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation = "buyStock:buySingleStockOperation" >
    …
  </wsol:CG>
  …
  <wsol:instantiate CGTName = "tns:CGT2" resService = "buyStock:
buyStockService" resPortOrPortType = "WSOL-MANY" resOperation =
"WSOL-MANY" resCGName = "CG9" >
    <wsol:parmValue name = "minAvail" >
      <wsol:numberWithUnitConstant>
        <wsol:value> 95 </wsol:value>
        <wsol:unit type = "QoSMeasOntology:Percentage" />
      </wsol:numberWithUnitConstant>
    </wsol:parmValue>
  </wsol:instantiate>
  …
  <wsol:include constructName = "SO1ns:SO1.QoScons2" resService = "buyStock:
buyStockService" resPortOrPortType = "WSOL-ANY" resOperation =
"WSOL-ANY" resName = "QoScons2buyStock" />
  …
</wsol:CG>
```

Fig. 2. Example Definition of a Constraint Group (CG)

The WSOL concept of a constraint group has several benefits. First, a constraint group can be reused across service offerings as a unit, using the WSOL inclusion reusability construct discussed in Subsection 3.5. Second, it is possible to specify in a single management responsibility statement that all constraints from a constraint group are evaluated by the same management entity. Third, constraints in different constraint groups can have the same relative constraint name, so using constraint groups enables name reuse. Fourth, constraint groups can be used for different logical groupings of constraints and statements. For example, one can use a constraint group to group constraints and statements related to a particular port, port type, or operation. In this way, the concept of a constraint group complements the concept of a service offering that assembles constraints and statements on the level of a Web

Service. In addition, one can use constraint groups to define aspects of service offerings. For example, one can group all functional constraints for one port type into one constraint group, QoS constraints for the same port type into another constraint group, and access rights for this port type into a third constraint group. This supports separation of concerns.

3.3 Definition and Instantiation of Constraint Group Templates (CGTs)

A **constraint group template (CGT)** is a parameterized constraint group. (Analogously to constraint groups, the name 'item group template' would be more appropriate.) This WSOL reusability construct is very useful because different classes of service often contain constraints with the same structure, but with different numerical values. WSOL contains two constructs for constraint group templates – one for their definition and the other for their instantiation. At the beginning of a **definition of a constraint group template**, one defines one or more abstract constraint group template parameters, each of which has a name and a data type. Constraint group template parameters often have the data type '*numberWithUnit*', which requires additional information about the used measurement unit. The definition of constraint group template parameters is followed by the definition of constraints, statements, and reusability constructs in the same way as for constraint groups. Constraints inside a constraint group template can contain expressions with constraint group template parameters.

Figure 3 shows an example definition of a CTG, named '*CGT2*'. This constraint group template has one parameter '*minAvail*', of the data type '*numberWithUnit*' and the measurement unit '*Percentage*' (defined in an external ontology of measurement units). The periodic QoS constraint '*QoScons4*' states that the measured value of the QoS metric '*DailyAvailability*' (defined in an external ontology of QoS metrics) must be greater than or equal to the value of the parameter *minAvail*'. This periodic QoS constraint is evaluated daily, at midnight, between the given start and stop date and no more than the given maximum number of repetitions.

An **instantiation of a constraint group template** provides concrete values for all constraint group template parameters. The result of every such instantiation is a new constraint group. One constraint group template can be instantiated many times, in different service offerings and for different Web Services. Two instantiations of the same constraint group template can provide different parameter values. For example, '*CGT2*' from Figure 3 can be instantiated with parameter values '*90 percents*', '*95 percents*', '*99 percents*', etc. One such instantiation of '*CGT2*' with the parameter value '*95 percents*' is shown in the middle of Figure 2. The result of this instantiation is the new constraint group '*CG9*' inside the defined '*CG7*'. Currently WSOL does not support partial instantiation of a constraint group template. In such an instantiation, values for only some constraint group template parameters would be supplied and the result would be a new constraint group template.

```
<wsol:CGT name = "CGT2" service = "WSOL-ANY" portOrPortType =
"WSOL-MANY" operation = "WSOL-MANY" >
  <wsol:parameter name= "minAvail" dataType = "wsol:numberWithUnit" unit =
"QoSMeasOntology:Percentage" />
  <wsol:constraint name = "QoScons4" xsi:type = "PeriodQosSchema:
PeriodicQoSconstraint" service = "WSOL-ANY" >
    <wsol:startTime time = "2003-04-01T00:00:00-05:00" />
    <wsol:period interval = "P1D" />
    <wsol:stopTime time = "2004-04-01T00:00:00-05:00" />
    <wsol:maxRepetitions number = "366" />
    <expressionSchema:booleanExpression>
      <expressionSchema:arithmeticWithUnitExpression>
        <expressionSchema:QoSmetric metricType = "QoSMetricOntology:
DailyAvailability" service = "WSOL-ANY" portOrPortType = "WSOL-ALL"
operation = "WSOL-ALL" measuredBy = "WSOL_INTERNAL" />
      </expressionSchema:arithmeticWithUnitExpression>
      <expressionSchema:arithmeticComparator type = ">=" />
      <expressionSchema:arithmeticWithUnitExpression>
        <expressionSchema:arithmeticWithUnitVariable      aWUName =
"tns:CGT2.minAvail" />
      </expressionSchema:arithmeticWithUnitExpression>
    </expressionSchema:booleanExpression>
  </wsol:constraint>
</wsol:CGT>
```

Fig. 3. Example Definition of a Constraint Group Template (CGT)

During instantiation, absolute names of constraints and statements in the instantiated constraint group templates change. To explain this WSOL feature, let us summarize naming in WSOL. All WSOL constraints, statements, service offerings, constraint groups, constraint group templates, and declared operation calls are named. We make a difference between relative and absolute names. A relative name of a WSOL item is a string that must be unique inside the containing service offering, constraint group, or constraint group template. An absolute name of a WSOL item is a namespace-qualified string that contains the relative name of this item, the relative names of all containing service offerings, constraint groups, and constraint group templates, and the namespace of the most general of these relative names. For example, '*CGT2*' shown in Figure 3 (we assume it is defined in the namespace '*ns2*') contains the constraint '*QoScons4*'. The relative name of this constraint is '*QoScons4*', while the absolute name is '*ns2:CGT2.QoScons4*'. When '*CGT2*' is instantiated inside '*CG7*' (shown in Figure 2), it becomes a new nested constraint group '*CG9*'. (We assume that '*CG7*' is defined in the namespace '*ns7*'.) During the

instantiation, the relative name of the constraint '*QoScons4*' does not change. However, its new absolute name inside '*CG7*' becomes '*ns7:CG7.CG9.QoScons4*'.

The WSOL concept of a constraint group template currently has some limitations, adopted for simpler implementation. The WSOL concept of a constraint group template currently has some limitations, adopted for simpler implementation. Constraint group templates cannot be defined inside service offerings, constraint groups, or other constraint group templates. They can be defined only as separate, non-contained, items in WSOL files. This also means that definitions of constraint group templates must not be nested. Also, since constraints inside a constraint group template may contain expressions with constraint group template parameters, these constraints must not be included inside other constraint group templates, constraint groups, or service offerings. In addition, constraint group templates cannot be defined inside service offerings and constraint groups. They can be defined only as separate, non-contained, items in WSOL files.

3.4 Extension (Single Inheritance)

WSOL supports **extension (single inheritance) of service offerings, constraint groups, and constraint group templates** with their **'*extends*' attribute**. This enables easy definition of new WSOL items (service offerings, constraint groups, or constraint group templates) from existing ones. The extending WSOL item inherits all constraints, statements, and reusability constructs from the extended WSOL item and can define or include additional ones. The specification of extension using the '*extends*' attribute is illustrated in Figure 2, where '*CG7*' is defined as an extension of the previously defined '*CG6*' (not shown in Figure 2). We have also studied the use of multiple inheritance in WSOL, but decided not to support it due to complexities, such as the possibility of name conflicts and conflicts in accounting parties. However, effects similar to multiple inheritance of constraint groups can be achieved by including multiple constraint groups inside a new constraint group. This is discussed in the next subsection.

Note that the extending WSOL items are not subcontracts (as defined in [15]) of the extended WSOL items. A subcontract weakens pre-conditions and strengthens post-conditions. This means that between the inherited and the additional pre-conditions the logical 'OR' operation is preformed, while for pre-conditions the logical 'AND' operation is preformed. In WSOL, the logical 'AND' operation is performed between the inherited and the additional constraints (unless these constraints are in a constraint group in which only some constraints have to be satisfied). This approach was chosen due to simplicity. It addresses reuse of WSOL items. For comparisons of WSOL service offerings, constraint groups, and constraint group templates, it would be useful to also support specification of subcontracts. We plan to address this issue in a future version of WSOL.

3.5 Inclusion

During definition of a new service offering, constraint group, or constraint group template, some of the constraints, statements, or constraint groups might have been already defined elsewhere. The WSOL **inclusion reusability construct** enables incorporating such constraints, statements, or constraint groups with a single XML element, instead of writing again their complete definitions. Reusable constraints, statements, and constraint groups can be defined once and included many times, across different service offerings, constraint groups, and/or constraint group templates, and even across different Web Services. Such reusable constraints, statements, and constraint groups can be defined inside service offerings, but also outside any service offering, e.g., in libraries of reusable WSOL items. The absolute name of the included constraint, statement, or constraint group changes during inclusion. In addition, its relative name can (but need not) be changed. The change of the relative name is useful when the inclusion also makes the applicability domain more specific, as mentioned in the next subsection. The WSOL inclusion reusability construct is illustrated in Figure 2, where the constraint '*QoScons2*' (defined in Figure 1 inside '*SO1'*) is included in '*CG7*'. The relative name of this included constraint is changed to '*QoScons2buyStock*', as specified with the '*resName*' attribute.

The inclusion of constraint groups can be used to model multiple inheritance. Let us assume that constraint groups '*CG1*', '*CG2*', and '*CG3*' all contain different constraints named '*C1*' and that a new constraint group '*CG4*' has to include all constraints, statements, and nested constraint groups from '*CG1*', '*CG2*', and '*CG3*'. Using multiple inheritance would lead to a name conflict between the three '*C1*' constraints. To avoid the name conflict, WSOL supports only the inclusion of '*CG1*', '*CG2*', and '*CG3*' in '*CG4*'. In this case, the three constraints can be referenced as '*CG1.C1*', '*CG2.C1*', and '*CG3.C1*'.

In the same way, the inclusion mechanism could be used to model the single inheritance of constraint groups, instead of using the extension mechanism. Nonetheless, we see benefits of supporting the extension of constraint groups in WSOL. First, service offerings and constraint group templates must not be nested or included, so for them the inclusion mechanism is not a replacement for the extension mechanism. The extension of constraint groups is analogous to the extension of service offerings and constraint group templates, so it can be supported for consistency of WSOL reusability constructs. Second, when extension is used, the absolute names of constraints are shorter. For example, suppose '*CG12*' (defined in some namespace '*ns12*') extends '*CG11*', which in turn extends '*CG10*' and there is a constraint '*C10*' in '*CG10*'. Then, the inherited constraint '*C10*' in '*CG10*' has the absolute name '*ns12:CG12.C10*'. However, if '*CG12*' includes '*CG11'*, which in turn includes '*CG10*', then '*C10*' has the absolute name '*ns12:CG12.CG11.CG10.C10*'. Third, the extension mechanism is relatively simple, easy to implement, and well know to potential developers of WSOL files from object-oriented programming.

We have recently built into WSOL optional naming of expressions, which are parts of constraints, and the possibility of inclusion of expressions into other expressions. This feature further improves reusability of WSOL constraints. On the other

hand, names of constraints currently cannot be used as Boolean variables in WSOL expressions. This would enable inclusion of constraints in the middle of expressions defining other constraints. However, we would have to decide whether this is just inclusion of the constraint expression or the value of the constraint (potentially evaluated by another management party) is referenced. The benefits of the former approach can also be achieved with inclusion of expressions, while the latter approach involves some subtleties. Therefore, we have currently built into WSOL only the construct for inclusion of expressions.

3.6 Specification of Applicability Domains

All WSOL constraints, statements, service offerings, constraint groups, constraint group templates, and operation calls are defined for some applicability domain. An **applicability domain** defines to which operations, port types, ports, and/or Web Services the given WSOL item applies. Declarations of used QoS metrics, discussed in Subsection 3.8, also contain specification of applicability domains, denoting for which operations, port types, ports, and/or Web Services the QoS metric is measured or computed.

In WSOL, an applicability domain is specified using the **attributes '*service*', '*portOrPortType*', and '*operation*'**. Definitions of constraint group templates, constraint groups, statements, and non-periodic constraints, as well as declarations of operation calls and used QoS metrics have these three applicability domain attributes. For example, the definition of the QoS constraint '*QoScons2*' in Figure 1 contains all three attributes. However, definitions of service offerings and periodic QoS constraints have only the attribute '*service*' and do not have attributes '*portOrPortType*' and '*operation*'. This is because they do not describe an operation, port type, or port, but a complete Web Service. For example, the definition of the service offering '*SO1*' in Figure 1 and the definition of the periodic QoS constraint '*QoScons4*' in Figure 3 both have only the attribute '*service*'.

In the following paragraphs, we present details of the WSOL concept of applicability domains. First, we define the concepts of specific and abstract applicability domains. Then, we discuss how actual applicability domains are determined. Next, we introduce four special constants for applicability domain attributes. After that, we present rules for relationships between applicability domains of containing and contained WSOL items. In this respect, we define the terms 'subdomain', 'more specific' and 'more abstract'. Finally, we discuss specialization of applicability domains during inclusion and constraint group template instantiation.

3.6.1 Specific and Abstract Applicability Domains

We categorize applicability domains into specific and abstract. A **specific applicability domain** refers to a particular operation (or a group of operations) of a particular port (or a group of ports) of a particular Web Service. For example, the functional pre-condition '*C7*' shown in Figure 2 has a specific applicability domain – '*buySingleStockOperation*' of '*buyStockServicePort*' of '*buyStockService*'. On the other

hand, in an **abstract applicability domain**, the value of one or more of the applicability domain attributes refers to any operation, port type, port, and/or Web Service. For example, the periodic QoS constraint '*QoScons4*' shown in Figure 3 has an abstract applicability domain – any Web Service. Another example of an abstract applicability domain is 'a particular operation of a particular port type (but not a particular port of a particular Web Service)'.

The benefit of having abstract applicability domains is that WSOL items that have abstract applicability domains can be included (in the case of constraint group templates: instantiated) in different service offerings, constraint groups, and constraint group templates. This means that using abstract applicability domains is beneficial for reusability of WSOL items. On the other hand, to perform monitoring and management activities with WSOL service offerings, it is necessary to know precisely for which Web Service, port, and operation a WSOL constraint should be evaluated (or a QoS metric should be measured or computed). Consequently, using specific applicability domains is important for manageability of WSOL items. Since WSOL was developed primarily for management applications discussed in [1], manageability is more important than reusability. To conclude, the goals of reusable and manageable WSOL items conflict with each other.

3.6.2 Actual Applicability Domain

To provide some support for reusability, while ensuring manageability of WSOL items, we have developed the concept of an actual applicability domain. The **actual applicability domain** of a contained WSOL item is calculated as an intersection of the values of its applicability domain attributes and the actual applicability domain of its immediate containing WSOL item (service offering, constraint group, or constraint group template). If the contained WSOL item is defined with an abstract applicability domain and the containing WSOL item has a specific actual applicability domain, then the actual applicability domain of the contained WSOL item can be specific. In Figure 1, '*QoScons2*' is defined with an abstract applicability domain – every operation in every port of any Web Service. However, when it is inside '*SO1*', it will be evaluated only for operations of '*buyStockService*', the Web Service for which '*SO1*' is specified. Consequently, the actual applicability domain of '*QoScons2*' inside '*SO1*' is specific. In this way, WSOL items can be both reusable and precise enough for management activities.

To ensure that applicability domains of containing WSOL items and their contained WSOL items produce meaningful actual applicability domains and do not conflict, we have defined rules for relationships between these applicability domains. In addition, we have defined rules for the specialization of domains during inclusion of items with abstract applicability domains. While these rules support both manageability and reusability of WSOL items, they also introduce some additional complexity into WSOL. We summarize these rules later in this paper, while their more detailed discussion and illustration can be found in [4].

3.6.3 Special Constants for Applicability Domain Attributes

Before we discuss the rules mentioned above, let us introduce four special constants used in WSOL applicability domain attributes. Detailed explanation and examples of these four constants are given in [4], but their introduction in this paper should help the reader to understand the given WSOL examples more easily.

To represent special values of the applicability domain attributes '*service*', '*portOrPortType*', and '*operation*', WSOL has built-in constants '*WSOL-ANY*', '*WSOL-EVERY*', '*WSOL-MANY*', and '*WSOL-ALL*'. '*WSOL-ANY*' is used in abstract applicability domains, while the latter three constants are treated as specific applicability domains.

1. **'*WSOL-ANY*'** is used for representing abstract applicability domains. For example, the QoS constraint '*QoScons2*', shown in Figure 1, can be applied to any Web Service, as specified in the attribute '*service*'.
2. **'*WSOL-EVERY*'** is used to denote that a WSOL item is applied to every operation inside the given port and/or every port inside the given Web Service. This constant must not be used as a value for the attribute '*service*'. For example, the QoS constraint '*QoScons2*', shown in Figure 1, is applied to every operation of every port of whatever Web Service '*QoScons2*' is applied to. (As mentioned before, '*QoScons2*' inside '*SO1'* is applied to '*buyStockService*'.)
3. **'*WSOL-MANY*'** is used to denote that a constraint group or constraint group template contains WSOL items that are applied to different operations of the same port or port type and/or different ports of the same Web Service. This constant can be used only for the attributes '*operation*' and '*portOrPortType*' of constraint groups and constraint group templates. For example, the constraint group '*CG7*', shown in Figure 2, contains WSOL items that are applied to different operations and ports of the Web Service '*buyStockService*'.
4. **'*WSOL-ALL*'** is used only for the attributes '*operation*' and '*portOrPortType*' of declarations of used QoS metrics. When it is used only for the attribute '*operation*', it specifies that the calculation of the QoS metric uses measurements for all operations of the given port or port type. When it is used for both attributes '*operation*' and '*portOrPortType*', it specifies that the calculation of the QoS metric uses measurements for all operations of all ports of the given Web Service. For example, the QoS metric of the type '*DailyAvailability*', shown in Figure 3, uses availability data for all operations of all ports. It represents daily availability for the whole Web Service, not for a particular operation or port.

3.6.4 Rules for Relationships between Applicability Domains

Let us now summarize the **relationships between applicability domains of containing and contained WSOL items**. The containing WSOL items can be a service offering, a constraint group, or a constraint group template. The contained WSOL item can be a constraint, a statement, or a constraint group. For brevity, we denote the applicability domain of the contained WSOL item as '*AD-In*' and the applicability domain of the containing WSOL item as '*AD-Out*'.

In order for 'AD-In' and 'AD-Out' not to conflict, it must be:

1. that '*AD-In*' and '*AD-Out*' are equivalent, or

2. that '*AD-In*' is a subdomain of '*AD-Out*', or
3. that '*AD-In*' is more abstract than '*AD-Out*'.

All other cases are considered semantic errors in WSOL. Let us explain the terms 'subdomain' and 'more abstract' and illustrate them with examples.

In WSOL, '**subdomain**' means that '*AD-In*' is a part of '*AD-Out*'. For example, in Figure 2, the applicability domain of the contained functional post-condition '*C7*' is the operation '*buySingleStockOperation*' of the port '*buyStockServicePort*' of the Web Service '*buyStockService*', while the applicability domain of the containing constraint group '*CG7*' is the whole Web Service '*buyStockService*'. This means that the applicability domain of '*C7*' is a subdomain of the applicability domain of '*CG7*'. This rule was built into WSOL for easier discovery and prevention of semantic errors when a syntactically correct WSOL item is defined, included or (in case of constraint group templates) instantiated in the wrong context. An example of such semantic error is when one defines a functional pre-condition for the operation '*Op1*' of some Web Service '*WS1*' and then tries to include this pre-condition for a service offering of another Web Service '*WS2*' that does not have the same operation '*Op1*'.

On the other hand, '**more abstract**' means that '*AD-Out*' is a special case of '*AD-In*'. The opposite term is '**more specific**'. For example, in Figure 1, the applicability domain of the contained QoS constraint '*QoScons2*' is 'every operation of every port of any Web Service', while the applicability domain of the containing service offering '*SO1*' is a particular Web Service, '*buyStockService*' (and all its ports and operations). This means that the applicability domain of '*QoScons2*' is more abstract than the applicability domain of '*SO1*'. This rule, an exception to the rule discussed in the previous paragraph, was built into WSOL to enable specification of reusable abstract applicability domains. For example, '*QoScons2*' can be reused across different service offerings, constraint groups, and/or constraint group templates, such as '*CG7*' shown in Figure 2.

3.6.5 Specialization of Applicability Domains

During constraint group template instantiation and inclusion, the applicability domain of the instantiated constraint group template or the included constraint, statement, or constraint group can be made more specific. In other words, some or all applicability domain attribute values that were '*WSOL-ANY*' can be substituted with the names of particular operations, port types, ports, and/or Web Services. '*WSOL-ANY*' can also be replaced with one of the constants '*WSOL-EVERY*', '*WSOL-MANY*', or '*WSOL-ALL*' where the use of the latter constants is allowed. This **specialization of the applicability domain** is expressed with the **attributes *'resService', 'resPortOrPortType',* and *'resOperation'*** of the WSOL constructs for constraint group template instantiation and inclusion.

For example, the constraint group template '*CGT2*', defined in Figure 3, is instantiated inside the constraint group '*CG7*', as shown in the middle of Figure 2. Since the applicability domain of '*CGT2*' is any Web Service, while the applicability domain of '*CG7*' is the Web Service '*buyStockService*', the specialization of the applicability domain was used. This was done with the '*resService*' attribute in the constraint group template instantiation construct. The values of the attributes '*resPor-*

tOrPortType' and '*resOperation*' are the same as for the attributes '*PortOrPortType*' and '*Operation*' of '*CGT2*' and could have been omitted. Another example is the inclusion of the QoS constraint '*QoScons2*', defined in Figure 1, into the constraint group '*CG7*', as shown near the bottom of Figure 2. Again, the applicability domain is specialized from any Web Service to '*buyStockService*'.

Note that in the discussed two examples the specialization of applicability domains was not strictly necessary because of the rule for determining actual applicability domains. An example where such specialization is necessary is outlined next. Suppose a reusable QoS constraint is defined outside any service offering, constraint group, or constraint group template with the applicability domain 'any operation of any port of any service'. When it is needed to apply this QoS constraint to only one particular operation of a particular port of a particular Web Service and such QoS constraint is included in some service offering, the specialization of the applicability domain attributes '*operation*' and '*portOrPortType*' is necessary.

The specialization of applicability domains during constraint group template instantiation and inclusion improves both reusability and manageability of WSOL items. In particular, the reusability of WSOL items with abstract applicability domains is enhanced. On the other hand, the manageability is improved because the additional specific information about applicability domains supports precise monitoring of particular Web Service operations.

3.7 Declaration of Operation Calls

Expressions in WSOL constraints can contain calls to operations implemented by another Web Service, by the management entity evaluating the given constraint, or by the Web Service for which the constraint is evaluated. (In the latter two cases, although the called operations are described in WSDL, they are invoked using internal mechanisms, without any SOAP call.) A **declaration of an operation call** enables referencing results of the same operation call (i.e., the same invocation) in different sub-expressions of one constraint or in several related constraints, without re-invoking the operation.

Figure 4 shows an example declaration of an operation call and its use in WSOL expressions. At the top of the figure, the call to the operation '*currentTime*' of the port type '*timeLookupPortType*', implemented at the port '*timeLookupServicePort*' of the '*timeLookupService*' Web Service, is declared. For easier parsing and checking of WSOL files, the information about both the abstract port type and the particular port called is given in declarations of operation calls, although only the latter information is strictly necessary. This operation call is denoted '*extOp6*'. After this declaration of an external operation call, Figure 4 shows the declaration of the access right '*AccRight6*', where the results of '*extOp6*' are used. '*AccRight6*' allows access to any operation of '*buyStockServicePort*' of '*buyStockService*' only in the period from 9AM to 5PM. In the definition of '*AccRight6*', the message part '*currentTime*' of the response message '*currentTimeResponse*' of the operation called in '*extOp6*' is referenced twice – first to compare it with 9AM and then to compare it with 5PM.

```
<wsol:externalOperationCall callID = “extOp6” calledOperDescription =
“timeLookup:timeLookupPortType.currentTime”  calledOperImplementation =
“timeLookup:timeLookupService.timeLookupServicePort” service = “buyStock:
buyStockService”  portOrPortType = “buyStock: buyStockServicePort” operation =
“WSOL-ANY” />

<wsol:constraint name = “AccRight6” xsi:type = “accessRightSchema:accessRight”
service = “buyStock:buyStockService” portOrPortType = “buyStock:
buyStockServicePort” operation = “WSOL-ANY” >
  <expressionSchema:booleanExpression>
    <expressionSchema:booleanExpression>
      <expressionSchema:timeExpression>
        <expressionSchema:externalOperationResult callID = “tns:extOp6” resultPart-
Name = “timeService: currentTimeResponse.currentTime” />
      </expressionSchema:timeExpression>
      <expressionSchema:timeComparator type = “==” />
      <expressionSchema:timeExpression>
        <expressionSchema:timeOperator type = “AFTER” />
        <expressionSchema:timeExpression>
          <expressionSchema:timeConstant>
            <expressionSchema:time value = “09:00:00-05:00” />
          </expressionSchema:timeConstant>
        </expressionSchema:timeExpression>
      </expressionSchema:timeExpression>
    </expressionSchema:booleanExpression>
    <expressionSchema:binaryBooleanOperator type = “AND” />
    <expressionSchema:booleanExpression>
      <expressionSchema:timeExpression>
        <expressionSchema:externalOperationResult callID = “tns:extOp6”
resultPartName = "timeService:currentTimeResponse.currentTime” />
      </expressionSchema:timeExpression>
      <expressionSchema:timeComparator type = “==” />
      <expressionSchema:timeExpression>
        <expressionSchema:timeOperator type = “BEFORE” />
        <expressionSchema:timeExpression>
          <expressionSchema:timeConstant>
            <expressionSchema:time value = “17:00:00-05:00” />
          </expressionSchema:timeConstant>
        </expressionSchema:timeExpression>
      </expressionSchema:timeExpression>
    </expressionSchema:booleanExpression>
  </expressionSchema:booleanExpression>
</wsol:constraint>
```

Fig. 4. Example Declaration of an Operation Call and Its Use in Expressions

3.8 External Ontologies and WSOL Declarations of Used QoS Metrics

By '**definition of a QoS metric**' we mean 'all the information necessary to measure this QoS metric or compute it from other QoS metrics, as well as the information necessary to use this QoS metric'. For example, a QoS metric 'response time' can be computed as 'time immediately after the operation execution – time immediately before the operation execution' and its measurement unit can be 'second'.

An important reusability feature of WSOL is that QoS metrics used in WSOL specifications are not actually defined in WSOL files. They are defined in reusable **external ontologies** of QoS metrics [15]. WSOL also supports external ontologies of measurement units and currency units. Ideally, appropriate standardization bodies would define such ontologies and make them well-known. In practice, any other interested party (ranging from powerful multinational companies to interested individuals) can also define and publish such ontologies.

Once a QoS metric is defined in a reusable external ontology, its use for particular operations, port types, ports, and/or Web Services is declared in WSOL files. This is done in the WSOL construct for the **declaration of a used QoS metric**. For example, Figure 1 shows the QoS constraint '*QoScons2*' that compares measured response time, as defined in some external ontology '*QoSMetricOntology*', with the constant '*0.3 seconds*'. The response time is measured for whatever operation '*QoScons2*' is applied to. The value of the attribute '*measuredBy*' states that the response time is measured by the management party that evaluates '*QoScons2*'.

Apart from the increased reusability, the outsourcing of definitions of QoS metrics, measurement units, and currency units can have several other benefits:

First, the use of external ontologies eases comparisons between WSOL service offerings specified for the same or for different Web Services. If QoS metrics, monetary units, and currency units were defined within a service offering, comparisons of service offerings using the same QoS metrics, monetary units, and/or currency units would be harder.

Second, the use of common external ontologies can decrease chance of semantic misunderstanding between provider Web Services and their consumers. Some QoS metrics, such as response time, can be defined in several ways [11]. Even when definitions of QoS metrics are precise, a semantic misunderstanding can occur if the involved parties have different interpretations of the basic terms used in these definitions. For unambiguous understanding of providers and consumers both precise definitions of QoS metrics and the use of common basic terminology are necessary. Ontologies can provide both.

Third, outsourcing definitions of QoS metrics, monetary units, and currency units into reusable ontologies might ease compilation of WSOL files and reduce run-time overhead of their monitoring, metering, and evaluation. This is because optimized code for monitoring and metering of most common QoS metrics (or at least some basic QoS metrics) from such ontologies can be built into the management infrastructure for WSOL. For example, a specialized management third party used for monitoring and metering of QoS metrics (see [1] for more details) can have built-in optimized code for most common QoS metrics, e.g., '*QoSMetricOntol-*

ogy:ResponseTime' used in '*SO1*' (Figure 1). When such a party encounters a reference to a known QoS metric from a known ontology, it does not have to retrieve and compile the definition of this QoS metric. Instead, it can simply reference its existing internal implementation. This example assumes that definitions of particular QoS metrics in this ontology do not change.

We have summarized the requirements for ontologies of QoS metrics, measurement units, and currency units in [11]. The ontologies we currently use for our experiments with WSOL have very simple structure and only a few entries. Our current ontologies of QoS metrics are simple collections of names with information about appropriate data types and measurement units. Similarly, ontologies of measurement units and currency units are simple collections of names without any additional information. We plan more work in this area, including the use of WSOL expressions in such ontologies, in the future.

4 Potential Influences of WSOL Reusability Constructs on Related Works

As overviewed in Section 2, there are several recent works on the formal and precise description of Web Services, their constraints (particularly QoS), and SLAs. Ideally, future standards for the comprehensive description of Web Services, in addition to WSDL, should integrate good features from all relevant works. We believe that WSOL reusability constructs (along with some other WSOL concepts and features, as discussed in [3]) can be a very useful input into such future standards. Some WSOL reusability constructs would also be a useful addition to the competing works.

In particular, WSLA and WSML have relatively few reusability constructs. They both have some support for templates, but not as flexible as WSOL constraint group templates. In addition, WSLA has the concept of an obligation group (similar to the WSOL concept of a constraint group) and the concept of an operation group (more powerful than the specification of specific applicability domains in WSOL). Addition of other WSOL reusability constructs—particularly inclusion and extension (inheritance)—to these languages would enable easier development of WSLA and WSML specifications. Note also that WSLA and WSML files contain precise definitions of the QoS metrics, while we suggest outsourcing these definitions into reusable external ontologies. WSLA, WSML, and WSOL can be good starting points for the standardization of the language for such ontologies. On the other hand, the QoS metrics built into the SLAng schema can be used as input for the development of ontologies.

SLAng has no reusability construct, except that the language schema can be viewed as a predefined template. Consequently, we believe that SLAng would benefit greatly from WSOL reusability constructs discussed in this paper. As a first step, we suggest outsourcing definitions of QoS metrics from the SLAng schema into external ontologies. Further, SLAng could be extended with flexible templates, the contents of which could be defined by developers of SLAng files, not only developers of the SLAng language. The schemas for different types of SLAs already built into the

SLAng language could then be rewritten as templates. Supporting extension of such SLAng templates (and complete SLAs) and inclusion of their parts would also be advantageous.

The influences between reusability constructs in WSOL and WS-Policy can go in both directions. WS-Policy has several good reusability features, many similar to those present in WSOL. It enables definition of policies either inside definitions of subjects to which they refer (e.g., inside WSDL files) or independently from definitions of subjects. In the latter case, the policy and the subject are associated through an external attachment. An external attachment contains definition of an applicability domain, which contains an unordered set of subjects (e.g., WSDL services, ports, and/or operations). This definition of applicability domains is more flexible than the one used in WSOL, but it leaves greater space for semantic errors (e.g., when a policy containing references to message parts is attached to a subject where these message parts do not exist). Next, WS-Policy enables inclusion of parts of policies into other policies. This is analogous to the WSOL inclusion of constraint groups, constraints, statements, and expressions. Further, WS-Policy defines policy operators that group parts of policies. There are three policy operators in WS-Policy: '*All*', '*ExactlyOne*', and '*OneOrMore*'. Since they can be named and included as units, policy operators are similar to WSOL definitions of constraint groups. Our recent WSOL additions of constraint groups and constraint group templates in which only some constraints have to be satisfied and of naming and inclusion of expressions were influenced by WS-Policy.

On the other hand, some WSOL reusability constraints could be introduced into WS-Policy. In particular, policies and their parts may be parameterized, but WS-Policy does not provide further detail about this topic. We suggest extending WS-Policy with a precise definition of templates, similar to WSOL constraint group templates. Also, supporting inheritance, in addition to inclusion, would enable easier comparison of policies. Some other extensions of WS-Policy, such as precise specification of management information for the specified policies, are also needed [3].

DAML-S is object-oriented, so it has the concepts of a class (instantiated by a number of objects) and a property (when referencing an object, it models WSOL inclusion) and supports inheritance of classes. These are powerful reusability constructs. Another reusability concept is that service profiles (describing what the Web Service does), service models (describing how the Web Service works), and service groundings (describing how to access the Web Service) are independent and reusable. However, separation of DAML-S service profiles into smaller reusable units (corresponding to WSOL service offerings and constraint group) and specification of their static and dynamic relationships could be useful. Further, while classes can model templates to some extent, we suggest extending DAML-S with the explicit concept of a template.

WSMF defines the concept of a goal, which can be reused across Web Services. WSOL service offerings can be used for representing such goals. On the other hand, WSMF does not discuss in detail reusability of Web Service descriptions. As discussed above, we view reusability constructs as important mechanism for defining relationships between Web Service descriptions, which are useful for selection of

Web Services and their classes of service. We suggest extending WSMF with a more thorough discussion of various relationships relevant for Web Services.

5 Conclusions and Future Work

WSOL has a diverse set of reusability constructs: definition of service offerings (SOs), definition of constraint groups (CGs), definition and instantiation of constraint group templates (CGTs), extension, inclusion, specification of applicability domains, and declaration of operation calls. These constructs enable defining common or similar parts of WSOL specifications once and using these definitions many times. Consequently, they provide easier specification of new WSOL items from existing WSOL items for the same Web Service or different Web Services.

We see practical applications for all WSOL reusability constructs. For example, the definition of abstract applicability domains enables defining reusable WSOL items (service offerings, constraint groups, constraint group templates, constraints and/or statements) that can be shared between various Web Services through inclusion and constraint group template instantiation. In particular, constraint group templates can be instantiated many times, with different parameter values. Since significant similarities can exist even for classes of service of different Web Services, constraint group templates are particularly reusable WSOL items.

Ideally, an appropriate standardization body (or at least, some powerful companies) would define most common and most reusable Web Service description items (e.g., WSDL port types, WSOL items) and make them well-known. Such libraries of reusable Web Service description items would complement reusable ontologies of QoS metrics, measurement units, and currency units. If such reusable libraries and ontologies were defined and used widely in practice, the heterogeneity of Web Services would be alleviated since it would be easier to compare Web Services and their classes of service, as discussed next.

WSOL reusability constructs determine static relationships between service offerings, which show similarities and differences between service offerings. Consequently, they can be used in the process of selection of Web Services and their service offerings. Such static relationships complement dynamic relationships between service offerings, specified outside definitions of service offerings. 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. WSOL static and dynamic relationships between service offerings can guide this process, but appropriate algorithms and heuristics have to be developed. Further, static relationships between service offerings support the mechanisms for dynamic (i.e., runtime) manipulation of classes of service discussed in [2], particularly dynamic creation of new classes of service.

Several recent related works—WSLA, WSML, WS-Policy, SLAng, DAML-S, and WSMF—address issues that partially overlap with WSOL. In some aspects, they are more powerful than WSOL. However, WSOL also has its advantages [3], such as the richer set of reusability constructs. Future standards for the comprehensive descrip-

tion of Web Services (additional to WSDL) should integrate good features from all relevant works, including WSOL. Some WSOL reusability constructs would also be a useful addition to the competing works.

While the focus of our current and future research is on the WSOI management architecture, we are also working on enhancing the WSOL language. In particular, we have recently extended WSOL in several ways, some of which are related reusability constructs. One example is the addition of constraint groups and constraint group templates in which only some constraints have to be satisfied. Another example is the possibility of naming and inclusion of expressions. These and some other recent WSOL improvements (e.g., the more consistent and more versatile syntax for WSOL statements) will be explained and illustrated in a forthcoming publication. We also plan some other minor additions to reusability constructs in future versions of WSOL. An example is subcontracting of service offerings, constraint groups, and constraint group templates, as discussed in Section 3.4. Other improvements of WSOL, such as specification of service fees for third-party management and accounting parties, are also envisioned. Nonetheless, WSOL is relatively complete and stable and its reusability constructs have both the expressive power and significant practical applications. They can positively influence the competing works and future standards in this area.

References

1. Tosic, V., Pagurek, B., Patel, B. Esfandiari, B., Ma, W.: Management Applications of the Web Service Offerings Language (WSOL). In Proc. of the 15th Conference On Advanced Information Systems Engineering - CAiSE'03 (Velden, Austria, June 2003). Lecture Notes in Computer Science (LNCS), No. 2681. Springer-Verlag (2003) 468-484
2. Tosic, V., Ma, W., Pagurek, B., Esfandiari, B.: On the Dynamic Manipulation of Classes of Service for XML Web Services. In Proc. of the 10th Hewlett-Packard Open View University Association (HP-OVUA) Workshop (Geneva, Switzerland, July 2003). Hewlett-Packard (2003)
3. Tosic, V., Patel, K., Pagurek, B.: WSOL – A Language for the Formal Specification of Classes of Service for Web Services. In Proc. of ICWS'03 - The First International Conference on Web Services (Las Vegas, USA, June 2003). CSREA Press (2003) 375-381
4. Patel, K.: XML Grammar and Parser for the Web Service Offerings Language. M.A.Sc. thesis, Carleton University, Ottawa, Canada. Jan. 30, 2003. On-line at: http://www.sce.carleton.ca/netmanage/papers/KrutiPatelThesisFinal.pdf (2003)
5. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement (WSLA) Language Specification, Version 1.0, Revision wsla-2003/01/28. International Business Machines Corporation (IBM). On-line at: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf (2003)
6. Sahai, A., Durante, A., Machiraju, V.: Towards Automated SLA Management for Web Services. Res. Rep. HPL-2001-310 (R.1), Hewlett-Packard (HP) Labs Palo Alto. July 26, 2002. On-line at: http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf (2002)
7. Lamanna, D.D., Skene, J., Emmerich, W.: SLAng: A Language for Defining Service Level Agreements. In Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems - FTDCS 2003 (Puerto Rico, May 2003). IEEE-CS Press (2003) 100-106

8. Hondo, M., Kaler, C. (eds.): Web Services Policy Framework (WS-Policy), Version 1.0. Dec. 18, 2002. BEA/IBM/Microsoft/SAP. On-line at: ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf (2002)
9. The DAML Services Coalition: DAML-S: Semantic Markup for Web Services. WWW page for DAML-S version 0.7. Oct. 2, 2002. On-line at: http://www.daml.org/services/daml-s/0.7/daml-s.html (2002)
10. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Internet white paper and internal report, Vrije Unversiteit Amsterdam. On-line at: http://informatik.uibk.ac.at/users/c70385/wese/wsmf.paper.pdf (2002)
11. Becker, C., Geihs, K., Gramberg, J.: Representing Quality of Service Preferences by Hierarchies of Contracts. In Proc. of the Workshop Elektronische Dienstleistungswirtschaft und Financial Engineering – FAN'99 (Augsburg, Germany, 1999). Schüling Verlag (1999)
12. Frolund, S., Koistinen, J.: Quality of Service Specification in Distributed Object Systems Design. In Proc. of the 4th USENIX Conference on Object-Oriented Technologies and Systems - COOTS '98 (Santa Fe, USA, Apr. 1998), USENIX (1998)
13. Zinky, J.A., Bakken, D.E., Schantz, R.E.: Architectural Support for Quality of Service for CORBA Objects. John Wiley & Sons, Theory and Practice of Object Systems, Vol. 3, No. 1 (Apr. 1997)
14. Meyer, B.: Applying "Design by Contract". IEEE, Computer, Vol. 25, No. 10 (Oct. 1992) 40-51
15. 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 at CAiSE'02 (Toronto, Canada, May 2002). Lecture Notes in Computer Science (LNCS), No. 2512. Springer-Verlag (2002) 237-247