Management Applications of the Web Service Offerings Language (WSOL)

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

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

Abstract. We discuss possible Web Service Management (WSM) and Web Service Composition Management (WSCM) applications of the Web Service Offerings Language (WSOL) and how the language supports these applications. WSOL is a novel language for the formal specification of classes of service, various constraints (functional constraints, Quality of Service – QoS, and access rights), and management statements (subscription and pay-per-use prices, monetary penalties, and management responsibilities) for Web Services. Describing a Web Service in WSOL, in addition to the Web Services Description Language (WSDL), enables monitoring, metering, and management of Web Services. Metering of QoS metrics and evaluation of constraints can be the responsibility of the provider Web Service, the consumer, and/or one or more mutually trusted third parties (SOAP intermediaries or probes). Further, manipulation of classes of service (switching, deactivation/reactivation, and dynamic creation) can be used for dynamic (i.e., run-time) adaptation and management of Web Service compositions.

1 Introduction

The World Wide Web Consortium (W3C) 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" [1]. Here, 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) language, and the UDDI (Universal Description, Discovery, and Integration) directory.

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 the management of Web Services and Web Service compositions. In particular, WSDL does not support specification of various constraints, management statements, classes of service, Service Level Agreements (SLAs) and other contracts between Web Services. Explicit, precise, and unam-

J. Eder and M. Missikoff (Eds.): CAiSE 2003, LNCS 2681, pp. 468–484, 2003.

biguous specification of such information is crucial for management activities [2, 3, 4]. Therefore, we have decided to develop our own XML language for this purpose and named this language the '**Web Service Offerings Language (WSOL)**'.

WSOL can be used for several purposes. For example, it can be used for selecting Web Services that are best for particular circumstances [2]. However, we are particularly interested in applications in the **Web Service Management (WSM)** and the **Web Service Composition Management (WSCM)**. WSM is the management of a particular Web Service or a group of Web Services within the same domain of management responsibility. On the other hand, WSCM is the management of Web Service compositions (a.k.a. orchestrations, choreographies, flows, networks). While there are important differences between WSM and WSCM [5], WSOL supports both WSM and WSCM. In this paper, we discuss possible WSM and WSCM applications of WSOL and how the language supports these applications.

In this section, we have introduced the general area of our research. In the next, we describe the main constructs and concepts of WSOL. In Section 3, we discuss applications of WSOL in the management of Web Services and how the language supports these applications. In Section 4, we examine how manipulation of WSOL service offerings can be used for dynamic adaptation and management of Web Service compositions. A brief overview of some recent related works is presented in Section 5. We summarize the conclusions and directions for future work in Section 6.

2 The Web Service Offerings Language (WSOL)

WSOL is a language for the formal specification of classes of service, various constraints, and management statements for Web Services. It is an XML-based language compatible with WSDL version 1.1. The syntax of WSOL is defined using XML Schema. In this section, we describe the main characteristics of WSOL, emphasizing those that are crucial for WSM and WSCM applications of WSOL.

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

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

We summarize the main characteristics of these categories of constructs in the following five subsections. Precise syntax, illustrative examples, and discussion of the WSOL language constructs are given in [6] and, partially, in [2].

To verify the WSOL syntax, we have developed a **WSOL parser** called 'Premier' [6]. Its implementation is based on the Apache Xerces XML Java parser. This parser produces a DOM (Document Object Model) tree representation of WSOL files and reports eventual syntax errors and some semantic errors. We have also designed Java classes that will be the results of the compilation of WSOL files. We have not yet finished a code generator that creates the designed Java classes from DOM trees pro-

duced by our WSOL parser. A prototype WSOL compiler would then be a combination of the 'Premier' WSOL parser and this code generator.

2.1 Classes of Service and Service Offerings

When described in WSOL, a provider (supplier) Web Service can offer multiple classes of service to its consumers (requesters). By a '**class of service**' we mean a discrete variation of the complete service and quality of service (QoS) provided by one Web Service. Classes of service of one Web Service refer to the same functionality (i.e., the 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. The benefits of multiple classes of service per one Web Service are advocated in detail in [7].

We define a '**service offering**' in WSOL 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 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. 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. Figure 1 shows an example WSOL service offering. Descriptions of service offerings are usually long and complex, so we have shown in Figure 1 only example parts that will be discussed later in this paper.

2.2 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 the formal specification of:

1. **Functional constraints**. These constraints define conditions that a functionally correct operation invocation must satisfy. They usually check some characteristics of message parts of the invoked operation. WSOL enables specification of pre-, post-, and future-conditions, as well as invariants. The novel concept of a future-condition [2], [5] is introduced to model conditions evaluated some time after the provider finishes execution of the requested operation and sends results to the consumer. It enables specification of operation effects that cannot be easily expressed with post-conditions. An example is delivery confirmation for goods bought using Web Services.
2. **QoS (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. They can be checked for a particular operation invocation or periodically, at specified times. For the specification

of QoS constraints, WSOL needs external ontologies of QoS metrics and measurement units. We have summarized requirements for such ontologies in [8]. 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.

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 consumers) may use a service offering and other security issues are outside the scope of WSOL.

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

Fig. 1. Parts of an Example WSOL Service Offering

WSOL constraints are defined using the <*constraint*> element, which is independent of particular types of constraint. The *type* attribute of the <*constraint*> element refers to the XML schema defining a particular type of constraint. We have defined XML schemas for the above-mentioned types of constraint. Using the XML Schema mechanisms, additional types of constraint can be defined. An example WSOL constraint, the QoS constraint '*QoScons2*', is shown in Figure 1. This QoS constraint

contains a comparison of a measured QoS metric '*ResponseTime*' and the constant '*0.3 second*'. It is evaluated for every operation of the '*buyStockService*' Web Service. The '*measuredBy*' attribute states that the QoS metric is measured by the same entity that evaluates this QoS constraint.

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. WSOL also supports checking operation message parts that are arrays (of any data type) using quantifiers ForAll and Exists. Arithmetic expressions can contain standard arithmetic operators (+, -, 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. In addition, 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.

2.3 Management Statements

A WSOL **statement** is any construct, other than a constraint, that states some important management information about the represented class of service. WSOL enables formal specification of several management statements:

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

In addition, WSOL has the general <*statement*> element for the specification of additional statements. It is analogous to the general <*constraint*> element.

Price statements specify the price that a consumer using the particular service offering has to pay for successful use of the Web Service. A service offering can contain subscription price statements and/or pay-per-use price statements. A **subscription price statement** specifies the monetary amount a consumer pays for using this service offering during some period. A **pay-per-use price statement** states the monetary amount a consumer pays for invoking particular operation. **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. A **management responsibility statement** specifies what entity has the responsibility for checking a particular constraint, a constraint group, or the complete service offering. A management entity can be the provider Web Service, the consumer, or an independent third party trusted by both the provider and the consumer [9]. Figure 1 shows an example WSOL management responsibility statement, '*MangResp1*'. In this example, the provider (supplier) Web Service evaluates the access

right '*AccRght1*', the consumer evaluates the pre-condition '*Precond3*', and a third party evaluates the QoS constraint '*QoScons2*'.

2.4 Reusability Constructs and the Syntax of Service Offerings

The WSOL reusability constructs enable easier specification of new service offerings, constraints, or management statements from existing ones. WSOL has a number of reusability constructs:

1. constraint groups (CGs),
2. inclusion statements,
3. constraint group templates (CGTs),
4. template instantiation statements, and
5. declarations of external operation calls.

A **constraint group (CG)** is a named set of constraints and/or statements. Arbitrary levels of nesting of CGs are allowed. When a new CG is defined and some of the contained constraints, statements, 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 **inclusion statements**. On the other hand, new constraints, statements, 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. 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. Definition of parameters is followed by definition or inclusion of constraints, statements, and nested CGs in the same way as for CGs. Constraints inside a CGT can contain expressions with CGT parameters. A CGT is instantiated when concrete values are supplied for all CGT parameters in a **template instantiation statement**. One CGT can be instantiated many times with different parameter values. The result of every such instantiation is a new CG. The concept of a CGT in WSOL is a very powerful specification mechanism. Many classes of service contain constraints with the same structure, but with different constant values. In our opinion, it is an even more important specification concept than the extension (single inheritance) of CGs, CGTs, and service offerings. A **declaration of an external operation call** is used to enable referencing results of the same external operation call in several related constraints.

Syntactically, a WSOL **service offering** is similar to a CG. It is a set of defined or included constraints, statements, and CGs (including instantiations of CGTs) that all refer to the same Web Service. WSOL supports extension (single inheritance) of service offerings, similarly to the extension of CGs and CGTs. However, service offerings must not be nested. Another syntactic difference is that every service offering has exactly one accounting party, specified in a special attribute, '*accountingParty*', of the <*serviceOffering*> element and not in management responsibility statements. The accounting party is a special management party responsible for keeping track of the use of the provider Web Service and management third parties, as well as what constraints were satisfied and what were not. While it is syntactically similar to a CG, a

service offering has special run-time characteristics. For example, consumers can choose and use service offerings, not CGs. In addition, dynamic relationships can be specified only for service offerings, not for CGs. Figure 1 shows an example WSOL service offering. Its parts were discussed in subsections 2.2 and 2.3.

2.5 Dynamic Relationships between Service Offerings (SODRs)

Dynamic relationships between service offerings (a.k.a. **service offerings dynamic relationships – SODRs**) are those that can change during run-time, e.g., after dynamic creation of a new class of service. For example, one **SODR** 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. On the other hand, **static relationships between service offerings** are those that are built into definitions of service offerings and do not change during run-time. Two important examples of such relationships are inheritance of service offerings and instantiation of the same CGT with different parameter values. Both static and dynamic relationships between service offerings are useful for easier selection and negotiation of service offerings. In addition, dynamic relationships are very useful for dynamic adaptation of Web Service compositions discussed in Section 4.

After researching several alternatives, we have decided to represent SODRs as triples <*SO1, S, SO2*> where:

1. *SO1* is the used service offering,
2. *S* is the set of constraints and/or CGs from SO1 that are not satisfied (a CG is not satisfied if at least one of its constraints or nested CGs is not satisfied); and
3. *SO2* is the appropriate replacement service offering.

These triples are specified in a special XML format outside definitions of service offerings (often in special files) to make their evolution independent from the evolution of other characteristics of a service offering. However, the XML format for the specification of these triples is an integral part of the WSOL language. It is built upon the WSOL concepts of a constraint and a CG, as well as WSOL solutions for naming of constraints, CGs, and service offerings. We are also considering an extension of this XML format so that *S* could also contain arbitrary WSOL Boolean expressions. This would enable specification of complex relationships between the unsatisfied constraints and CGs, as well as using external operation calls.

3 Applications of WSOL for Web Service Management

In addition to the WSOL language, we are developing the corresponding management infrastructure and management algorithms. Appropriate specification of management information, such as WSOL constraints and management statements, is the key for successful management activities. Functional constraints can help in determining whether a Web Service behaves correctly. Consequently, they are useful in fault man-

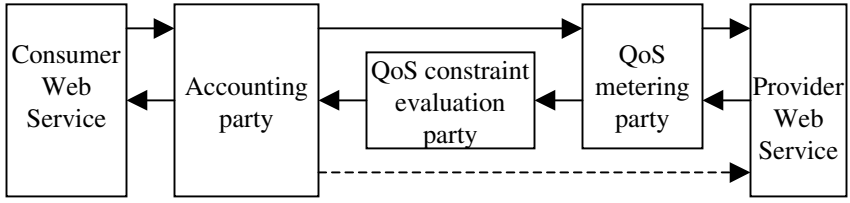

Fig. 2. An example configuration of management third parties as SOAP intermediaries

agement. Formal and precise specification of QoS constraints is the basis for monitoring and metering of 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. Consequently, formal and precise specification of QoS constraints is particularly useful in performance management. Access rights limit access to operations and ports of a Web Service. While WSOL access rights are used primarily for service differentiation, they could also be a part of a comprehensive security management solution for Web Services. Statements about prices and monetary penalties are invaluable in accounting management. WSOL service offerings are precise and complete enough to serve as simple contracts or SLAs for Web Service monitoring, metering, control, accounting, and billing. Further, dynamic (i.e., run-time) manipulation of service offerings, discussed in the next section, is a useful tool for both Web Service Management (WSM) and Web Service Composition Management (WSCM).

Figure 2 shows an example configuration of management third parties as **SOAP intermediaries**. In this example, only QoS constraints are evaluated and distinction is made between the accounting party, the QoS metering party, and the QoS constraint evaluation party. When a consumer submits a request for executing a provider's operation, the management third parties are organized as SOAP intermediaries for the request, as well as the eventual response message. The request from the consumer to the provider first goes through the accounting party, which logs that the request has been made. The request is then forwarded to the QoS metering party, which performs necessary activities. For example, for metering response time the QoS metering party logs the time that will be considered as the beginning time of the operation invocation. Next, the request message is forwarded to the provider Web Service, which performs the requested operation and sends the response message. The response message first goes through the QoS metering party. In the response time measurement example, the QoS metering party logs the time that will be considered as the ending time of the operation invocation and subtracts from it the logged beginning time. The information about the measured QoS metrics can be transferred to the QoS constraint evaluation party along with the original response message from the provider. One way to piggyback this information is to use SOAP headers. The QoS constraint evaluation party receives the response message and the information about the measured QoS metrics and evaluates appropriate constraints. It forwards to the accounting party the response message together with the information whether the evaluated constraints were satisfied

or not and appropriate details if some constraints were violated. The accounting party logs the received management information, calculates prices and/or penalties to be paid, and forwards the response message to the consumer. If some QoS constraints were not satisfied, the accounting party notifies the provider with appropriate details. This can help the provider to adapt its behavior to meet guarantees for future operation requests.

Note that the discussed example is only one possible way to use WSOL for WSM. There are other management scenarios that can be accommodated with WSOL. For example, some QoS metrics, such as availability, can be measured using probing instead of message interception. WSOL supports this by 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. Further, to reduce overhead, some QoS constraints can be evaluated **periodically**, either for randomly selected operation invocations (on average: 1 in n) or at particular date/time instances. Also note that often there is no need to have a large number of specialized third parties. The overhead can in the above example is reduced when instead of three separate parties for accounting, QoS metering, and evaluation of QoS constraints, only one third party with all these functions is used. It is also possible to put all these functions into the provider Web Service, reducing the overhead further. However, in such a case, the consumer has to trust the provider.

We have designed a WSOL management infrastructure supporting evaluation of WSOL constraints, metering and calculation of used QoS metrics, and accounting of executed operations and evaluated WSOL constraints. The design of this infrastructure and, particularly, its proof-of-concept prototype implementation are based on extensions of Apache Axis (Apache eXtensible Interaction System) [10], a popular open-source SOAP engine. A SOAP engine is an application that receives, processes, and sends SOAP messages. Axis has several good features that we found crucial for our WSOL management infrastructure. Here we emphasize the two most important. First, Axis has a modular, flexible, and extensible architecture based on configurable chains of pluggable SOAP message processing components, called handlers. Such architecture enables implementing our WSOL management infrastructure as a set of additional handlers and handler chains plugged into Axis easily. Second, Axis defines a SOAP message processing node that can be used for provider Web Services, consumer Web Services, and SOAP intermediaries. Consequently, it can be used for all WSOL management parties.

The design and prototype implementation of our WSOL management infrastructure will be discussed in detail in a forthcoming publication. Here we outline the main concepts. An Axis handler [10] can process the input, the output, or the fault SOAP message. It can alter the SOAP message—e.g., add/remove headers—or perform some other message processing—e.g., measurement of QoS metrics. Consequently, we implement metering and calculation of QoS metrics, evaluation of WSOL constraints, and accounting inside specialized Axis handlers that we have developed. We have designed these handlers so that a WSOL compiler can generate them automatically from WSOL files. However, since our prototype WSOL compiler is not yet fully im-

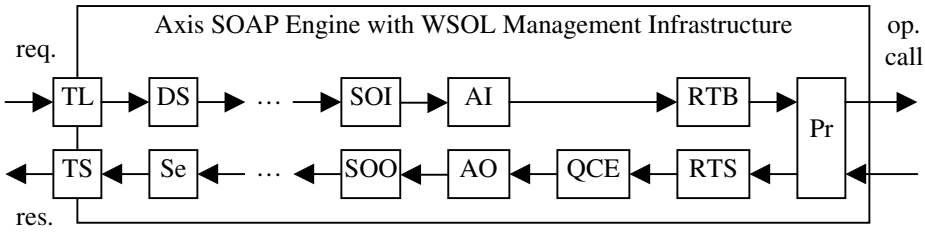

Fig. 3. An Example Configuration of Handlers inside the Provider-side Axis SOAP Engine with WSOL Management Infrastructure

plemented, we have manually implemented some of these handlers in our prototype WSOL management infrastructure.

The crucial Axis data structure passed between handlers is the message context [10]. It contains information about the request message, the response message, and a bag of properties. The message context properties determine how handlers process the message and can be modified by handlers. We transport WSOL information between WSOL-specific Axis handlers in special properties of the message context. For WSOL information transported in SOAP headers between management parties, translation between SOAP headers and Axis message context properties is provided. In addition to the special message context properties, we have also developed data structures for storing descriptions of WSOL constructs and the actual measured or computed data. Examples of stored data are values of QoS metrics and the information whether a constraint is satisfied or not. We use this data for accounting and for determining whether dynamic adaptation, discussed in the next section, is needed.

An Axis chain [10] is an ordered, pipelined collection of handlers. There are tree types of chain in Axis. A transport chain performs processing related to the transport of SOAP messages. A global chain performs other processing applicable to all Web Services. A service chain performs processing characteristic for a particular Web Service. Since processing of WSOL constraints and management statements differs between service offerings of the same Web Service, we could have introduced the fourth type of chain – a 'service offering chain'. For simplicity, we have put all WSOL-related handlers into service chains, but added some program code deciding what handlers are executed in particular cases.

In Figure 3, we have shown an example configuration of handlers inside the provider-side Axis SOAP engine extended with WSOL management infrastructure. The example is analogous to the example in Figure 2, but in Figure 3 all shown modules are parts of the provider Web Service. In other words, the provider Web Service measures response time, evaluates a QoS constraint limiting response time, and performs accounting. The Transport Listener (TL) component of Axis receives the request SOAP message and passes it to the standard Axis handler called Deserializer (DS), which creates a message context instance used by other handlers. After Deserializer, several other standard Axis handlers are executed, not shown in Figure 3. The first WSOL-specific handler in the input flow is Service Offering Input (SOI), which

performs processing characteristic for all service offerings. The next WSOL-specific handler, Accounting Input (AI), records the request message. Then, the Response Time Begin (RTB) handler stores into message context the start time for measuring response time. After this handler, the standard Axis handler, Provider (Pr), is executed. It dispatches the call to the Java object implementing the requested operation of the Web Service. This Java object returns its results back to the Provider handler. After Provider, the WSOL-specific handler Response Time Stop (RTS) stores into message context the stop time for measuring response time, as well as the difference between this stop time and the start time stored by RTB. The QoS constraint limiting response time is evaluated in the WSOL-specific handler QoS Constraint Evaluation (QCE). This handler stores its results into the message context. The Accounting Output (AO) handler uses the information from the message context to calculate prices and eventual penalties to be paid. The last WSOL-specific handler in the output flow is Service Offering Output (SOO), which performs characteristic for all service offerings. After this handler, some standard Axis handlers, not shown in Figure 3, are executed. The last of these handlers is Serializer (Se), which packs information from the message context instance into SOAP. The Transport Sender (TS) component of Axis sends the response SOAP message to the consumer. The information about the measured response time, the evaluated constraint, and eventual prices or penalties to be paid is in the SOAP header.

While the evaluation of periodic constraints differs from the example illustrated in Figure 3, it is also supported by our WSOL management infrastructure. Timer, a special active object in our infrastructure, initiates evaluation of periodic constraints and measurement or calculation of periodic QoS metrics. The processing is done in one or more modules, similar to Axis handlers. The results of such evaluation, measurement, or calculation can be stored locally for future processing. They can also be reported to other management parties in a special notification SOAP message.

4 Applications of WSOL for Web Service Composition Management

We are also investigating management and dynamic adaptation of Web Service compositions without breaking an existing relationship between a provider Web service and its consumer. To achieve this goal we are exploring management and dynamic adaptation mechanisms that are based on the manipulation of service offerings in WSOL. Our dynamic adaptation mechanisms include switching between service offerings, deactivation/reactivation of existing service offerings, and creation of new service offerings. These mechanisms can be used between operation invocations that are part of the same transactions or session. The crucial language support for these mechanisms is the specification of dynamic relationships between service offerings (SODRs), discussed in Subsection 2.6.

Dynamic switching between service offerings can be initiated by a provider Web Service or its consumer. The consumer can initiate it to dynamically adapt the service

it receives without finding another Web service. The provider Web Services can initiate it to gracefully upgrade or degrade its service and/or 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 provider Web Service has to be sought. The deactivated service offering may 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 on demand of important consumers. It then becomes a substitute for negotiation of a custom-made contract or SLA between Web Services. Note that dynamic creation of new service offerings can be non-trivial and incur non-negligible overhead. Therefore, we use it only in exceptional circumstances.

Deactivation, reactivation, and dynamic creation of service offerings can be followed by appropriate deactivation, reactivation, or creation of SODRs.

Compared to finding alternative Web Services (i.e., rebinding 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. These capabilities are relatively simple and incur relatively low overhead, while providing additional flexibility. (Dynamic composition of new service offerings can be an exception for the previous statement.) 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 not a complete replacement for, finding alternative Web Services. The first step in dynamic adaptation of a Web Service composition should be to try to find a replacement service offering from the same Web Service. If this is not possible, only the second step should be to try to find a replacement Web Service and perform re-composition. In some cases, the used provider Web Service can provision a temporary replacement service offering while the consumer searches for another, more appropriate, Web Service.

We have integrated the support for these dynamic adaptation mechanisms into our WSOL management infrastructure, discussed in the previous section. Hereafter, we denote the parts of our WSOL management infrastructure that support dynamic adaptation mechanisms as ‘WSCM modules’. WSCM modules are usually parts of the provider-side WSOL management infrastructure, but some can also be used in consumers and third parties. While our WSCM modules are not based on the functionality provided in Apache Axis, they use the same data structures as our extensions for ac-

counting of executed operations and evaluated WSOL constraints. These data structures store information such as which constraints or CGs were satisfied or unsatisfied. This information is used in determining whether the provider Web Service should switch a consumer to a more appropriate service offering and maybe even deactivate a service offering that can no longer be fulfilled. In addition, data structures in our WSCM modules store descriptions of SODRs and supplementary information, such as what service offerings are active or deactivated and what consumers use particular service offerings. This information is used in determining what service offering is the best candidate for switching to and in handling of affected consumers.

We have designed algorithms for autonomous (i.e., without external intervention) provider-side switching, deactivation, and reactivation of service offerings, and deactivation and reactivation of SODRs. These algorithms are executed in a provider-side WSOL management infrastructure when certain conditions are met. They use the data structures mentioned above. Currently, we have only rudimentary support for dynamic creation of service offerings and SODRs.

On the other hand, consumers can also initiate switching of service offerings and, in theory, creation of service offerings. Further, it is useful to enable that selected external WSCM entities (human administrators or specialized software) can have access to our dynamic adaptation mechanisms. Therefore, we expose these mechanisms as operations of the special 'service offering management (SOM)' port of every Web Service. While we suggest that every WSOL-enabled Web Service provide the SOM port, some or all Web Services of the same vendor (e.g., Web Services using the same SOAP engine instance) can share the actual implementation of its operations. The SOM port contains operations for switching, activation, deactivation, and creation of service offerings and for activation, deactivation, and creation of SODRs. However, it also contains some other operations that are crucial for WSOL-enabled Web Services. One example is the operation that returns a WSOL file with descriptions of all service offering available to a particular consumer. A similar operation returns a WSOL file with descriptions of all relevant SODRs. Another example is the input-only operation that a third-party accounting party uses to inform the provider Web Service about the values of metered or calculated QoS metrics, evaluated WSOL constraints, and their monetary consequences. Note again that only selected external entities are allowed access to the majority of operations in the SOM port. We will discuss the SOM port in detail in a forthcoming publication.

5 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., [11]). At the beginning of our research, there was no relevant work of this kind for Web Services. In parallel with our research, several related works emerged.

The most important related works to WSOL are the two recent languages for formal XML-based specification of custom-made SLAs for Web Service: the **Web Service Level Agreements (WSLA)** [4, 9] from IBM and the HP work on the formal

specification of Web Service SLAs [3, 12]. The latter work seems to be part of the HP's **Web Service Management Language (WSML)**. SLAs in these two languages contain QoS constraints and management information. 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. Further, these languages specify more detail for QoS constraints than WSOL and specify custom-made SLAs, not classes of service. In these aspects, they are more powerful than WSOL. It seems that this results in higher run-time overhead than the overhead of the simpler WSOL. These languages are accompanied by appropriate management infrastructures [9, 12]. These infrastructures are more powerful, but also more complex, than the infrastructure we are developing for WSOL. Both WSLA and WSML have some support for templates, but only at the level of an SLA, not its parts. They do not have support for inheritance and the other reusability constructs present in WSOL. 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 very good languages for their domain and purpose, they do not address all the issues that WSOL does.

Another recent related work is **WS-Policy** [13] – a general framework for the specification of policies for Web Services. A policy can be any property of a Web Service or its parts, so it corresponds to WSOL concepts of a constraint and a management statement. WS-Policy is only a general framework, while the details of the specification of particular categories of policies will be defined in specialized languages. The only such specialized language currently developed is WS-SecurityPolicy. WS-PolicyAssertions can be used for the formal specification of functional constraints, but the contained expressions can be specified in any language. It is not clear whether and when some specialized languages for the specification of QoS policies, prices/penalties, and other management issues will be developed. Another set of issues is where, when, and how are WS-Policy policies monitored and evaluated. WS-Policy has a number of good features, such as flexibility, extensibility, and reusability. However, some of the advantages of WSOL are the explicit support for management applications, built-in support for various constraints and management statements, unified representation of expressions, wider range of reusability constructs, and specification of classes of service and relationships between them.

Further, the **DAML-S (DAML-Services)** [14] community works on semantic descriptions of Web Services, including specification of some functional and some QoS constraints. However, constraints in DAML-S are not currently specified in a precise, formal, and detailed notation, as in WSOL. They are only placeholders for the future description of rules. In addition, they are specified for a more comprehensive service description, not for the actual control and management. These are two major differences between DAML-S and WSOL. While DAML-S has the concept of a service profile, there is no concept of a class of service and no specification of dynamic relationships. Consequently, we find that WSOL has a clear advantage in Web Service Management and Web Service Composition Management.

Apart from these recent works that partially address similar issues to WSOL, there are several other resent works that recognize the importance of the formal specification of various constraints, SLAs, and contracts for Web Services and special types of Web Service (such as Grid Services and Semantic Web enabled Web Services). In [2], we concluded that the unique characteristics of WSOL, compared to the recent related works, are its **expressive power**, **mechanisms for reduction of run-time overhead**, and **support for management applications**.

6 Conclusions and Future Work

WSOL supports management applications with:

1. the formal and unambiguous specification of various constraints, statements about prices and monetary penalties, and management responsibility statements;
2. the possibility to specify management third parties in management responsibility statements, in the '*measuredBy*' attribute of QoS metrics, and in external operation calls to management third parties that act as probes;
3. the explicit specification of accounting parties (specific management parties); and
4. the built-in format for the specification of dynamic relationships between service offerings (SODRs).

WSOL can be used for both Web Service Management (WSM) and Web Service Composition Management (WSCM). WSOL service offerings can serve as simple contracts or SLAs for Web Service monitoring, metering, control, accounting, and billing. They can be metered and evaluated by the provider Web Service, the consumer, and/or one or more mutually trusted third parties (SOAP intermediaries or probes). Manipulation of service offerings can be used as a lightweight complement and addition to rebinding of Web Service compositions.

Several recent related works—WSLA, WSML, and WS-Policy—address issues that partially overlap with WSOL. In some aspects, they are more powerful than WSOL. However, WSOL also has its advantages, such as classes of service, SODRs, reusability constructs, and relative simplicity and lightweightness. An integration of WSLA, WSML, WS-Policy, and WSOL would benefit WSM and WSCM.

While WSOL can be improved in several ways, we consider the language relatively complete and stable. We direct our research efforts mainly towards the further development of the WSOL management infrastructure, its prototype implementation, and the research of WSOL applications in WSCM. We have set up an experimental computer network on which we run in parallel multiple Web Service compositions. In this environment, we are experimenting with WSCM both with and without WSOL. For example, in some experiments we compare switching of service offerings with replacing a deactivated Web Service with its equivalent found in a local UDDI directory. We want to gain more precise understanding of the management applicability and boundaries of WSOL. We also trying to uncover and clarify the most important issues for the future management-related research using WSOL. We will describe the results of such experiments in a forthcoming publication, along with further details about the WSOL management infrastructure and its prototype.

We have left some important areas for future work. One of them is discovery and selection of WSOL service offerings. We have to address the integration of WSOL into UDDI to enable discovery of WSOL service offerings. Further, we believe that static and dynamic relationships between service offerings can be very useful for comparing similar service offerings in the process of negotiation and selection. More research in this area is needed. Another important area is security. WSOL could be used with security technologies for Web Services. For example, different keys could be used for encryption of the message body and various QoS measurements and constraint evaluation results, so that only relevant management parties would see them.

References

1. World Wide Web Consortium (W3C): Web Services Description Requirements. W3C Working Draft 28 October 2002. On-line at: http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028/ (2002)
2. Tosic, V., Pagurek, B., Patel, K.: WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services. Res. Rep. OCIECE-02-06. Ottawa-Carleton Institute for Electrical and Computer Engineering. Nov. 15, 2002. On-line at: http://www.sce.carleton.ca/netmanage/papers/TosicEtAlResRepNov2002.pdf (2002)
3. Sahai, A., Durante, A., Machiraju, V.: Towards Automated SLA Management for Web Services. Research Report HPL-2001-310 (R.1), Hewlett-Packard (HP) Laboratories Palo Alto. July 26, 2002. On-line at: http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf (2002)
4. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, Vol. 11, No 1 (Mar. 2003) Plenum Publishing (2003)
5. Tosic, V., Pagurek, B., Esfandiari, B., Patel, K., Ma, W.: Web Service Offerings Language (WSOL) and Web Service Composition Management (WSCM). In Proc. of the OOWS'02 (Object-Oriented Web Services) workshop at OOPSLA 2002 (Seattle, USA, Nov. 2002) On-line at: http://www.research.ibm.com/people/b/bth/OOWS2002/tosic.zip (2002)
6. 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)
7. 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 at CAiSE'02 (Toronto, Canada, May 2002). Lecture Notes in Computer Science (LNCS), No. 2512. Springer-Verlag (2002) 57–67
8. 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
9. Dan, A., Franck, R., Keller, A., King, R., Ludwig, H.: Web Service Level Agreement (WSLA) Language Specification. In Documentation for the Web Services Toolkit, Version 3.2.1. Aug. 9, 2002. International Business Machines Corporation (IBM) (2002)

10. The Axis Development Team: Axis Architecture Guide, Version 1.0. Apache Axis WWW page. On-line at: http://cvs.apache.org/viewcvs.cgi/~checkout~/xml-axis/java/docs/architecture-guide.html (2003)
11. Beugnard, A., Jezequel, J.-M., Plouzeau, N., Watkins, D.: Making Components Contract Aware. Computer, Vol. 32, No. 7 (July 1999) IEEE (1999) 38–45
12. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA Monitoring for Web Services. In Proc. of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2002 (Montreal, Canada, Oct. 2002). Lecture Notes in Computer Science (LNCS), No. 2506. Springer-Verlag (2002) 28–41
13. 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)
14. 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)