On Extending WS-Policy with Specification of XML Web Service Semantics

VLADIMIR TOSIC, ABDELKARIM ERRADI, PIYUSH MAHESHWARI
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052
AUSTRALIA
http://masc.web.cse.unsw.edu.au/

Abstract: - Several languages for specification of various aspects of semantics, such as quality of service (QoS), of XML Web services have appeared recently. However, none of them is widely accepted by industry. On the other hand, WS-Policy (a general framework for the specification of policies for Web services) has strong industry support, but currently lacks detailed specification of ontological meaning, QoS, and other important aspects of semantics. We propose extending WS-Policy with comprehensive specification of semantics of Web services. First, we discuss specification of semantics with policies. Then, we state principles for describing semantics of Web services. Finally, we discuss strengths and weaknesses of WS-Policy and our proposal for extensions in the new WS-Policy4MASC language. These extensions enable that a Web service composition can be comprehensively described with WSDL, WSBPEL, and WS-Policy4MASC, without the need for OWL-S, WSLA, WSOL, or WS-Agreement.

Key-Words: - Web services, semantics, policy, WS-Policy, Web service management, quality of service, adaptability.

1 Introduction

An XML (Extensible Markup Language) Web service is a software application identified by a URI (Uniform Resource Identifier), described in XML-based languages, and supporting direct interactions with other software using XML-based messages over Internet-based protocols. The three main Web Service technologies are the SOAP protocol for XML messaging, the Web Service Description Language (WSDL) for service interface description, and the Universal Description, Discovery, and Integration (UDDI) registry for service publication and discovery. The Web Services Business Process Execution Language (WSBPEL) for describing Web service compositions is also increasingly popular.

While there has been a lot of recent progress regarding Web Services, a number of issues have not yet been studied completely. One of these issues is the specification of semantics of Web services. Hereafter, by ‘semantics’ we mean formal and precise description of the meaning of terms related to Web services. For example, these terms can be Web service names, endpoints, interfaces, operations, messages, input output parts, quality of service metrics, or used measurement units. As advocated by the Semantic Web community, semantics of Web services is needed to successfully perform automatic Web service discovery, selection, composition, invocation, and interoperation [1].

We will use the following example to illustrate semantics of Web services, policies, and our suggestions. Assume that there is a current weather report Web service with one operation: Integer weatherTemperature(String postalCode). In other words, the Web service operation ‘weatherTemperature’ receives one input parameter of the String data type and returns one output parameter of the Integer data type. A semantic description of this operation would somehow associate the name of this operation with the notion of current weather reports, the input parameter with the notion of postal codes, and the output parameter with the notion of weather temperature. More precise semantic descriptions are also possible. For example, it is possible to specify that the operation reports weather in Canada and uses Celsius degrees, so that the input parameter must be a valid Canadian postal code, while the output parameter represents the temperature in Celsius degrees and that the temperature was measured not more that 1 hour ago (so it is relatively current).

This paper proposes specification of semantics of Web services using policies and, in particularly, developing extensions to the general, industry-developed Web Services Policy Framework (WS-Policy) [2]. We have been integrating this proposal into our new XML language WS-Policy4MASC, which is an extension of WS-Policy. While we suggest a significantly different approach from the approaches previously advocated in the Semantic Web literature, it much better addresses Web service management requirements.

This section provided a general introduction to the semantics of Web services. The following section defines the term ‘policy’ and illustrates how policies can be used to represent semantics of Web services. Section 3 outlines the principles that we propose for specification of semantics of Web services. They are based on past publications and our extensive experience in developing middleware for Web services, particularly for Web service management. Section 4 discusses strengths and weaknesses of WS-Policy and outlines the proposed WS-Policy extensions. Section 5 presents how we have implemented most of these proposed extensions in our new WS-Policy4MASC language. The final section summarizes conclusions and future work items.

2 Using Policies to Represent Semantics

In the area of management of networks and distributed systems, policy-driven management [3] has caught considerable attention during the last several years. There is an ongoing discussion on the precise definition of the term policy, but we will hereafter use the term 'policy' to denote high-level, implementation-independent, operation and management goals and/or rules expressed in a human-readable form. Policies can be seen as decision-making guidelines specifying the rules governing choices in the behavior of a system. A policy-driven management system refines these high-level goals and rules into many low-level, implementation-specific, actions controlling operation and management of particular system elements. For example, a policy could be used to: (1) ensure compliance, (2) configure behavior, or (3) achieve adaptability. Several classifications of policies exist. We find the classification from [4] particularly useful. It differentiates action policies (describing actions to be taken in a particular state), goal policies (describing desired states of the system), and utility function policies (defining value of each possible state).

Policies can be used to represent semantics (i.e., meaning) if they are interpreted as rules about meaning. We will illustrate this on the example introduced in the previous section. Several approaches are possible. One way is to use two action policies. The first specifies that whenever the input parameter is not a valid Canadian postal code, then an error message (caused by client misbehavior) will be reported. The second specifies that whenever the output parameter is not a valid current Canadian weather temperature in Celsius degrees, then an error message (caused by service misbehavior) will be reported. Another way to describe the same semantics is to use two corresponding goal policies – one describing the desired state of the operation's input parameter, the other describing the desired state of the operation's output parameter.

In both approaches, the two policies describe operation's pre-conditions and post-conditions. To be able to use these policies for run-time monitoring and management activities, these policies (or additional related policies) should also describe how these conditions (requirements/guarantees) are checked during run-time. For example, the pre-condition policy can specify that for checking the input parameter, it invokes the postal code verification Web service by Canada Post for every submitted input parameter. The post-condition policy can specify that the verification of the output parameter involves limiting its values (e.g., between –70C and +50C) and using Environment Canada Web services to periodically and randomly (e.g., on average once in 1000 invocations) check that the values provided by the current weather report Web service are really current weather temperatures in Canada and not some made-up numbers within the above-mentioned limits or historical numbers from the past.

Note that simply annotating that the 'weather' operation is associated with an ontological definition of the 'current Canadian weather report for the given postal code', its input parameter is associated with an ontological definition of the 'Canadian postal code', and its output parameter is associated with an ontological definition of the 'current temperature' can also be done with goal policies. These policies can also require that value provided for the input parameter is annotated with an association to the same ontological definition of the 'Canadian postal code', and the value provided for the output parameter is associated with the same ontological definition of the 'current temperature'. However, in our opinion such declarative descriptions (although championed by some members of the Semantic Web community) are not as useful in practice as the previously mentioned management-oriented policies. They can be used for formal reasoning about Web services (e.g., in Web service selection), but they miss information crucial for run-time checks of provided values. For example, they do not specify when and by which party the necessary checks are performed.

3 Principles for Specification of Web Service Semantics

The authors in [5] studied comprehensive contractual description of Web services and suggested several principles for the work on a unifying framework addressing various functional, quality, and infrastructure contracts for Web services. We identify here several principles for specification of semantics of Web services, based on [5], other past publications, and our extensive experience [6, 7, 8] in developing middleware for Web services and Web service management:

1. Specification of semantics should be optional for Web services. Although semantics is useful, not everybody will want to use it, e.g., due to the overhead of corresponding specification, reasoning, and management activities. Consequently, semantic descriptions should be a layer above WSDL.

2. There should be a unified (hopefully: standardized) format for representing expressions. Expressions can be used to describe various relationships between terms. Their unification enables easier reasoning, significantly reduces the run-time overhead, and makes selection and management of Web services easier.

3. The number of languages for describing Web services should be kept small. This is because there is less run-time overhead in supporting one language than a group of languages, even if they are compatible. Further, this reduces redundancies and potential incompatibilities. The currently dominant approach within the Semantic Web community is to use OWL-S (OWL-Services) [1], which is based on OWL (Web Ontology Language), which is, in turn, based on RDF (Resource Definition Framework). In spite of past efforts, there are some redundancies and potential incompatibilities between WSDL and OWL-S. OWL-S is now advocated as a “complement, not competitor” to WSDL and a similar future relationship with WSBPEL might be developed. There are also several additional and complementary Web service languages outside the Semantic Web community, such as WS-Policy [4]. This means that a Web service provider has to support a large number of languages, which all have different tools and introduce significant run-time overhead.

4. Reuse and extension of the widely accepted Web Service languages. There are already many languages for Web Services. In our opinion, development of new languages or popularization of less-known languages will probably not be as effective as reuse and extension of languages in which companies made investments. WSDL is the only Web Service language that is widely accepted, so it has to be used. Further, it seems that the acceptance of WSBPEL is gaining momentum. Further, a number of languages have recently appeared for specification of requirements and capabilities for Web services, including WS-Policy, WSLA (Web Service Level Agreement) [10], Web Service Offerings Language (WSOL) [6], Web Services Agreement Specification (WS-Agreement) [11], and OWL-S [1]. However, it seems that the industrial support is strongest for WS-Policy.

5. The specification of semantics must support monitoring of functional and QoS characteristics of Web service executions. This requires that the language must enable specification of which QoS metrics (e.g., response time) are monitored or calculated, when/where/how this is done, and how the monitored/calculated values are exchanged between management parties. Further, this requires specification of functional and QoS conditions (requirements and guarantees) that are evaluated, when/where/how this evaluation is done, what party is responsible for satisfaction of these conditions, and how the results are exchanged between management parties.

6. The specification of semantics must support control (particularly: adaptation) of execution of Web services and Web service compositions.

7. The specification of semantics must support Web service management and ontological reasoning.

There are numerous additional detailed requirements for a corresponding specification language. They can be found in [9].

4 WS-Policy – The Current Status and the Proposed Extensions

The Web Services Policy Framework (WS-Policy) [4], an industrial specification standardized by the World Wide Web Consortium (W3C). It defines an extensible container to hold domain-specific policy assertions. It also provides a general framework for attaching attributes/metadata to services and for placing range of interaction constraints with respect to various QoS aspects, such as security (e.g., encryption type, authentication mode) or reliable messaging. It is intended as a complement to WSDL and WSBPEL. In the context of Web services, policies can be defined and communicated either statically or dynamically. Static policies can be attached to the service contract (e.g., WSDL) while dynamic policies are created and communicated to service consumers during interactions with the service.

In the WS-Policy model, a policy is defined as a collection of policy alternatives, each of which is a collection of policy assertions. A policy assertion binds a variable to one or more possible values using a policy vocabulary defined by domain-specific languages, such as WS-SecurityPolicy. WS-PolicyAttachment defines a generic mechanism to associate a policy with subjects to which the policy applies, such as WSDL elements or Web service registry information. Various policy subjects are possible, such as service, endpoint, operation, message, or message part. A policy scope is a set of policy subjects to which a policy may apply.

WS-Policy has a number of good features. For example, it is flexible and extensible – policies can be specified both inside and outside WSDL files. Further, it has some reusability mechanisms, such as inclusion and grouping of policies. Nevertheless, it must be noted that 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 are WS-SecurityPolicy and WS-ReliableMessaging. 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 quality of service (QoS) policies, prices/penalties, and other management information will be developed. This is a serious limitation. Some unification and standardization of common elements, such as expressions, of various WS-Policy languages would reduce the overhead of supporting this framework. WS-Policy also does not have concepts of a contract, such as a service level agreement (SLA) or class of service. Consequently, we advocate extending WS-Policy with specification of these concepts. Further, WS-Policy does not detail where, when, and how are policies monitored and evaluated. Since many policies have to be monitored and controlled during run-time, WS-Policy needs better support for management applications, including explicit specification of such management information.

We propose that ontological meaning for monitored data items (e.g., message parts and QoS metrics) is specified in the extended WS-Policy with a simple construct OntologicalMeaning that has 2 attributes:
(1) OntologicalDefinition – a qualified XML name containing namespace of the used ontology and name of the ontological concept within this ontology; and
(2) OntologyLanguage – the URI of the language in which the referenced ontology is defined.
The actual definitions of ontological concepts would be in external, reusable and extensible, ontologies. In the current practice, ontologies are defined in several languages, such OWL, RDF, RDF Schema, and XML Schema. By allowing the use ontologies in different languages, the interoperability suffers and the requirement of using minimal number of Web service languages is not satisfied. While some simple ontology format (e.g., as defined for WSOL) can be used by default, a simple format is not enough for supporting ontological reasoning. On the other hand, this approach has a good characteristic that if a Web service does not understand the ontology, it at least knows the name of the ontological concept and can perform simple syntax matching of ontological meanings.

For the specification of functional pre- and post-conditions, prices/penalties, QoS constraints, and management statements, we suggest that WSOL concepts of a constraint and a statement are re-defined as policy assertions in a specialized WS-Policy extension language. WSOL has a standardized expression schema (defined separately from the rest of the language, to achieve reusability) and it could be reused in the extended WS-Policy. The new concept of a contract could be defined in the extended WS-Policy as a collection of policies, policy attachments, and additional information (e.g., contract parties, validity, etc.). The WSOL concept of a service offering could be used as a role model for this definition. Additional WS-Policy extensions with policies related to Web service compositions, such as recovery policies, can be taken from [7].

5 Implementation of the Proposed Extensions in WS-Policy4MASC

We have been implementing the above suggestions in our WS-Policy4MASC extension of WS-Policy. Its goal is to enable specification of policies for monitoring of functional and QoS aspects (such as performance and reliability) and different types of adaptation for Web services and their compositions, in a way that can be used for automatic configuration of our MASC (Manageable and Adaptable Service Compositions) middleware presented in [8]. To be able to perform policy-driven management of Web services and their compositions in the MASC middleware, we needed a machine processeable and precise format for declarative specification of various types of policies. We have chosen WS-Policy as the basis for our policy specification in WS-Policy4MASC, but added original detailed constructs useful for QoS monitoring and dynamic adaptation. Note that WS-Policy4MASC is also compatible with other Web services standards such as WSDL and WSBPEL, as well as Microsoft .NET 3.0 technologies, such as the Extensible Application Markup Language (XAML).

WS-Policy4MASC will be described in detail and illustrated in a forthcoming paper, but this paper summarizes its main characteristics. Our language extends WS-Policy by defining XML schemas with new types of policy assertions. Goal policy assertions specify requirements and guarantees to be met in desired normal operation (e.g., response time of a particular activity has to be less than 1 second). They guide monitoring activities in MASC. Action policy assertions specify actions to be taken if certain conditions are met (e.g., some guarantees were not satisfied). For example, these actions can be removal, addition, replacement, skipping, or retrying of a sub-process (or individual activity) or process termination. They guide adaptation and other control actions in MASC. Utility policy assertions specify monetary values assigned to particular situations (e.g., execution of some action). They can be used by MASC for billing and for selection between alternative action policy assertions. Meta-policy assertions can be used to specify which action policy assertions are alternative and which conflict resolution strategy (e.g., minimization of costs) should be used. In addition to these 4 new types of policy assertions, WS-Policy4MASC enables specification of additional information that is necessary for run-time policy-driven management (monitoring, control). For example, this includes information about conditions when policy assertions are evaluated/executed, parties performing this evaluation/execution, a party responsible for meeting a goal policy assertion, monitored data items, states, state transitions, schedules, events, and various expressions (e.g., Boolean and arithmetic with units).

The described construct OntologicalMeaning is also defined in the XML schema for WS-Policy4MASC, but it is currently not yet used by the MASC middleware. The default ontology language is the simple ontology schema that was first defined for WSOL.

WS-Policy4MASC satisfies, at least to some extent, all principles for specification of Web service semantics that we have listed in Section 3. First, WS-Policy4MASC is an optional language, in a layer additional to and compatible with WSDL and WSBPEL. Second, it defines its own unified format for specification of expressions, which is an improvement of the expression format defined for WSOL. Third, it is possible to comprehensively describe Web services and Web service compositions using only WSDL, WSBPEL, and WS-Policy4MASC, while the need for the other languages is eliminated (except for definition of optional ontological meaning). Fourth, it extends WS-Policy, which is already used in practice for purposes compatible to the purpose of WS-Policy4MASC. Fifth, it provides detailed specification of Web service monitoring activities, primarily through goal policy assertions. Sixth, it enables detailed specification of Web service control activities, particularly adaptation of Web service compositions. Action policy assertions and, to some extent, utility policy assertions and meta-policy assertions are the key constructs in this regard. Seventh, the WS-Policy4MASC construct for ontological meaning provides some support for ontological reasoning, but improvements are possible in this area.

A partial example of WS-Policy4MASC constructs is shown in Figure 1. It is from our series of scenarios related to a stock trading case study [8]. The <When> element specifies that when a process (e.g., a Web service composition) is in the Executing state and the PortfolioValueReceived event occurs the Boolean expression called IntlPortfolio is evaluated to check whether the portfolio contains only amounts in local currency or currency conversion is needed. The < ActionPolicyAssertion> element specifies that if all conditions in the above <When> element are specified, then the process (e.g., Web service composition) orchestrator is the management party responsible for executing a set of process addition actions (these actions are not shown in Figure 1 for brevity).

Within the MASC middleware, WS-Policy4MASC policy assertions are stored in a policy repository, which is a collection of instances of policy classes. The policy classes are generated automatically from the WS-Policy4MASC XML schema, using an XML-schema-to-classes generator (in our .NET 3.0 and C#-based prototype of MASC, we used the XSD tool from .NET 3.0). When MASC starts, our MASCPolicyParser within it imports WS-Policy4MASC files, creates instances of corresponding policy classes, and stores these instances in the policy repository. Using this policy information, monitoring modules in MASC configure themselves to monitor relevant events. When such an event happens, it triggers evaluation of goal policies, execution of action policies, calculation of utilities, and/or other effects (e.g., a state transition). Detailed discussion of the architecture of the MASC middleware, our prototype implementation, and its evaluation on case studies was published in [8].

```
<masc-se:When MASCID=”CurrencyConversionNeeded”>
  <masc-se:AllowedStates>
    <masc-se:StateRef To=”tns:Executing”/>
  </masc-se:AllowedStates>
  <masc-se:PossibleTriggerEvents>
    <masc-se:EventRef To=”tns:PortfolioValueReceived”/>
  </masc-se:PossibleTriggerEvents>
  <masc-ex:BooleanExpressionRef To=”tns:IntlPortfolio”/>
</masc-se:When>
…
<masc-ap:ActionPolicyAssertion MASCID=”AddCurrencyConversion”
  ManagementParty=”masc-cn:MASC_WSORCHESTRATOR”>
  <masc-se:WhenRef To=”tns:CurrencyConversionNeeded”/>
  <masc-ap:Actions>
   <!— The content of the following element is omitted for brevity -->
   <masc-ap:ProcessAddition> … </masc-ap:ProcessAddition>
  </masc-ap:Actions>
</masc-ap:ActionPolicyAssertion>
```

Figure 1. An Example of WS-Policy4MASC Constructs

6 Conclusions and Future Work

The existing solutions for specification of various aspects of Web service semantics are partial and often mutually incompatible, so a unifying framework is highly needed. We advocate that an extended and semantically-enriched WS-Policy can play a key unifying role in annotating WSDL and WSBPEL web service descriptions with various rules and support Web service management (monitoring and control), as well as service customization/versioning. The suggested WS-Policy extensions for the specification of goal policy assertions, action policy assertions, and utility policy assertions cover the same need as WSLA [10] and WS-Agreement [11], while these extensions plus the suggested specification of ontological meaning address the same need as OWL-S [1]. Therefore, our work enables that an XML Web service composition can be comprehensively described using only WSDL, WSBPEL, and our new WS-Policy4MASC. A particular novelty of WS-Policy4MASC are utility policy assertions and meta-policy assertions. They are used for specification of monetary and intangible business values and algorithms that are used for selection between alternative actions, respectively.

We have completed definition of the main XML schemas for WS-Policy4MASC. The focus of our past work was on supporting monitoring and dynamic adaptation (which is the main goal of the MASC project). While we made some progress towards specification of ontological meaning, we plan additions that will improve expressive power. Since these are relatively small additions, the main item for our ongoing work is further development of the proof-of-concept prototype implementation of the MASC middleware that uses WS-Policy4MASC policy assertions. While we already have a working prototype (discussed in [8]), we use an iterative development process to add new features into it (and, sometimes, the MASC architecture) and evaluate them on case studies. For example, the current MASC architecture and its prototype have no support for ontological reasoning, so some support might be added in the future. In some cases, changes to the MASC architecture require changes to the WS-Policy4MASC schemas (i.e., the language grammar), so our language will continue to evolve.

The complicated task of automating Web service policy interoperability (e.g., consistency checking) requires further research. Some of the challenges are that policies evolve over time and vary with service's deployment context (which may change dynamically) and runtime environment (which is constantly changing). Two main problems need to be addressed: (1) How can we ensure that composed services have compatible and consistent policies? (2) How can we dynamically detect and resolve/mediate conflicts between policies of composed services? Using WS-Policy extensions discussed in this paper, one stream of our future work will investigate novel algorithms and a policy middleware to address these open issues.

Acknowledgements:
The work on the MASC middleware and the WS-Policy4MASC language is a part of the research project "Building Policy-Driven Middleware for QoS-Aware and Adaptive Web Services Composition" led by Dr. Piyush Maheshwari and sponsored by the Australian Research Council (ARC) and Microsoft Australia. Please note that some of the proposals in this paper, such as the OntologicalMeaning construct, go beyond the scope of the mentioned research project.

The project is conducted at the University of New South Wales (UNSW). In addition to this university, Vladimir Tosic is affiliated with the Department of Computer Science, the University of Western Ontario, London, Canada, while Piyush Maheshwari is affiliated with IBM India Research Lab, New Delhi, India.

The authors thank A/Prof. Boualem Benatallah from the UNSW for his help in administration of this research project, insightful discussions, and comments.

References:

[1] D. Martin (ed.), *OWL-S: Semantic Markup for Web Services, version: 1.1 (Nov. 2004)*, WWW page at: www.daml.org/services/owl-s/1.1/overview/, 2004

[2] J. Schlimmer (ed.), *Web Services Policy Framework (WS-Policy), version: Sept. 2004*, WWW page at: www6.software.ibm.com/software/developer/library/ws-policy.pdf, 2004

[3] M. Sloman, Policy Driven Management for Distributed Systems, *Journal of Network and Systems Management*, Plenum, Vol. 2, No. 4, Dec. 1999, pp. 333-360.

[4] J. O. Kephart, and W. E. Walsh, An artificial intelligence perspective on autonomic computing policies, in *Proc. of Policy 2004* (June 2004, Yorktown Heights, USA), IEEE, 2004, pp. 3-12.

[5] V. Tosic, and B. Pagurek, On Comprehensive Contractual Descriptions of Web Services, in *Proc. of EEE-05* (March 2005, Hong Kong, China), IEEE, 2005, pp. 444-449.

[6] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma, Management Applications of the Web Service Offerings Language (WSOL), *Information Systems*, Elsevier, Vol. 30, No. 7, Nov. 2005, pp. 564-586.

[7] A. Erradi and P. Maheshwari, AdptiveBPEL: Policy-Driven Middleware for Flexible Web Services Composition, in *Proc. of the MWS 2005 workshop at EDOC 2005* (September 2005, Enschede, The Netherlands), IEEE, 2005, pp. 5-12.

[8] A. Erradi , P. Maheshwari, and V. Tosic, Policy-Driven Middleware for Self-Adaptation of Web Services Compositions, in *Proc. of Middleware 2006* (Melbourne, Australia, Nov. 27 - Dec. 1, 2006), *Lecture Notes in Computer Science (LNCS)*, Vol. 4290, Springer, 2006, pp. 62-80.

[9] A. Erradi (ed.), *Manageable and Adaptable Service Compositions (MASC)*, WWW page at: masc.web.cse.unsw.edu.au, 2006

[10] A. Keller, and H. Ludwig, The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services, *Journal of Network and Systems Management*, Plenum, Vol. 11, No 1, March 2003, pp. 57-81.

[11] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, *Web Services Agreement Specification (WS-Agreement), version 2006/09*, Global Grid Forum (GGF), 2006.