Semantic Management of Web Services

Daniel Oberle[1], Steffen Lamparter[1], Andreas Eberhart[2], and Steffen Staab[3]

[1] Institute AIFB, University of Karlsruhe, Germany
lastname@aifb.uni-karlsruhe.de
[2] Hewlett-Packard, Waldorf, Germany
andreas.eberhart@hp.com
[3] ISWeb, University of Koblenz-Landau, Germany
staab@uni-koblenz.de

Abstract. We present *semantic management of Web Services* as a paradigm that is located between the two extremes of current Web Services standards descriptions and tools, which we abbreviate by *WS**, and Semantic Web Services. On the one hand, WS* does not have an integrated formal model incurring high costs for managing Web Services in a declarative, but mostly manual fashion. On the other hand, the latter aims at the formal modelling of Web Services such that *full automation* of Web Service discovery, composition, invocation, etc., becomes possible — thereby incurring unbearably high costs for modelling. Based on a set of use cases, we identify who benefits from what kind of semantic modelling of Web Services, when and for what purposes. We present how an ontology is used in an implemented prototype.

1 Introduction

Different Web Service standards, we refer to them as *WS**, factorize Web Service management tasks into different aspects, such as input/output signatures, workflow, or security. The advantages of WS* are multiple and have already benefited some industrial cases. WS* descriptions are exchangeable and developers may use different implementations for the same Web Service description. The disadvantages of WS*, however, are also visible, yet: Even though the different standards are complementary, they must overlap and one may produce models composed of different WS* descriptions, which are inconsistent, but do not easily reveal their inconsistencies. The reason is that there is no coherent formal model of WS* and, thus, it is impossible to ask for conclusions that come from integrating several WS* descriptions. Hence, solving such Web Service management problems or asking for other kinds of conclusions that derive from the integration of WS* descriptions remains a purely manual task of the *software developers* accompanied by little to no formal machinery.

Researchers investigating Semantic Web Services have clearly articulated these shortcomings of WS* standardizations and have been presenting interesting proposals to counter some of them [1, 2]. The core of their proposals lies in creating *semantic* standards. Their principal objective is a wide-reaching formalization that allows *full automation* of the Web Service management tasks such as discovery and composition. The potential advantage is the reduction of management efforts to a minimum; the disadvantages, however, are also apparent: Neither is it clear, what kind of powerful machinery could constitute a semantic model that would allow for full automation, including all

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 514–519, 2005.

aspects of all web services that might matter in some way, nor does it appear to be possible that real-world developers could specify a semantic model of Web Services that would be fine-grained enough to allow for full automation anytime soon.

Therefore, we postulate that *semantic* management of Web Services should not try to tackle full automation of *all* Web Service management tasks as its objective. We claim that the full breadth of Web Service management requires an understanding of the world that is too deep to be modelled explicitly. Instead, we foresee a more passive role for semantic management of Web Services. One that is driven by the needs of the developers who must cope with the complexity of Web Service integration and who could use valuable tools for integrating previously separated aspects.

It is the contribution of this paper to clarify what kind of objectives could and should be targeted by semantics modelling of Web Services and to present a prototype that implements this framework. The kind of objectives that are to be approached are constrained by a *trade-off* between expending efforts for managing Web Services and expending efforts for semantic modelling of Web Services. At the one end, the objective of full automation by semantic modelling will need very fine-grained, detailed modelling of all aspects of Web Services — essentially everything that an intelligent human agent must know. Thus, *modelling efforts* skyrocket at the end of fine-grained modelling. At the other end, where modelling is very coarse and little modelling facilitates management, *management efforts* of distributed systems soar as experiences have shown in the past.

In this paper we try to approach the trade-off by identifying promising use cases. The use cases demonstrate that *some* management tasks can be facilitated by a justifiable amount of semantic modelling (section 2). For each use case, we identify who benefits from what kind of semantic modelling of Web Services, when and for what purposes. In addition, the use cases allow us to derive a set of modelling requirements for an appropriate management ontology which has been presented in [5]. We describe our implemented prototype, and detail how one of the use cases is realized by this system (cf. section 3) before we conclude.

2 Use Cases

This section discusses three use cases that trade off between management and modelling efforts (an extensive survey of use cases is given in [4]). That means, they propose to facilitate *some* of the typical Web Service management tasks by a justifiable amount of *semantic* descriptions (i.e., metadata in terms of an ontology). They try to approach the trade-off point by answering the following questions:

Question 1. Who uses the semantic descriptions of Web Services?

We see two major groups of users constituted by *(i)* software developers and *(ii)* administrators. These two groups of users have the need to predict or observe how Web Services interact, (might) get into conflict, (might) behave, etc. It will be very useful for them to query a system for semantic management of Web Services that integrates aspects from multiple WS* descriptions — which has not been possible so far, but is now allowed by the approach and system we present here.

Question 2. What does he/she/it use the semantic descriptions of Web Services for?

There is a large number of use cases where the integration of semantic descriptions may help the developer or administrator. Hence, the list below is neither exhaustive nor are the individual use cases mutually exclusive. The reader may note that it is germane to semantic descriptions to state what there is and not how it is to be combined and what is its sole purpose.

Question 3. When does he/she/it use the semantic descriptions of Web Services?

We consider development time, deployment time and runtime.

Question 4. Which aspects should be formalized by our ontology?

The answers to the last questions let us derive a set of modelling requirements for a suitable ontology.

Detecting Loops in Interorganizational Workflows. Web Services based applications usually make use of asynchronous messaging, bringing upon quite complex interaction protocols between business partners. Current workflow design workbenches only visualize the local flow and leave the orchestration of messages with the business partners up to the developer. Enough information is available in machine-readable format such that a tool can assist the developer in this task. For instance, the structure of the local flow can be combined with publicly available abstract flows of the partners in order to detect loops in the invocation chain that would lead to non-termination of the system. As shown in the bioinformatics domain [3], automated composition of workflows is likely to be inappropriate in most cases. Hence, we propose to support the developers in their management tasks and not to replace them.

Who:	Developer	*When:*	Development time
What for:	Code debugging	*Which aspects:*	Workflow information (plans)

Policy Handling. Policies play an increasing role, as demonstrated by the recent WS-Policy proposal. The idea of a policy is to lay out general rules and principles for service selection. Thus, rather than deciding whether an invocation is allowed on a case by case basis at runtime, one excludes services whose policy violates the local policy at development time. The major benefit is that policies can be specified declaratively. This use case does not aim at fully automated policy matching at run time, as we think that the full generality of policy matching imposes further problems that remain to be solved. Let alone the lack of WS-Policy engines so far. Instead we propose to apply semantic modelling in order to make policy handling more convenient for the developer. As our running example, we consider a large WS-BPEL workflow where checking for external task service invocations which are associated with a policy remains a tedious and manual task.

Who:	Developer, System	*What for:*	Excluding unsuitable services
When:	Development time	*Which aspects:*	Policies

Aggregating Service Information. Services will often be implemented based on other services. A service provider publishes information about its service. This might include service level agreements indicating a guaranteed worst-case response time, the cost of the service, or average availability measures. The service requestor, in this case a

composite service under development, can collect this information from the respective service providers. In turn, it offers a service and needs to publish similar measures. The semantic management must support the administrator supports the administrator by providing a first cut of this data by aggregating the data gathered from external providers. Similar to the statements given in [3], we argue that full automatic generation of such data will probably yield unwanted and inappropriate results. We see the computation results as an estimate which can be overridden manually by the administrator.

Who:	Administrator	*What for:*	Suggestion for deployment parameters
When:	Deployment time	*Which aspects:*	Quality of service

The answers to *Which aspects?* give us a clear indications of what concepts a suitable management ontology must contain. The organization of these concepts into our Core Ontology of Web Services is described in [5].

3 KAON SERVER

This section presents our prototype for semantic management of Web Services, called KAON SERVER.[1] We first discuss its architecture and demonstrate how it realizes our policy handling use case.

3.1 Overview

KAON SERVER is based on the open-source application server JBoss[2] and applies the tools of the KAON ontology toolsuite for reasoning and querying with the various aspects of Web services according to our Core Ontology of Web Services [5]. KAON SERVER obtains semantic descriptions from existing WS* descriptions, programme code, performance measurements, code reflection and modelling tools already in use. Obtaining comprises: i) parsing the XML documents, ii) extraction of relevant tags and iii) addition of the extracted information as instances to the ontology. The *Metadata Collector* component of the KAON SERVER carries out this task by taking the URLs of WS* descriptions as input. Runtime information stemming from monitoring components can be integrated, too. Another advantage of our approach is that the application logic (servlets, EJBs) may exploit the inference engine by reflection techniques in order to reflect on its own status. Finally, the developer might query the inference engine by using the admin console which is essentially an ontology browser with query interface.

3.2 Realizing the Policy Handling Use Case with the KAON SERVER

In this section we demonstrate how we have realized the policy handling use case by applying KAON SERVER. As an example for a conclusion derived from both a WS-BPEL and WS-Policy description, consider the following scenario. Let's assume a web shop realized with internal and external Web Services composed and managed by a WS-BPEL engine. After the submission of an order, we have to check the type of the

[1] Available at `http://kaon.semanticweb.org/server`

[2] `http://www.jboss.org`

```
...
 <process name="checkAccount">
  <switch ...>
   <case condition="getVariableData
     ('creditcard')='VISA'">
    <invoke partnerLink="toVISA"
     portType="visa:CCPortType"
     operation="checkCard"...>
    </invoke>
   </case>
  <case condition="getVariableData
   ('creditcard')='MasterCard'">
    <invoke partnerLink="toMastercard"
     portType="mastercard:CCPortType"
     operation="validateCardData"...>
   </invoke>
...
```

```
<wsp:Policy>
 <wsp:ExactlyOne>
  <wsse:SecurityToken>
   <wsse:TokenType>
    wsse:Kerberosv5TGT
   </wsse:TokenType>
  </wsse:SecurityToken>
  <wsse:SecurityToken>
   <wsse:TokenType>
    wsse:X509v3
   </wsse:TokenType>
  </wsse:SecurityToken>
 </wsp:ExactlyOne>
</wsp:Policy>
```

Fig. 1. WS-BPEL example on the left and WS-Policy example on the right hand side

customer's credit card for validity depending on the credit card type (VISA, MasterCard etc.). We assume that credit card providers offer this functionality via Web Services. The corresponding WS-BPEL process `checkAccount` thus invokes one of the provider's Web Services depending on the customer's credit card. The left hand side of Figure 1 below shows a snippet of the WS-BPEL process definition.

Suppose now that the Web Service of one credit card provider, say MasterCard, only accepts authenticated invocations conforming to Kerberos or X509. It states such policies in a corresponding WS-Policy document, such as the one sketched on the right hand side in Figure 1. The invocation will fail unless the developer ensures that the policies are met.

Applying KAON SERVER, checking for the existence of external policies boils down to simply querying the inference engine (cf. [5] for the complete example). Both the WS-BPEL process and the WS-Policy document are obtained by the metadata collector of KAON SERVER. That means, the documents are retrieved, parsed, relevant tags are extracted and added as instances to the ontology. WS-BPEL information and WS-BPEL processes are represented by means of the ontology. Note that for this example it suffices to model the existence of a policy and not the policy itself.

The developer can employ a simple query to find out whether an external service requires compliance with a specific policy. Without our approach the developer would have to collect and check this information manually by analyzing WS-BPEL and WS-Policy documents.

As we may recognize from this small example, it is desirable to pose a query rather than manually checking a complex set of process definitions. Without KAON SERVER, the developer would have to check all WS-BPEL nodes for external invocations and corresponding WS-Policy documents manually at development time. We encounter more sophisticated examples where we query for particular policy constraints or where we have large indirect process cascades.

As mentioned in the policy handling use case in section 2, we do not aim at fully automated policy matching at run time, as we think that the full generality of policy matching imposes further problems that remain to be solved. In addition, there are no WS-Policy engines available so far.

Finally, Table 1 shows the benefit of our approach by comparing the effort with and without semantic management for the running example. While using the paradigm of

Table 1. Effort comparison for the running example

Effort	Without semantics	Using semantic management
Management	For each process in the WS-BPEL document: Check for external Web service invocation and check for existence of WS-Policy document	One query to retrieve external Web service processes with attached policies
Modelling	creating and maintaining the WS-BPEL and WS-Policy documents	Same as without semantics because semantic descriptions are automatically obtained

semantic management of Web Services reduces management efforts, no additional modelling efforts are required because KAON SERVER obtains the semantic descriptions automatically from WS* documents.

4 Conclusion

We have shown in this paper what *semantic management of Web Services* may contribute to Web Service management in general. We have described use cases for semantic management of Web Services that can be realized with existing technology and that provide immediate benefits to their target groups, i.e. software developers and administrators who deal with Web Services. Through the use cases we have shown that semantic descriptions may play a fruitful role supporting an integrated view onto Web Service definitions in WS*. At the basis of the integration we have put our Core Ontology of Web Services.

While we have implemented a prototype as proof-of-concept of our approach, in the long run the viability and success of semantic descriptions will only be shown in their successful use of integrated development and runtime environments. The development of the corresponding paradigm of Semantic Management of Web Services through use cases, ontologies, prototypes and examples is an important step into this direction.

Acknowledgements. This work was financed by WonderWeb, an EU IST project, by SmartWeb, a German BMBF project and by ASG (IST-004617), an EU IST project.

References

1. V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A service creation environment based on end to end composition of web services. In *Proceedings of WWW 2005*, pages 128–137. ACM, 2005.
2. R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web Service Semantics - WSDL-S. Technical report, University of Georgia, Apr 2005.
3. P. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying Semantic Web Services to Bioinformatics: Experiences Gained, Lessons Learnt. In *3rd Int. Semantic Web Conference*, volume 3298 of *LNCS*. Springer, 2004.
4. D. Oberle, S. Lamparter, A. Eberhart, and S. Staab. Semantic management of web services. Technical report, University of Karlsruhe, 2005.
5. D. Oberle, S. Lamparter, S. Grimm, D. Vrandecic, S. Staab, and A. Gangemi. Towards ontologies for formalizing modularization and communication in large software systems. Technical report, University of Karlsruhe, 2005.