Semantic Management of Web Services

Daniel Oberle, Steffen Lamparter
Institute AIFB
University of Karlsruhe
Germany
lastname@aifb.uni-karlsruhe.de

Andreas Eberhart
Hewlett-Packard
Waldorf
Germany
andreas.eberhart@hp.com

Steffen Staab
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 (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. Semantic management of Web Services trades off between the two extremes. 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 arrive at a scheme that is comparable to model driven engineering (MDE), but which is more powerful and not restricted for use at compile time. In addition, the use cases also determine requirements of our semantic modelling leading us to a Core Ontology of Services. We present how the core ontology is used in an implemented prototype.*[1]

1. Introduction

Different Web Service standards, WS*, factorize Web Service management tasks into different aspects, such as input/output, 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, discovering 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 [5]. 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. Again, the potential advantages are obvious; 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, nor does it appear to be possible that real-world software 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 WS* descriptions, who could use valuable tools for integrating previously separated aspects, but who rely on their own abstracting understanding (at which they arrive with the help of the semantic tools) of the situation.

It is the purpose of this paper to clarify what kind of objectives could and should be targeted by semantics modelling of Web services, to describe characteristics of ontolo-

1 Maintained at `http://kaon.semanticweb.org/server`

gies that support this task and to present a prototype that implements this framework.

The kind of objectives that are to be approached are constrained by a *costs trade-off* between investing efforts for managing Web Services and investing efforts for semantic modelling of Web Services. The costs tradeoff is depicted qualitatively in Figure 1. 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 costs* skyrocket at the end of fine-grained modelling. At the other end, where modelling is very coarse and little modelling facilitates management, costs for managing distributed systems soar as experiences have shown in the past. No matter what the exact scale of granularity and costs are, the qualitative indication of management and modelling costs such as done in Figure 1 leads to an overall total cost picture as indicated in the same figure.

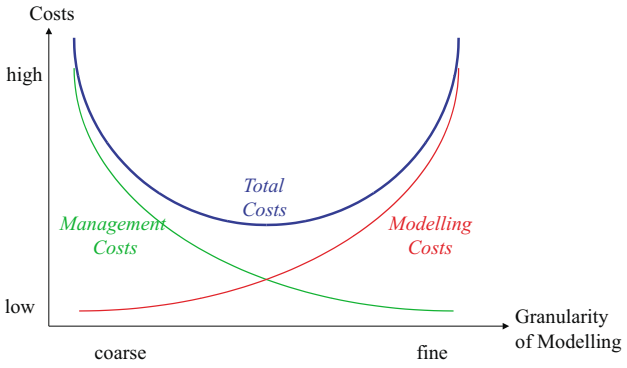

Figure 1. Working Hypothesis

In this paper we elaborate on how to find a good trade-off between modelling costs by considering user requirements and management costs by considering available semantic modelling technology. In order to approach the trade-off point, we follow the strategy that we first identify promising use cases for exploiting 'simple' semantic modelling (section 2). We identify who benefits from what kind of semantic modelling of Web Services, when and for what purposes, based on the use cases. We then provide a solid, but easily reusable ontology foundation to minimize modelling costs and respond to modelling requirements derived from the use cases (cf. section 3). We describe the rationale underlying our prototype (cf. section 4), before we tie together these pieces by an example (cf. section 5). Eventually, we compare to related work and conclude.

2. Use Cases

This section discusses promising use cases for exploiting semantic modelling. They try to approach the trade-off point between modelling and management cost by identifying who benefits from what kind of semantic modelling of Web Services, when and for what purposes. In particular, each use case will answer 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. As a third 'group of users', we foresee that developers program dynamically using inspection-like mechanisms to query and modify semantic descriptions in order to allow for autonomous control of Web Service interaction. Thus, running Web Services constitute a third group.[2]

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 three different stages, viz. development time, deployment time and runtime.

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

One the one hand, we want to be able to automate management tasks covering a broad range of aspects (like security, policies, interface descriptions etc.). On the other hand, the complexity of the ontology has to be kept small to avoid overburdening the developer. The use cases serve as modelling requirements also for building a suitable management ontology in section 3.

2 Our approach here focuses on the user groups of developers and administrators. We consider the autonomous exploitation by programmes rather a, desirable, side effect of our approach.

2.1. Analyzing Message Contexts

Message passing plays *the* central role for Web Services. A message sent to a component can in turn trigger several other messages being sent out on behalf of the initial message. Messages may carry a context with information about the sender, the sender's credentials, or the message's transactional context. During the deployment of a component or flow, the administrator makes important choices as to how messages are propagated. These include whether the sender information is carried along or whether the new message is sent on behalf of a new user (also called the run-as paradigm). Similar choices are made with respect to the transactional settings. Components and flows can choose to always open a new transaction, require a prior transactional context, or open a new transaction when needed. In a large network of direct and indirect invocations, it is crucial to be able to detect configuration errors, such as a situation where a component switching to user context X and calling Y, which does not have user X in its access control list.

Who: Administrator
What for: Plausibility check on deployment settings
When: Deployment time
Which aspects: Service profile

2.2. Selecting Service Functionality

As more and more services become available, developers will need some tool support in browsing and selecting an appropriate service. The canonical approach to this task is a taxonomic categorization of services according to their capabilities. Naturally, searching for services of a certain capability class C should also yield all services classified as instances of subclasses of C.

Who: Developer
What for: Service discovery and selection
When: Development time
Which aspects: Service taxonomy

2.3. Policy Matching

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.

Since policies can change dynamically, one can also imagine a scenario, where a service bus matches policies during runtime in order to select an appropriate provider out of a pool of competitors. At this point, the reader may note, that the current policy-related proposals are not based on a uniform logical framework. We believe that this would make the realization of a policy engine and the integration of the various approaches from the areas of privacy, security, or access control much easier.

Who: Developer, System
What for: Excluding unsuitable services
When: Development and runtime
Which aspects: Policies

2.4. Detecting Loops in the Invocation Chain

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. We believe that 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.

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

2.5. Incompatible Inputs and Outputs

Type checking is not as straightforward anymore using loosely coupled services operated by a large number of organizations. Furthermore, the interpretation of a B2B term such as 'price' might be different, even though syntactically it refers to an agreed-upon XML Schema type. For instance, different partners might have different assumptions about the currency and taxation details. A system, which automatically compares communication inputs and outputs according to a more detailed model, will help to prevent unexpected behavior in a system.

Who: Developer
What for: Code debugging
When: Development time
Which aspects: Interface description

2.6. Relating Communication Parameters

This use case is again motivated by e-business policies. Assume a policy states that the entire supply chain must consist of ISO 9000 certified partners. Enforcing this policy requires correlating communication paths with information about the organizations operating the communication endpoints. Another example would be a policy stating that confidential information should only be sent across a secure communication channel. In this case, knowledge about message payload types such as credit card information must be connected with the properties of the underlying transport.

Who:	Developer, Administrator
What for:	Plausibility check on flow
When:	Development and deployment time
Which aspects:	Service profile

2.7. Change Management

A system no longer being under the tight control of a single organizational unit will definitely be prone to service versioning issues. Updating a single component already requires close cooperation between the parties involved and this will without a doubt be much harder in Web Services based applications. Consequently, the respective process composition tool suite should provide support for monitoring the providers' service interface definitions.

Who:	Developer
What for:	Code debugging
When:	Development time
Which aspects:	Interface description

2.8. 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 numbers. 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 numbers. We envision a tool that supports the administrator by providing a first cut of this data by aggregating the data gathered from external providers.

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

2.9. Quality of Service

While the previous use case was based on data gathered from service providers, one might want to obtain own statistics on the reliability and availability of business partners' IT infrastructure. Assuming the system is aware of potential endpoints implementing a required service, these endpoints can be pinged regularly. If an actual request arrives, aggregated availability information can be used to direct subsequent requests to one or the other third party service.

Likewise, a provider needs to make sure it provides an adequate service level for its customers. In case of performance bottlenecks, it might have to make an educated decision on which jobs to grant higher priority and which job to drop or decline. Existing service level agreements and of course the respective penalties play an important role here.

Who:	Administrator, System
What for:	Performance optimization
When:	Runtime
Which aspects:	Quality of service

3. Ontology

The use cases we presented in the previous section introduced modelling requirements: In order the realize semantic management of Web services we have to model *service profiles*, *service taxonomies*, *policies*, *workflow information*, *interface descriptions* as well as *quality of service information*.

Available semantic modelling technologies provide a powerful means that support us here: ontologies. They formalize concepts and concept relationships (associations) very similar to conceptual database schemata or UML class diagrams. However, ontologies typically feature logic-based representation languages that come with executable logic calculi. The calculi allow us to reason and query at runtime [10].

For semantic management of Web Services we need to formalize an ontology that meets the aforementioned modelling requirements, thus weaving together aspects from multiple WS* descriptions. Querying and reasoning will enable us to ask for conclusions that come from integrating several WS* descriptions — e.g. predicting or observing how Web Services interact, (might) get into conflict, (might) behave, etc. — what had to be done manually before.

In this section, we present a Core Ontology of Services[3] that models all the necessary aspects. Some parts are already captured by the DOLCE foundational ontology [8], the Ontology of Plans [3] and the Core Legal Ontology [2].

3 Maintained at `http://cos.ontoware.org`.

Therefore, our Core Ontology of Services reuses them and adds the remaining parts like discussed below.

At the heart of the Core Ontology of Services, we find the Service Management Description that defines several sub-descriptions to conquer the complexity and to have a modular design. Figure 2 sketches the descriptions in an UML diagram for the sake of readability. The following paragraphs discuss the sub-descriptions briefly[4]:

- Information contained in a Service Management Description may be obtained from several distributed sources like WSDL and WS-Policy documents, performance measurements or code reflection. We thus want to model Information Source and Age of Information for every sub-description. Specializations of the Service Management Description may be put into a *service taxonomy* (cf. use case 2.2). It also allows to store typical *service profile* information as required by use cases 2.1 and 2.6.
- We reused the Core Legal Ontology (CLO) [2] for the Policy Description. The CLO concentrates on the ontological nature of the legal domain, i.e. the entities of the mental, social, or properly legal world, and the constraints that legal norms are supposed to generate over those worlds. Its basic concepts and associations meet our requirements of expressing Web Service *policies* (cf. use case 2.3) and they are general enough to harmonize existing efforts like WS-Policy or XACML that all introduce their own vocabulary. It also introduces theories of satisfaction that can be exploited later to match policies. Basically, the Policy Description talks about Users, their rights towards particular Tasks and Constraints on these Tasks. A Service Management Description can have several Policy Descriptions to reflect alternatives.
- The Interface Description corresponds to the modelling requirement of the same name and thus to use cases 2.5 and 2.7. The main concept is Operation which is associated with an arbitrary number of Computational Inputs and Computational Outputs. All are Functional Roles which are played-by Information Objects and Datatypes (a special kind of Information Object), respectively.
- The Quality of Service Description groups together service and runtime information. It meets the modelling requirement to express *quality of service* information as introduced in use cases 2.8 and 2.9. At the moment we consider Maximum Duration, Worst-case Response Time, Cost and Average Availability but also Certificates that model information about ISO 9000 conformity, for instance.
- Components of the Plan Description are mostly taken from the Ontology of Plans [3]. It introduces the notion of Task with according specializations (case, branching, synchronization, concurrency, cycling, etc.), successor and predecessor associations between Tasks. It provides a way of modelling *workflow information* (cf. use case 2.4) platform-independently and allows to conceptually harmonize efforts like BPEL that introduce their own vocabulary.

4. Prototype

After formalizing the Core Ontology of Services we propose a way to implement semantic management of Web Services. On the one hand, we have to think about who or what will provide semantic descriptions of Web Services. On the other hand, there are several target platforms where the semantic technology can be integrated. In this section, we answer these questions leading us to our prototype — the KAON SERVER.

Harvesting semantic descriptions The amount of semantic descriptions that are provided manually by the software developer must be as small as ever possible, because software developers will not be very willing to adopt a large new paradigm at a time when they are just getting used to WS*. In our prototype we harvest semantic descriptions from programme code and modelling tools already in use. They are obtained from several distributed sources like WSDL and WS-Policy documents, performance measurements or code reflection.

Platform for semantic management There are several obvious target platforms to integrate the semantic management of Web Services, e.g. enterprise application management tools like IBM Tivoli, software IDEs like Eclipse, workflow engines like Microsoft's Biztalk or Application Servers. For our prototype we have chosen the latter option because of two reasons: a) Application Servers come in handy as industry-strength Web Services based applications are often realized with Application Servers anyway, b) semantic technologies are already integrated in our ontology-based Application Server, called *KAON SERVER*[5]. It is based on the open-source Application Server JBoss[6] and adds ontology infrastructure in order to reason with software components like EJBs or Servlets [6, 7].

Implementation The approach we follow is non-invasive and does not intervene in the existing infrastructure, i.e. ex-

4 Parts of the ontology are written in sans serif, references to the modelling requirements introduced in section 2 are written in *italics*.

5 Available at `http://kaon.semanticweb.org/server`

6 `http://www.jboss.org`

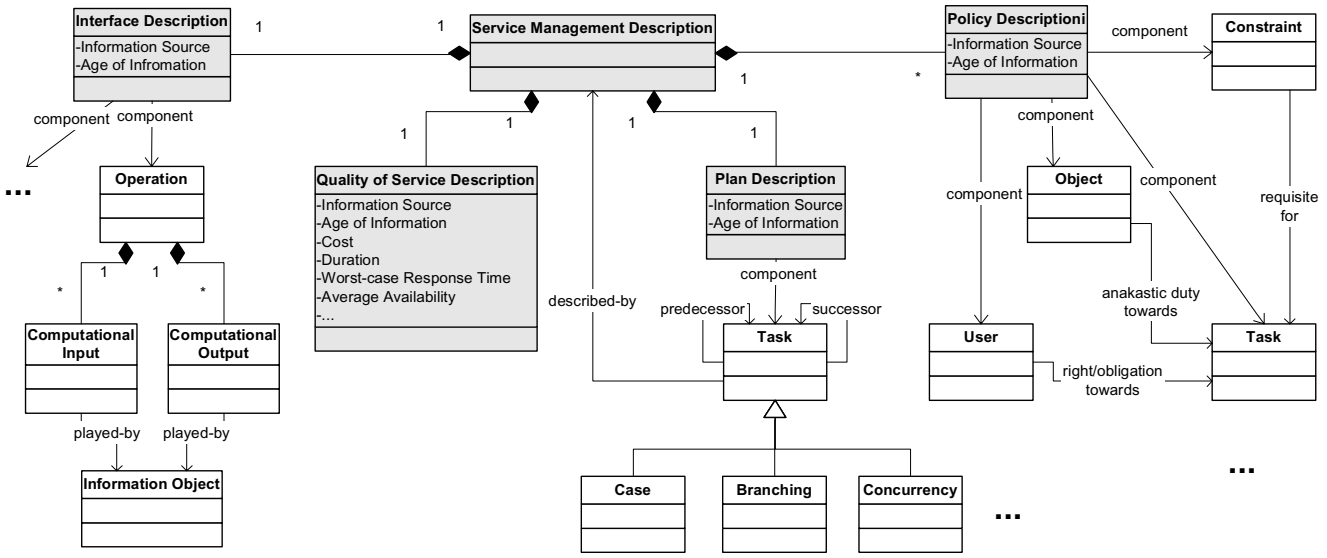

Figure 2. Sketch of the Core Ontology of Services

isting WS* descriptions are still fed into their corresponding engines for security, transaction or workflow. It is still necessary for the developer to familiarize and work with WS*. However, WS* descriptions are harvested (i.e. parsed and integrated) into the ontology what enables the developer to query and reason with a harmonizing conceptual model spanning several aspects.

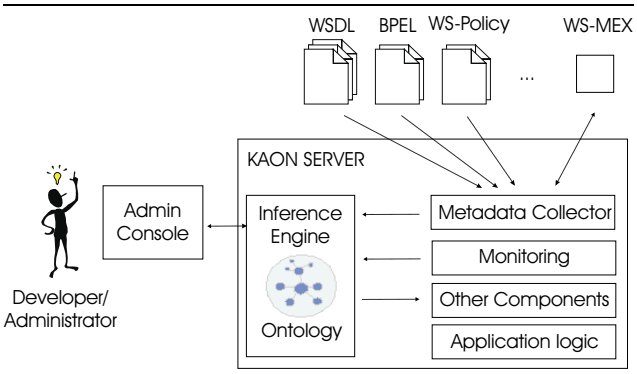

Figure 3. Integration in KAON SERVER.

Figure 3 illustrates the general idea. The KAON SERVER features an *Inference Engine* to reason with software components like EJBs or Servlets. Its actual *Ontology* is an applied version of the Core Ontology of Software Components. We extend it by the Core Ontology of Services to formalize relevant management aspects of Web Services and the relationship to software components. The integration of ontologies is feasible because all of them feature the same ontological foundation in the form of DOLCE.

We assume a web application being built by Servlets or EJBs that in turn invoke a number of Web Services, i.e. they contain the "wiring" *Application Logic*. A new *Metadata Collector* component harvests *WSDL*, *BPEL* or *WS-Policy* documents of used Web Services. We might also leverage WS-MetaDataExchange (*WS-MEX*) — a proposal that enriches each Web Service with standardized methods for retrieving WSDL, XML-Schema and WS-Policy information. The component integrates the data into the ontology. Runtime information stemming from the *Monitoring* component can be integrated, too. The developer might query the inference engine by using the *Admin Console* which is essentially an ontology browser with query interface.

5. An Example

As an example for a conclusion derived from both a BPEL and WS-Policy description, consider the following case. Let's assume a web shop realized with internal and external Web Services composed and managed by a BPEL engine. After the submission of an order, we have to check the 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 BPEL process `checkAccount` thus invokes one of the provider's Web Services depending on the customer's credit card. The example below shows a snippet of the BPEL process definition.

```
...
<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>
   </case>
  ...
  </switch>
</process>
...
```

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 like the one sketched below. The invocation will fail unless the developer ensures that the policies are met. That means the developer has to check the policies manually at development time or has to implement this functionality to react to policies at runtime.

```
...
<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>
...
```

Since process and policy information can be harvested and integrated in our ontology, checking for the existence of external policies boils down to a simple query. An external service invocation is represented by a successor association between two Tasks where the second is described-by another Service Management Description (SMD). The simplified formula below grasps this knowledge.

$$\forall smd_1, smd_2, t_1, t_2 : externally\_invokes(smd_1, smd_2) \leftarrow SMD(smd_1) \wedge SMD(smd_2) \wedge Task(t_1) \wedge Task(t_2) \wedge component(smd_1, t_1) \wedge component(smd_1, t_2) \wedge successor(t_1, t_2) \wedge described\_by(t_2, smd_2)$$

Consequently, the developer could employ the query below to find out whether an external service requires compliance with a certain policy. The result of the query yields all smd that $externally\_invokes$ services with a Policy Description (PD) attached. Without our approach the developer would have to collect and check this information manually by analyzing BPEL and WS-Policy documents.

$$\forall smd :\leftarrow externally\_invokes(smd_{local}, smd) \wedge SMD(smd_{local}) \wedge \exists x : component(smd, x) \wedge PD(x)$$

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. We can think of more sophisticated examples where we query for particular policy constraints or where we have large indirect process cascades. In the latter case, it would suffice to declare the successor association as transitive and rule and query above can remain unchanged :

$$\forall t_1, t_2, t_3 : successor(t_1, t_3) \leftarrow successor(t_1, t_2) \wedge successor(t_2, t_3)$$

6. Related Work

On the one hand, developers have to face the multitude of WS* specifications like WSDL, WS-Policy, WS-Coordination, WS-Transaction or BPEL. Because of their sheer number and disjointness, managing Web Services with WS* creates high costs for the developer. There is no coherent formal model of WS* and there are no means to ask for, possibly undesirable, conclusions that arise from integrating several WS* descriptions.

On the other hand, several *semantic standards* are arising at the moment in a field of research that is often circumscribed as "Semantic Web Services" [5]. Unlike our approach, they aim at full automation of Web Service invocation, discovery and composition. However, common ontology languages are typically not expressive enough to reach these objectives. E.g. [5] uses a description logic which is extended by Golog to automate planning tasks. Golog is a high level programming language built on top of the situation calculus. In addition, the objective of full automation by semantic modelling will need very fine-grained, detailed modelling of all aspects of Web Services leading to high modelling costs. Our approach is located between the

two extremes of WS* and semantic standards and finds a good trade-off between modelling and management costs. The following paragraph discusses some of the semantic efforts.

Two approaches try to incorporate semantic technology in UDDI, for instance. The first, [12], proposes a taxonomy support for semantics in the registry. The primary aim is to allow for a better discovery and matchmaking by leveraging the semantic descriptions. The second tries to achieve similar goals by incorporating OWL-S profiles into the UDDI registry [9]. OWL-S [11] is one of the first core ontologies explicitly aiming at automatic discovery, automatic invocation, automatic composition and interoperation as well as automatic execution of Web Services. The Web Service Modelling Ontology (WSMO) [1] has goals similar to OWL-S. However, it additionally defines an Execution Environment (WSMX) for the dynamic discovery, selection, mediation, invocation and inter-operation of Semantic Web Services. [4] take into account that most semantic efforts have been disconnected from the emerging WS* standards. Hence, they propose a "bottom-up" approach of enriching BPEL by semantics. However, they also try to enable automated service discovery, customization, and semantic translation.

Our approach is similar to the way that model driven engineering (MDE) tackles a problem, viz. by abstracting modelling of some parts of an architecture, while retaining full control by the software engineer. MDE/MDA and semantic management however differ, because the latter constitutes a precise, formal, executable model that may be exploited not only at compilation time (like MDE/MDA), but also for hot deployment or during runtime.

Finally, we discussed the semantic management of software components in an Application Server in [6]. However, this work focusses on reusable ontology design to reduce modelling costs. Although the problems are similar, the situation here is more complex due to the mere fact of distribution what entails network delays, reliability, trust or additional security issues.

7. 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 the Core Ontology of 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 is financed by WonderWeb, an EU IST project, by SmartWeb, a German BMBF project and by ASG, an EU IST project. We are indebted to Aldo Gangemi, LOA, Rome, for the hints on the ontology.

References

[1] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce: Research and Applications*, 1:113–137, 2002.

[2] A. Gangemi, , M.-T. Sagri, and D. Tiscornia. A Constructive Framework for Legal Ontologies. Internal project report, EU 6FP METOKIS Project, Deliverable, 2004.

[3] A. Gangemi, S. Borgo, C. Catenacci, and J. Lehmann. Task taxonomies for knowledge content. Metokis deliverable d07, Jun 2004.

[4] D. J. Mandell and S. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In *2nd Int. Semantic Web Conference*, volume 2870 of *LNCS*, pages 227–247. Springer, 2003.

[5] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, Mar 2001.

[6] D. Oberle, A. Eberhart, S. Staab, and R. Volz. Developing and managing software components in an ontology-based application server. In *5th International Middleware Conference*, LNCS. Springer, 2004.

[7] D. Oberle, S. Staab, R. Studer, and R. Volz. Supporting Application Development in the Semantic Web. *ACM Transactions on Internet Technology (TOIT)*, 4(4), Nov 2004.

[8] A. Oltramari, A. Gangemi, N. Guarino, and C. Masolo. Sweetening ontologies with DOLCE. In *Ontologies and the Semantic Web, 13th Int. Conference, EKAW 2002, Proceedings*, 2002.

[9] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Importing the Semantic Web in UDDI. In *CAiSE 2002 International Workshop, WES 2002*, pages 225–236, 2002.

[10] S. Staab and R. Studer. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer Verlag, Heidelberg, 2004.

[11] The DAML Services Coalition. OWL-S 1.0 draft release. `http://www.daml.org/services/owl-s/1.0/`, Dec 2003.

[12] M. Voskob. UDDI Spec TC V4 Requirement - Taxonomy support for semantics. OASIS, 2004. `http://www.oasis-open.org`.