

Semantic Description of Behavior and Trustworthy Credentials of Web Services

Sudhir Agarwal and Sebastian Rudolph

Institute of Applied Informatics and Formal Description Methods (AIFB),
University of Karlsruhe (TH), Germany.
{agarwal,rudolph}@aifb.uni-karlsruhe.de

Abstract. The ever increasing need for outsourcing and automated execution of parts of business processes will lead to a large number of Web services. The resulting large market of Web services demands for automatic methods for finding, composing and selecting Web services according to some criteria specified by the user. In real business scenarios, the user's criteria are richer than just the correctness of input and output types. In order to develop and deploy algorithms that deal with those richer user criteria, functional and non-functional properties of Web services must be described formally.

In this paper, we present a novel combination of the polyadic π -calculus and the description logic $SHOIN(\mathbf{D})$ for describing functional properties of Web services. Furthermore, we present semantic-SPKI/SDSI certificates for describing non-functional properties of Web services in an interoperable and provable way while allowing users to define and reason about their trust in them. We present a prototype for semi-automatic extraction and management of Web service descriptions specified in the presented formalism.

1 Introduction

Web services are primarily meant for being integrated in software applications. A Web service can be integrated in a software application, if it fulfills constraints posed by the application. For example, consider a library system that manages a list of books that need to be purchased using their ISBN as identifier. Since the number of books to be ordered can be large, the library systems needs a Web service for ordering books automatically. For a Web service to be considered as suitable for the library system, it should accept an ISBN as input and output (i.e., deliver to the customer) a book with this very ISBN. Further, assume that the library system expects an order confirmation before the book is actually delivered. So, a suitable Web service should behave accordingly in order to fulfill the library's requirements. We call the properties of a Web service on the basis of which it can be decided whether the Web service is appropriate for a given software application or not *functional properties*.

Apart from the functional properties of a Web service, the library system might be interested in some quality of service attributes, e.g. that the Web

service should have fast delivery times, the Web service provider should have friendly employees and he should regularly donate for humanitarian purposes. We call such properties *non-functional properties*.

In order to automatically find, compose, and select Web services according to user criteria that may contain constraints on functional as well as on non-functional properties of Web services, both types of properties must be described formally. The approach we provide in this paper accomplishes this goal by integrating well-established formalisms from the areas of semantic knowledge representation, process specification, and certificate theory. A preliminary version of the approach has been presented in [1].

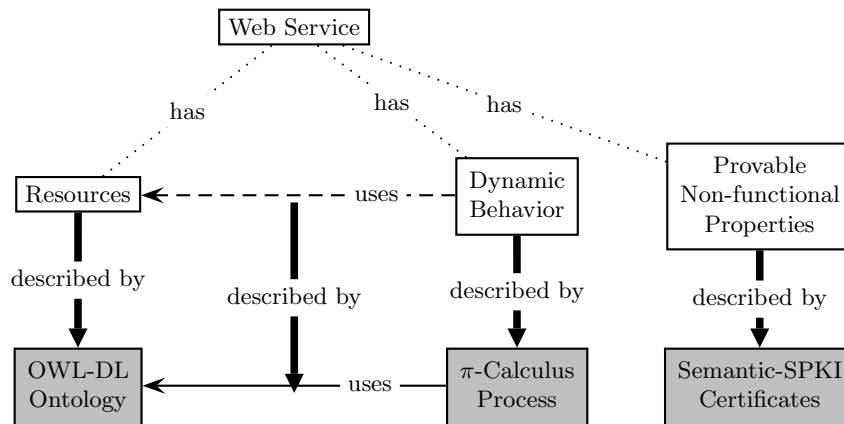


Fig. 1. Our Web Service Modeling Approach

Figure 1 shows the main idea behind our Web service modeling formalism. We consider functional and non-functional properties of Web services. In order to model functional properties, we consider resources and dynamic behavior of Web services. In Section 2, we show how resources can be modeled as $SHOIN(\mathbf{D})$ ontologies with DL-safe rules and how the dynamic behavior can be modeled as π -calculus process. Our main contribution regarding modeling of functional properties is the establishment of the connection between π -calculus process descriptions and $SHOIN(\mathbf{D})$ ontologies. Non-functional properties are critical in real business scenarios. In Section 3, we turn our attention to modeling non-functional properties of Web services. Our approach is based on the observation that if non-functional properties are not modeled in a way that they are provable then they are hardly of any use. We borrow ideas from certificate theory, especially Simple Public Key Infrastructure (SPKI) and present Semantic-SPKI, our semantic extension of SPKI. We then show how provable non-functional prop-

erties can be modeled as Semantic-SPKI certificates. In Section 4, we present our prototype for modeling and managing Web service descriptions. Finally, we discuss some related work in Section 5 and conclude in Section 6.

2 Modeling Functional Properties

In general, a Web service involves several actors that communicate with each other by exchanging resources. These resources can be real world objects as well as pieces of information.

We denote the finite set of all agents with \mathcal{A} . Each agent $A \in \mathcal{A}$ has a finite set \mathcal{R}_A of resources available. We use the description logic $\mathcal{SHOIN}(\mathbf{D})$ for modeling resources and resource schemas in an interoperable and machine understandable way [2].

2.1 Modeling Resources

Short Introduction to Description Logics A $\mathcal{SHOIN}(\mathbf{D})$ description logic knowledge base consists of a set of axioms, which can be distinguished into terminological axioms (building the so-called TBox \mathcal{T}) and assertional axioms or assertions (constituting the ABox \mathcal{A}). Based on names for concepts (as C, D, \dots), roles (R, S, \dots), and individuals (a, b, \dots), $\mathcal{SHOIN}(\mathbf{D})$ provides the following constructors to build complex concepts from simpler ones: *negation* $\neg C$, *conjunction* $C \sqcap D$, *disjunction* $C \sqcup D$, *existential quantifier* $\exists R.C$, *universal quantifier* $\forall R.C$, *cardinality constraints* $\geq nS$ and $\leq nS$, *nominals* $\{a_1, \dots, a_n\}$. Further, it supports concrete datatypes and there exist corresponding axioms for quantifiers and cardinality constraints for roles with a datatype range. A TBox consists of a finite set of concept inclusion axioms $C \sqsubseteq D$, where C and D are either both concepts or relations. The A-Box consists of a finite set of *concept assertions* $C(a)$, *role assertions* $R(a, b)$, *individual equalities* $a = b$, and *individual inequalities* $a \neq b$. Those assertional axioms or assertions introduce individuals, i.e. instances of a class, into the knowledge base and relate individuals with each other. The semantics of $\mathcal{SHOIN}(\mathbf{D})$ is based on an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set (the domain) and $\cdot^{\mathcal{I}}$ assigns to each concept name C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and to each role name R a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Based on those assignments, the validity of inclusion atoms and assertions is decided. For details about the semantics of $\mathcal{SHOIN}(\mathbf{D})$ constructors, T-Box axioms and A-Box axioms, we refer to [2,3,4]. A decidable rule extension of $\mathcal{SHOIN}(\mathbf{D})$, so-called DL-safe rules was presented in [4].

Definition 1 (DL-safe Rules). Let N_C denote the set of concept names, N_{R_a} the set of abstract roles names and N_{R_c} the set of concrete roles names. Let N_P be the set of predicate symbols such that $N_C \cup N_{R_a} \cup N_{R_c} \subseteq N_P$. A DL-atom is an atom of the form $A(s)$, where $A \in N_C$ or of the form $R(s, t)$, where $R \in N_{R_a} \cup N_{R_c}$. A rule r is called DL-safe if each variable in r occurs in a non-DL-atom in the rule body.

Modeling Resources We specify concrete resources as description logic individuals, among which relationships “=” and “≠” can be specified. These relationship types are necessary to achieve interoperability in the descriptions of individuals and are directly provided by expressive description logics, e.g. *SHOIN(D)*, which the decidable variant OWL-DL of the Web ontology language OWL¹ is also based on. The resources can be further classified into sets that can be hierarchically ordered according to the subset relationship. Again, expressive description logics provide the “⊆” relationship type to relate the sets. So, \mathcal{R}_A is a description logics ontology whose A-Box describes resources and relationships among them and whose T-Box contains axioms about the classification of resources of an agent $A \in \mathcal{A}$.

2.2 Modeling Behavior

Now, we turn our attention to modeling the third aspect of an agent, namely, his dynamic behavior. We use π -calculus for modeling the dynamic behavior so we first introduce it briefly and refer to [5,6] for details.

Short Introduction to Pi-Calculus π -calculus is a formalism for modeling labeled transition systems. The syntax for specifying agents can be summarized as follows:

$$P ::= \mathbf{0} \mid \bar{y}x.P \mid y(x).P \mid \tau.P \mid P_1 \parallel P_2 \mid P_1 + P_2 \mid \omega?P:Q \mid (\mathbf{new} \ x)P \mid A(y_1, \dots, y_n)$$

Null Process: $\mathbf{0}$ is a process that does nothing.

Prefix form: $\bar{y}x$ is called a negative prefix. \bar{y} may be thought of as an output port of an agent which contains it; $\bar{y}x.P$ outputs the name x at port y and then behaves like P . $y(x)$ is called a positive prefix. A name y may be thought of as an input port of an agent; $y(x).P$ inputs an arbitrary name z at port y and then behaves like $P\{z/x\}$, where z/x denote substituting z for x . The name x is bound by the positive prefix ‘ $y(x)$ ’. (Note that a negative prefix does not bind a name). τ is called a silent prefix. $\tau.P$ performs the silent action τ and then behaves like P .

Composition: $P_1 \parallel P_2$ consists of P_1 and P_2 acting in parallel. The components may act independently; also, an output action of P_1 (resp. P_2) at any output port x may synchronize with an input action of P_2 (resp. P_1) at x , to create a silent (τ) action of the composite agent $P_1 \parallel P_2$.

Summation: $P_1 + P_2$ behaves either like P_1 or like P_2 .

Match: $\omega?P:Q$ behaves like P if the condition ω is true, and otherwise like Q .

¹ <http://www.w3.org/2004/OWL/>

Restriction: $(\text{new } x)P$ restricts the scope of name x to the process P . Components of P can use x to interact with one another but not with other processes.

Agent Identifier: For any agent identifier A (with arity n), there must be a unique defining equation $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} P$, where the names x_1, \dots, x_n are distinct and are the only names which may occur unbound in P . Then $A(y_1, \dots, y_n)$ behaves like $P\{y_1/x_1, \dots, y_n/x_n\}$. Note that defining equations provide recursion, since P may contain any agent identifier, even A itself. The operational semantics of π -calculus maps a π process expression to a labeled transition system by viewing operations (communication operations and silent operation) as transitions and process expressions as states [5]. Polyadic π -calculus is a powerful

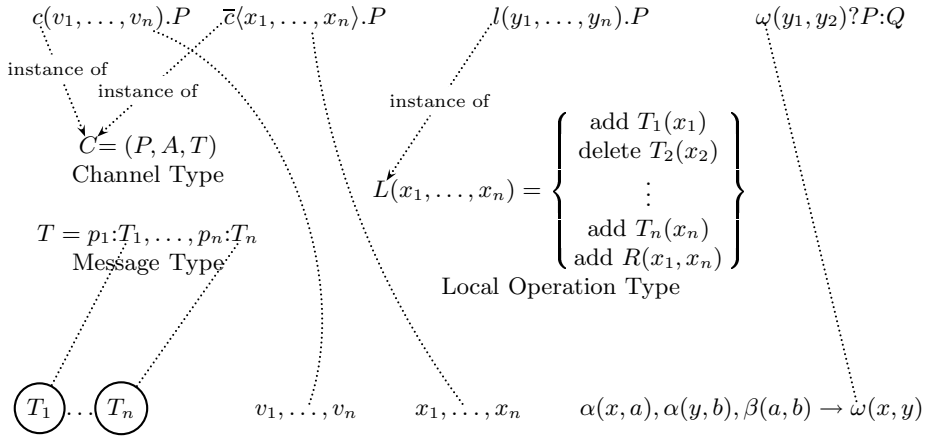


Fig. 2. Connection between Resources (bottom row describes T-Box, A-Box and R-Box) and Behavior (top row shows relevant π -calculus syntax elements)

tool for describing the dynamics of communicating mobile processes. However, π -calculus names, i.e. the objects that are communicated among actors do not have any structure and any semantics. This is because π -calculus is a pure process algebra not designed for reasoning about the meaning of the involved static objects in a process. Consequently, static objects are just considered as strings. In practice however, as in case of Web services, one needs to reason about dynamic behavior and resources at the same time. E.g. one may wish to know whether a Web service sends an order confirmation about the ordered book before it actually delivers the ordered book. To overcome this problem, we need a technique to connect the resource descriptions with the behavioral description. Figure 2 gives an overview of our approach for connecting π -calculus process expressions with DL knowledge bases.

Communicating Meaningful Names In π -calculus agents communicate via exchanging messages over a communication channel. However, π -calculus chan-

nels do not have any structure. In practice however, information about the type of communication protocol and the type of messages that can be transmitted over a channel is very useful. E.g. one may wish to know whether the book selling Web service will send the book via HTTP as PDF or via surface mail as hardcopy. To overcome this deficiency, we introduce channel types.

Definition 2 (Channel Type). *A communication channel type C is a tuple (P, A, T) , where P is a protocol, A an address and T a message type. A protocol can be e.g. “http”, “phone”, “fax”, “surface mail” etc. Each protocol supports a set of MIME types that it can transport. For example, “http” supports “XML” and “HTML” that can not be sent by “surface mail”. An address determines the communication target and its format is dependent on the protocol. For example, if the protocol is “http”, the address is some Web URL and if the protocol is “phone”, the address is a phone number.*

Polyadic π -calculus supports sorts to ensure that the communication only takes place if the arity of the incoming message matches with the expected arity. However, it does not ensure that the communication only takes place if the resources sent by one party are of the type that the receiving party expects.

Definition 3 (Message Type). *A message type T is a set of message parts p_1, \dots, p_n with each part p_i having the type T_i . Each T_i is a DL concept.*

For communication activities (input as well as output activities), we use C_i in place of c , where C is a channel type and i is the unique identifier of the instantiation of C thereby associating c with a channel type and thus a structure. Finally, by modeling channel types as description logic individuals we make sure that channel descriptions can be sent and received just like any other resources and thus the mobility is preserved.

In an input activity $c(v_1, \dots, v_n)$, v_1, \dots, v_n are variables. We model variables as DL A-Box individuals within the name space of the corresponding actor. In an output activity $\bar{c}(x_1, \dots, x_n)$, x_1, \dots, x_n are resources, which we also model as DL A-Box individuals. In order to bind a value x to a variable v , we can simply add the A-Box individual equality axiom $v = x$ in knowledge base of an reasoner that needs to reason over the functional properties of Web services.

Local Operation for Modeling Updates π -calculus suggests (and provides the necessary expressivity) to model even the simplest tasks like adding two number as processes, yet, modeling processes of practical interest with pure π -calculus syntax is tedious and one obtains unnecessarily long process expressions. Another consequence of modeling everything as processes is that one cannot support support black box views which are important in practice.

To overcome this problem, we introduce local operation types. A local operation is a decidable procedure that can add new DL axioms in the knowledge base or remove existing DL axioms from the knowledge base of the agent that executes the local operation. So, we model a local operation as $l(x_1, \dots, x_n)$ and its effects as a list of changes Δ , where each change $\delta \in \Delta$ is a parameterized DL

A-Box axiom. Furthermore, a change δ is adorned with “add” or “delete” which indicates whether the axiom corresponding to δ is added to or removed from the knowledge base. Every parameter of an axiom corresponding to a change belong to the set $\{x_1, \dots, x_n\}$. For example, if $A = \{classMember(x_1, x_2)\}$ belongs to the effects of an operation $l(x_1, x_2)$, executing l with arguments *Peter* and *Person* will add the axiom $classMember(Peter, Person)$ in the knowledge base. A concrete invocation of a local operation L_i is an instance of the local operation type L with appropriate parameters.

Conditions with DL-safe Rules π -calculus offers the notion $\omega?Q:R$ for conditional branching. Such process checks the condition ω and if it is true, it behaves like the process Q otherwise like the process R . However, since π -calculus names do not have any structure, the only condition that can be checked is equality. In practice however, one needs to check richer conditions, e.g. whether the income of person x is higher than the income of person y .

To overcome this deficiency, we make use of DL-safe rules extension[4]. We do so by using the name of an n -ary predicate in the head of a DL-safe rule as the condition in the process expression $\omega?Q:R$.

2.3 Semantics

The semantics of a π -DL process expression is given by its mapping to a labeled transition system (LTS).

Definition 4 (Labeled Transition System). *A labeled transition system is a tuple (S, A, \rightarrow) , where S denotes a set of states, A a set of actions and $\rightarrow \subseteq S \times A \times S$ a transition relation. We often write $s \xrightarrow{a} t$ for $(s, a, t) \in \rightarrow$.*

We describe a state of such an LTS with a process expression and the set of DL A-Boxes of the actors involved in the overall Web service process. Intuitively, a state describes the current knowledge states of the actors and the actions that an actor can perform. When actors perform actions (input, output or local actions), the A-Boxes of the actors involved in an action may change, bringing the system to a new state.

3 Modeling Trustworthy Credentials

In a large Web service market where millions of Web services are offered, there will be more than one Web service providing the same or similar functionality. This is analogous to having more than one shop where one can buy clothes. So, the credentials of Web services become very important, so that the potential users can decide which of the many alternatives they should choose. This is analogous to the phenomenon that despite having a large offer of clothes shops, most people usually go to their favorite shops because they like the atmosphere in the shop, or they find the salespersons friendly or they go to a shop that has been recommended to them by someone whom they trust in matter of clothes.

Current approaches for describing non-functional properties abstract from the issuer of credentials, which is not practical. Comparing with the clothes shops example, in which every cloth shop obviously advertises that it has great atmosphere and very friendly salespersons etc., if only Web service providers describe their Web services, the credentials of the Web services will be hardly of any practical use. Rather, there is a need for techniques in which parties different from the Web service providers issue credentials to Web services and users can build trust in Web services on the basis of such credentials.

3.1 Simple Public Key Infrastructure (SPKI)

The credential-based public key infrastructure SPKI/SDSI [7,8] allows each principal to issue credentials. Unlike other public key infrastructures, SPKI/SDSI requires no central certification authority. Thus, anyone can issue and trust credentials independently of others and may even define his own trust structure.

Definition 5 (Local and Extended Name). *A local name is a sequence of length two consisting of a public key K followed by a single identifier. Typical local names might be “ K Alice” or “ K project-team”. Here, K represents an actual public key. The local name “ K A” belongs to the local name space of key K . An extended name is a sequence consisting of a key followed by two or more identifiers. Typical extended names might be “ K Alice mother”, “ K microsoft engineering windows project-mgr” or “ K UNIKA personnel-committee”.*

Let \mathcal{N}_L denote the set of all local names and $\mathcal{N}_L(K)$ denote the local name space of key K . The SPKI/SDSI expressions are called “terms”. Intuitively, a term is something that may have a value. In SPKI/SDSI, values are always sets of keys.

Definition 6 (Term). *A term is either a key or a name. Let $\mathcal{T} = \mathcal{K} \cup \mathcal{N}$ denote the set of all terms.*

A name certificate provides a definition of a local name (e.g. K A) belonging to the issuer’s (e.g. K ’s) local name space. Only key K may issue (that is, sign) certificates for names in the local name space $\mathcal{N}_L(K)$. A name certificate C is a signed four-tuple (K, \mathbf{A}, S, V)

- The *issuer* K is a public key; the certificate is signed by K .
- The *identifier* \mathbf{A} (together with the issuer) determines the local name “ K A” that is being defined; this name belongs to the local name space $\mathcal{N}_L(K)$ of key K . It should be noted that name certificates only define local names (with one identifier); extended names are never defined directly, only indirectly.
- The *subject* S is a term in \mathcal{T} . Intuitively, the subject S specifies a new additional meaning for the local name “ K A”.
- The *validity specification* V provides additional information allowing anyone to ascertain if the certificate is currently valid, beyond the obvious verification of the certificate signature. Normally, the validity takes the form of a validity period (t_1, t_2) : the certificate is valid from time t_1 to time t_2 , inclusive.

3.2 Semantic-SPKI

SPKI – though being simple and powerful – has the drawback that the names of the properties are simple strings, which does not allow certification of complex properties, e.g. AIFB-Employee and above25 in a way that one can automatically reason about them. We overcome this problem with Semantic-SPKI. The main idea behind Semantic-SPKI is viewing SPKI names as DL concepts and public keys as DL individuals. This implies that in a name certificate, we use a DL concept expression at the place of the identifier A , which has the advantage that it becomes possible to issue complex properties flexibly and reason about them, since complex properties can be constructed by using DL constructors for building complex concepts. The subject of a name certificate is either a key or a name. In a semantic-SPKI name certificate (K, C, S, V) , if S is a key, we view the name certificate equivalent to the ABox assertion $C(S)$, if it is a name we view it equivalent to the TBox assertion $S \sqsubseteq C$.

Definition 7 (Trust Policy). A trust policy is defined by a DL query $C(x)$, which means an actor x is trusted if it can prove to possess the property C .

Given a trust policy and a set of certificates, finding a chain of certificates that satisfies the trust policy can be done by the so-called certificate chain discovery algorithm [9].

4 Implementation

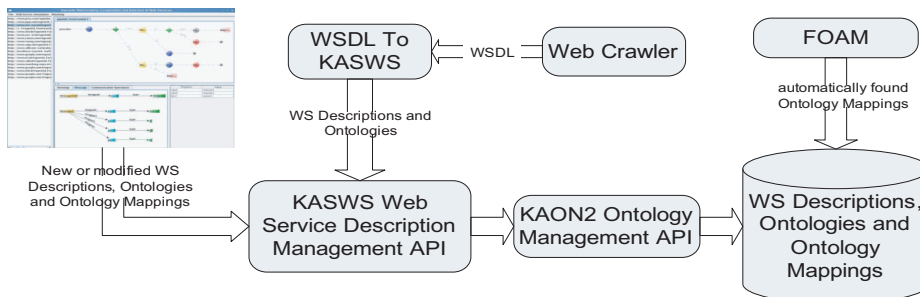


Fig. 3. Architecture of Karlsruhe Semantic Web Service Framework

Figure 3 shows the architecture of our prototypical implementation. In the following, we will discuss each of the components in detail.

4.1 KASWS Web Service Descriptions Management API

The KASWS system needs to maintain a large number of Web service descriptions in our formalism. In general, every Web service description may be associated with a separate ontology and there may be mappings among such ontologies.

So, the system also needs to manage a large set of ontologies. We use KAON2² to manage the repository. KAON2 is an infrastructure for managing OWL-DL ontologies with DL-Safe rules. The KASWS Web Service Management API allows to work with the Web service descriptions in the repository.

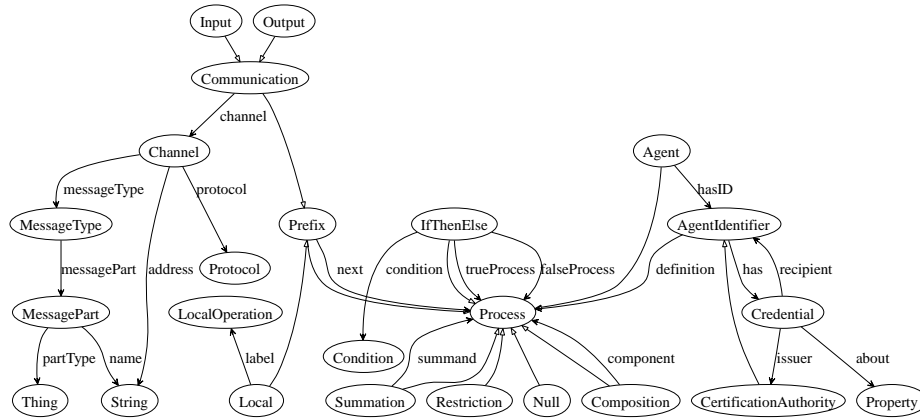


Fig. 4. KASWS Ontology

While the ontologies associated with Web services can be directly managed by KAON2, we developed an extra component to manage the behavioral descriptions of Web services. Although the KAON2 repository together with a repository of behavioral descriptions fulfilled the purpose, it was hard to manage them since they have to be synchronized all the time. Another drawback of having two repositories was that we needed to implement methods for the retrieval of information about the behaviour needed by the reasoning algorithms. The fact that KAON2 can efficiently manage any type of information expressible with OWL-DL and provides efficient query answering, that is retrieval of the information led us to the idea that modeling the behaviour of Web services as an ontology would save us to manage a separate repository. Figure 4 shows the main components of the DL ontology that we developed to able to maintain behavioral descriptions with KAON2.

4.2 Bootstrapping

In order to show the suitability of our Web service description formalism for existing Web services, we developed a tool for converting WSDL documents into descriptions in our formalism automatically. Obviously, an automatically generated description is often not semantically correct. Apart from that, a lot of information that is present inside the `documentation` tag in natural language

² <http://kaon2.semanticweb.org>

is not considered during the conversion. Therefore, automatically generated descriptions have to be manually completed or rectified. Even though, automatic conversion tools like WSDL to KASWS converter are very helpful, since one does not have to describe Web services from scratch, which saves time. Another advantage of such tools is of psychological nature. Most people have less hesitation to rectify information rather than creating new information (cf. CiteSeer³). In the following, we describe how WSDL to KASWS conversion tool works [10].

A WSDL document describes more than one Web service. For each Web service, there is an `Operation` block inside the `portType` block. The XML schema used for describing the Web services is defined in the `types` block. Each `Operation` block contains operation name, input message and output message.

- For each operation in the `portType` block, a new instance of the concept `AgentIdentifier` is created.
- For each message type defined in a `message` block, an instance of the concept `MessageType` is inserted. The order of occurrence of the messages inside the `Operation` block defines the data flow.
- The parts of a message type are listed inside the `message` block. For each part an instance of the concept `MessagePart` is created with the appropriate type.
- For every complex type the XML schema described in the `types` block, a corresponding concept is inserted. For the range of the properties, we do not use primitive data types like `xsd:string`, `xsd:int` etc., since they rather represent the serialization types needed at the time of execution of the Web service and it is not possible to relate them to other concepts of the ontology via subsumption. We rather create concepts from the property names by replacing the first character of a property name by its upper case representation. These concepts are then defined as ranges of the corresponding properties by inserting appropriate axioms in the knowledge base with the help of the KASWS WS Management API.
- Finally, the instance of the `AgentIdentifier` is connected to an instance of the concept `Process` via the property definition. The instance of `Process` is derived from the data flow information inside the `Operation` block.

Since the types used in various WSDL documents are not interconnected and it is a time consuming task to map the large number of ontologies manually, we use the tool FOAM⁴ for detecting simple mappings between the ontologies automatically [11].

4.3 Graphical User Interface

The Graphical User Interface component supports a user to do various tasks.

- It allows to describe a new Web service including the corresponding ontology graphically and save it in the knowledge base.

³ <http://citeseer.ist.psu.edu/>

⁴ <http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/>

- The automatically generated Web service descriptions from WSDL documents have very simple temporal structure due to the missing information in the corresponding documents. Apart from that, the automatically generated description may be semantically incorrect. Therefore, the GUI allows the user to edit existing descriptions and ontologies. Similarly, since the automatically found ontology mappings can be faulty, the GUI allows to manually edit the ontology mappings and add new more expressive ones [12].
- The GUI allows a user to specify his request graphically and send it to the KASWS reasoner. Furthermore, a user can select a Web service from the list of Web services and search for similar ones. The matches are presented graphically to the user.

5 Related Work

OWL-S formerly known as DAML-S [13] is perhaps the first initiative to address the need of describing Web services semantically. However, OWL-S suffers from many problems. Firstly, the OWL-S process model does not have a formal execution semantics. Note, that the execution semantics presented in [14] is of one of the first versions of DAML-S. Since OWL-S is an OWL ontology, OWL-S process model has a description logics semantics. However description logics can not capture behavioral semantics, in particular that of changing A-Boxes of the participating actors. We believe, that this is the major reason for the non-availability of matchmakers based on OWL-S process model. OWL-S matchmaker presented in [13] actually matches the subsumptions among the input and output parameter of a Web service, which is, as discussed in Section 1, not sufficient for automation. OWL-S however has identified and modeled some important non-functional properties of Web services, which we have abstracted from in this paper. We believe, that our formal model can be used to provide formal execution semantics to OWL-S.

WSMO (Web Service Modeling Ontology) provides the conceptual underpinning and a formal language for semantically describing Web services in order to facilitate the automatization of discovering, combining and invoking electronic services over the Web [15]. WSMO is more a formalized bird-eye view than a concrete Web service description language. There are reference implementations like WSMX that technically realize WSMO. WSMO abstracts from internal service logic while defining orchestration as description of how a Web service makes use of other Web services in order to achieve its capability, which is actually contradictory. A Web service discovery approach presented in [16] is very similar to the one presented in [13]. Nevertheless, we believe that our approach might be seen as a reference implementations of WSMO since we support all top level elements of WSMO, namely ontologies, goals, Web services and mediation.

WSDL-S does not provide a formalism for describing Web services semantically. Rather, it extends WSDL by providing extensibility elements to connect semantic descriptions to WSDL documents. So, our approach is complementary to WSDL-S in a sense that the descriptions of Web services with our formalism can be connected to WSDL-S documents.

Perhaps, the work that is closest to our work is [17], in which an approach is presented to characterize Web services with their transition behavior and their impacts on the real world (modeled as relational databases). Our work (though following similar thoughts) is different in some aspects. Firstly, we presented a concrete syntax for modeling the dynamic behavior whereas [17] does not. Secondly, we model the local knowledge bases of the participating actors with decidable description logics, which can be very helpful while proving important properties like decidability, soundness and completeness of any reasoning algorithms for discovery, composition, selection etc. Modeling knowledge bases of the involved actors with description logics is also more suitable since the Web ontology language OWL standardized by W3C is based on description logics.

Finally, to the best of our knowledge, ours is the only work that has addressed the need of access control policies and can model and reason about them together with other aspects of Web services.

6 Conclusion and Outlook

In this paper, we have presented a formalism for describing Web services semantically. We differentiated between functional and non-functional properties of Web services. Resources and behavior constitute the functional properties whereas the quality of service attributes build the non-functional properties. We have presented a novel combination of π -calculus and description logics to model the functional properties of Web services. We argued that it is not practical to abstract from the issuer of non-functional properties and hence they must be modeled in a way such that users can build their trust in them. We borrowed ideas from the field of security in distributed systems and proposed to use certificates for modeling non-functional properties. We showed how SPKI/SDSI certificates can be augmented with semantics by annotating the properties with description logic concepts.

Semantic description of Web services is a necessary step to enable automated reasoning about Web service descriptions. In this paper, our focus was on the description formalism and the development of algorithms for discovery, composition, selection etc. that make use of such rich descriptions is still an open and interesting research issue. In [18], we have presented a preliminary version of model-checking based matchmaking.

References

1. Agarwal, S., Studer, R.: Automatic Matchmaking of Web Services. In Zhang, L.J., ed.: IEEE 4th International Conference on Web Services, Chicago, USA, IEEE Computer Society (September 2006) 45–54
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory Implementation and Applications. Cambridge University Press (2003)

3. Horrocks, I., Patel-Schneider, P.F.: A Proposal for an OWL Rules Language. In: Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), ACM (2004) 723–731
4. Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with Rules. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: Proc. of the 3rd. Int. Semantic Web Conference (ISWC 2004). Volume 3298 of LNCS., Hiroshima, Japan, Springer (November 2004)
5. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I+II. *Journal of Information and Computation* (September 1992) 1–87
6. Sangiorgi, D., Walker, D.: *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA (2001)
7. Ellison, C.M., Frantz, B., Lampson, B., Rivest, R.L., Thomas, B.M., Ylonen, T.: Simple public key certificate. <http://world.std.com/cme/html/spki.html> (July 1999)
8. Ellison, C.M., Frantz, B., Lampson, B., Rivest, R.L., Thomas, B.M., Ylonen, T.: SPKI certificate theory. Internet RFC 2693 (September 1999)
9. Clarke, D.E., Elien, J.E., Ellison, C.M., Fredette, M., Morcos, A., Rivest, R.L.: Certificate Chain Discovery In SPKI/SDSI. *Journal of Computer Security* **9** (2001) 285–322
10. Bai, T.: Automatische Extraktion von Semantischen Beschreibungen von Web Services (Automatic Extraction of Semantic Description of Web Services). Student Research Project supervised by Sudhir Agarwal and Rudi Studer (October 2006)
11. Ehrig, M., Staab, S., Sure, Y.: Bootstrapping Ontology Alignment Methods with APFEL. In Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A., eds.: Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005. Volume 3729 of LNCS., Springer (November 2005) 186–200
12. Haase, P., Motik, B.: A Mapping System for the Integration of OWL-DL Ontologies. In Hahn, A., Abels, S., Haak, L., eds.: IHIS 05: Proc.s of the 1st Int. Workshop on Interoperability of Heterogeneous Information Systems, ACM Press (NOV 2005) 9–16
13. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics* **1**(1) (December 2003) 27–46
14. Ankolekar, A., Huch, F., Sycara, K.: Concurrent Execution Semantics for DAML-S with Subtypes. In Horrocks, I., Hendler, J.A., eds.: Proceedings of the First International Semantic Web Conference: The Semantic Web (ISWC 2002). Volume 2342 of Lecture Notes in Computer Science (LNCS)., Sardinia, Italy, Springer (2002) 14–21
15. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Pollers, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. *Applied Ontology* **1**(1) (2005) 77–106
16. Keller, U., Lara, R., Pollers, A., Toma, I., Kifer, M., Fensel, D.: WSMO Web Service Discovery (November 2004) <http://www.wsmo.org/TR/d5/d5.1/v0.1>.
17. Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic Composition Of Transition-Based Semantic Web Services With Messaging. In Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Per-ke Larson, B.C.O., eds.: VLDB '05: Proceedings of the 31st international conference on Very large data bases, Trondheim, Norway, ACM (August-September 2005) 613–624
18. Agarwal, S.: Model checking expressive web service descriptions. In: IEEE 5th International Conference on Web Services, Salt Lake City, Utah, USA, IEEE Computer Society (July 2007)