Semantic Description of Distributed Business Processes

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

Andreas Abecker
Research Center for
Information Technologies (FZI),
Karlsruhe, Germany.
abecker@fzi.de

Abstract

Today, more and more business processes are distributed in nature, involving business processes of other organizations that are exposed as services. Since a business process can be very complex in general, there is a need for automated support for checking whether a business process complies to organization's policies. On the other hand, the fast growing market of services and the need for continuously improving the business processes to cope with the competition, automated methods are required to find appropriate services and compose business processes. Fulfilling these requirements is a challenging task needing formalisms for describing business processes and services, specifying organization's policies and algorithms for discovery of services as well as verification and composition of business processes.

In this paper, we present formalisms for describing executable distributed business processes and for describing functional and non-functional properties of services. The novelty of the formalism for describing distributed business processes lies in the combination of the polyadic π-calculus and the description logic $\mathcal{SHOIN}(\mathbf{D})$ with DL-Safe rules. When the functionality of business processes are exposed as services, the quality of service attributes and access control policies need to be described in addition to the functional properties. For this purpose, we introduce semantic-SPKI/SDSI certificates and credential based access control. Hence, our formalism allows modeling of dynamic behavior including credential based access control along with the involved resources (information or real world objects) and quality attributes of business processes in a unifying way.

1 Introduction

Today, more and more business processes are collaborative business processes, involving many parties and running in a distributed environment. Organizations need to perform various tasks in order to achieve a more flexible and comprehensible management of their business processes. Figure 1 shows the major business process management tasks.

Analysis In the analysis phase domain experts and managers identify and define requirements for a new business process. These requirements determine the desired functionality of the business process considering any policies of the
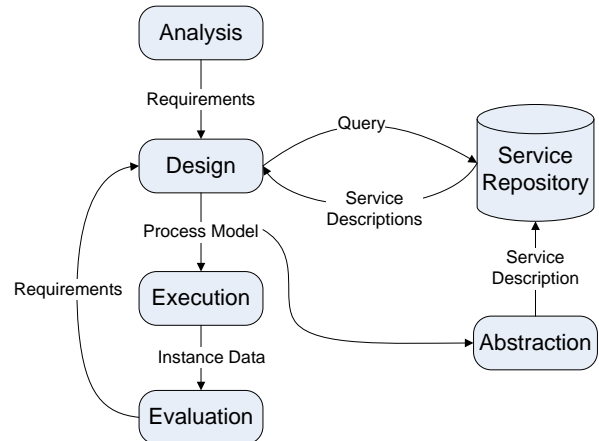
Figure 1: BPM Lifecycle

organization. Examples of such policies could be 'whenever something is purchased, payment should be done after the purchased good is delivered' or 'whenever the price of a good to be purchased is more than 1000 Euro, department manager must approve the purchase, before the order is placed'.

Design In this phase, engineers design the executable business process by considering all the requirements identified in the analysis phase.

Before a business process is executed, it needs to be checked, whether it behaves as desired and complies to organization policies. In order to enable computer support during this task, the requirements and the executable processes need to be specified formally such that automatic verification algorithms can be developed for them. For specifying requirements, typically modal logics are used. The field of temporal logics provides a plethora of candidates for this. In fact, in the approach presented in (Agarwal 2007b; 2007a), an extension of the μ-calculus is proposed. With such a formalism at hand, model checking techniques can be applied to automatically verify whether a process description has desired properties or not.

Designing an executable process from existing services and business processes can be seen as *composition*. While

composing a business process, the engineers consider which parts of the business process can be performed by business processes of other organizations exposed as services, which parts are covered by already designed processes of the organization and which parts need to be implemented new. For finding appropriate services for some part of the business process, engineers need to formulate an appropriate query and send it to a repository of services descriptions, which in turn sends back the matching service descriptions. As in case of verification, in order to enable computer support for composition, one needs languages for specifying requirements, describing services as well automatic discovery and composition algorithms for these languages.

Abstraction A distributed business process uses business process of other organizations exposed as services. Descriptions of such services are published such that other organizations can find and bind them in their business processes. In this task, a service description is created from a business process description by abstracting from details that the provider may not wish to disclose or users are not interested in.

Execution In this phase, a distributed business process is executed. During the execution, the execution environment interacts with the execution environments of other organizations if their services are embedded in the business process. That is it invokes the services as specified in the process description. Typically, a business process is designed to be executed more than once. A particular execution of a process can be termed as a *run* of the process. During the execution phase, information about the particular run, *instance data* is also stored.

Evaluation This phase consists of reasoning over the process runs. From the insights that one gains from process runs, one can derive new requirements and feed into the design stage to modify the process. In this phase, it is checked whether at process instance stage whether a running process complies with internal policies or the higher-order properties it has been annotated with.

In order to enable computer support for above mentioned management tasks, business processes, their instances (runs) and services must be described formally. The formal description of business processes must cover behavior, involved resources and changes in the resources of various actors involved in the process. Whereas services should be described in a way, such that (1) their descriptions are compatible with the process description language so that they can be embedded in a process. (2) users are able to reason about their quality attributes and (3) user are able to check whether they have access to a service or not. 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 (Agarwal & Studer 2006).

The paper is structured as follows. In Section 2, we present a language to model the dynamic behavior of a dis-
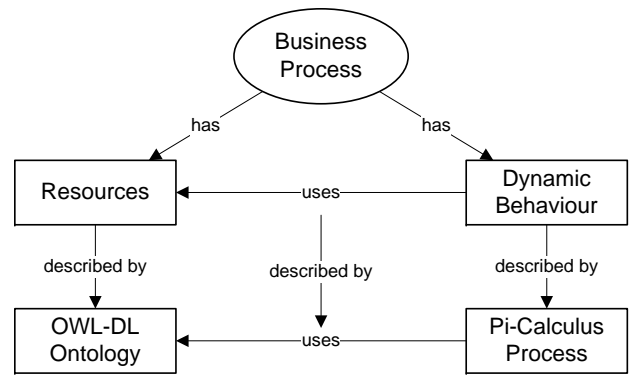

Figure 2: Our Business Process Modeling Approach

tributed business process, involved resources and changes in the resources its execution in a unified way and with formal semantics. The formalism is a novel combination of the process algebra polyadic π-calculus and description logics $\mathcal{SHOIN}(\mathbf{D})$ with DL-Safe rules. In Section 3, we turn our attention to modeling services. When business processes are offered as services, the quality of service attributes and access control policies become important. We present Semantic-SPKI, our semantic extension to the Simple Public Key Infrastructure (SPKI) and show how provable non-functional properties can be modeled as Semantic-SPKI certificates and how credential based access control can be integrated in behavioral description of a service. In Section 4, we present our prototype for modeling and managing descriptions business processes and services. The prototype mainly consists of a graphical user interface, an API and a repository. Finally, we conclude in Section 6 after discussing some related work in Section 5.

2 Modeling Executable Business Processes

The main focus of the existing business process description languages is the execution of the processes. Execution, though certainly important, is one of the later phases of the whole business process life cycle. In order to support other stages, e.g. verification and composition, business processes need to be described with formal semantics. Many of the existing approaches either do not have formal semantics or only the formal semantics for the dynamic behavior of the process. In practice however, it is equally important to be able to reason about the involved resources. Mere reasoning about the dynamic behavior of the processes is not enough.

In general, a distributed business process involves several actors that communicate with each other by exchanging messages. The messages contain resources that can be real world objects or information objects.

Figure 2 shows the main idea behind our business process modeling formalism. We consider resources and dynamic behavior of business processes. We model resources as $\mathcal{SHOIN}(\mathbf{D})$ ontologies with DL-safe rules and dynamic as behavior as π-calculus process. Our main contribution regarding modeling of functional properties is the establishment of the connection between process descriptions and on-

tologies.

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 (Baader *et al.* 2003).

2.1 Modeling Resources

Short Introduction to $\mathcal{SHOIN}(\mathbf{D})$ 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,...), roles (R, S,...), and individuals (a, b,...), $\mathcal{SHOIN}(\mathbf{D})$ 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* $\{a_1, \ldots, a_n\}$. 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 (Baader *et al.* 2003; Horrocks & Patel-Schneider 2004; Motik, Sattler, & Studer 2004). A decidable rule extension of $\mathcal{SHOIN}(\mathbf{D})$, socalled DL-safe rules was presented in (Motik, Sattler, & Studer 2004).

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. $\mathcal{SHOIN}(\mathbf{D})$, which the decidable variant OWL-DL of the Web ontology language OWL[1] is also based on. The resources can be further classified into sets that can be hierarchically ordered according to the subset relation-

ship. Again, expressive description logics provide the "$\sqsubseteq$" 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 behavior of a business process. While the λ-calculus formed the foundation for many computer science related topics like programming languages, the description of workflows required a different approach. In a typical workflow tasks are not only executed in sequential order, rather tasks are executed in parallel by different employees to speed up the processing. These different – then again sequential – processing paths have to be created and joined at some points in the business process. Even further, parallel processing tasks could depend on each other. The optimization of business processes usually adds parallelism and dependencies as this is an effective way to reduce the throughput time.

These kinds of parallel processes are hard to describe in terms of the λ-calculus. To overcome the limitations of sequential systems, an approach to represent parallel systems called Petri nets has been adapted for workflow representation. Petri nets have a simple but yet powerful mathematical foundation as well as a strong visual representation. They use the concept of an explicit state representation for parallel systems. Each Petri net is always in a precisely defined state denoted by the distribution of tokens over places contained in the net. The state of the system could then be changed by firing transitions which relocate the token distribution over the places. Petri nets have been adapted by many systems that are used in the business process management domain to describe business processes.

Beside the advantages of Petri nets for the business process management domain, that include strong visualization capabilities, mathematical foundations, as well as their main purpose, the description of parallel systems, Petri nets also have some drawbacks. The main drawbacks are the static structure of the nets (that do not support dynamic process structures) as well as the missing capabilities for advanced composition as for instance recursion. Of course, Petri have been extended with support for dynamic structure, like self modifying Petri nets, recursion, and objects. However, these enhancements also complicate the theory of the nets and thus have reached restricted usage only. A broad research on the capabilities of Petri nets regarding common patterns of behavior found in business processes showed that they fulfill basic tasks like splitting and merging process paths easily, while they fail at advanced patterns like multiple instances of a task with dynamic boundaries. Whereas there exist approaches to overcome some or all of the limitations regarding the behavior, the static structure and limited composition of Petri nets remains (van der Aalst & van Kees 2002).

To overcome the limitations of Petri nets, theories of mobile systems have been developed. Thereby a mobile system is made up of entities that move in a certain space. The space consists of processes, and the entities that move are either links between the processes (link passing mobility) or the

processes themselves (process passing mobility). A theory for mobile systems, the π-calculus, overcomes the limitations of Petri nets regarding the static structure and limited composition capabilities at the cost of a more complex representation. The π-calculus represents mobility by directly expressing movements of links in an abstract space of linked processes (i.e. link passing mobility). Practical examples are hypertext links that can be created, passed around, and disappear. The π-calculus does not, however, support another kind of mobility that represents the movement of processes. An example is code that is sent across a network and executed at its destination. The π-calculus uses the concept of names with a certain scope for interaction between different parallel processes. Names are a collective term for concepts like channels, links, pointers, and so on. As the mobile system evolves, names are communicated between processes and extrude or intrude their scope regarding to certain processes. As the synchronization between processes is based on interaction and received names are also used as communication channels, the link structure is changed dynamically all the time the mobile system evolves.

Our language for modeling behavior of distributed business processes is based on π-calculusȘo we first introduce it briefly and refer to (Milner, Parrow, & Walker 1992; Sangiorgi & Walker 2001) for more details.

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

$$P \quad ::= \quad \mathbf{0} \mid y[v_1 \ldots, v_n].P \mid y\langle x_1 \ldots, x_n\rangle.P \mid$$
$$\tau.P \mid [x = y]P \mid P_1 \parallel P_2 \mid P_1 + P_2 \mid$$
$$@A\{y_1, \ldots, y_n\}$$

The syntax of π-calculus is defined recursively. *Null process* $\mathbf{0}$ is a process that does nothing. This process is used to denote the termination of a process. *Input process* $y[v_1, \ldots, v_n].P$ is a process that inputs arbitrary names $z_1, \ldots, z_n$ at port y, binds them to names $v_1 \ldots, v_n$ and then behaves like the process $P\{z_i/v_i\}$, where z_i/v_i denotes substituting z_i for v_i. The names $v_1, \ldots, v_n$ are bound by the input process $y[v_1, \ldots, v_n].P$. *Output process* $y\langle x_1, \ldots, x_n\rangle.P$ is a process that outputs the names $x_1 \ldots, x_n$ at port y and then behaves like the process P. The *Silent* process $\tau.P$ performs the silent action τ and then behaves like the process P. *Match* process $[x = y]P$ is a process that compares two names x and y for equality and behaves like the process P, if x and y are equal or like $\mathbf{0}$ if they are not equal. *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* $P_1 + P_2$ denotes the non-deterministic choice and behaves either like P_1 or like P_2. In contrast to deterministic choice, in which a process evolves depending on the truth value of some condition, e.g. equality, in case of non-deterministic choice, the user of the process selects one of the alternatives and the selected alternative is executed. In π-calculus process expressions can be given a name, such that complex process expressions do need to be defined every time they are used but embedded by invoking the named process expression. This is similar to defining a method in a programming language and invoke the method in other methods instead of copying the body of the method everywhere it should be used. The named process expression is called *Agent Identifier*. For any agent identifier A (with arity n), there must be a unique defining equation $A(x_1, \ldots, x_n) \stackrel{def}{=} P$, where the names $x_1, \ldots, x_n$ are distinct and are the only names which may occur unbound in P. Now, the process *Agent* $@A\{y_1 \ldots, y_n\}$ behaves like $P\{y_1/x_1, \ldots, y_n/x_n\}$. Note that defining equations provide recursion, since P may contain any agent identifier, even A itself.

Connecting the Ontology with Behavior Polyadic π-calculus is a powerful 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 business processes, one needs to reason about dynamic behavior and resources at the same time. E.g. one may wish to know whether a business process 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.

We model a business process as an agent identifier. As described above, the names $x_1, \ldots, x_n$ in the definition of the agent identifier $A(x_1, \ldots, x_n)$ are the only names that may occur unbound in the defining process expression. While defining a business process as an agent identifier, the names $x_1, \ldots, x_n$ correspond to the elements of the ontology associated with the business process. In order to simplify the modeling, our agent identifier expressions only contain one parameter, which denotes the URI of the associated ontology instead of a list of all the elements of the ontology.

Communication Channels In π-calculus agents communicate via exchanging messages over a communication channel. However, π-calculus channels 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 business process 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 $p_1, \ldots, p_n$ 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 y, where C is a channel type and i is the unique identifier of the instantiation of C thereby associating y 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 process expression $y[v_1, \ldots, v_n].P$, $v_1, \ldots, v_n$ are variables. We model variables as DL A-Box individuals within the name space of the corresponding actor. The reason for doing this is that variable v can be bound to a value α (which is again an A-Box individual) by adding an A-Box individual equality axiom $v = x$ in the knowledge base. Once, we have such an axiom in the knowledge base, the variable v can be used just as a value as in case of programming languages. In an output process $y\langle x_1, \ldots, x_n \rangle.P$, $x_1, \ldots, x_n$ are resources, which are also modeled as DL A-Box individuals and are available since all the individual names of the associated ontology can occur freely in the process expression.

Local Operations π-calculus suggests and provides the necessary expressivity to model even the simplest tasks like adding two number as processes. Thus 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 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 updates the A-Box of the agent that executes the local operation. A local operation may perform a query on the local knowledge base or some calculation to create add new individuals and add corresponding axioms in the knowledge base to relate the individuals with each other. It can also remove existing DL axioms from the knowledge base. So, we define a local operation type $L(x_1, \ldots, x_n)$ as as a list of change types Δ, where each change type $\delta \in \Delta$ is a parameterized proposition. Furthermore, a change type δ is adorned with "+" or "-" which indicates whether the proposition corresponding to δ is added to or removed from the knowledge base. For example, if

the change type $\{+classMember(x_1, x_2)\}$ belongs to the change types of a local operation type $L(x_1, x_2)$, executing L with arguments $Peter$ and $Person$ will add the axiom $classMember(Peter, Person)$ in the knowledge base. A concrete invocation $l = L_i$ of a local operation type L is the i-th instance of the local operation type L with appropriate parameters.

Deterministic Choice The process expression of type match, $[x = y]P$ of the pure π-calculus supports only equality check of two names x and y. This is mainly due to the fact that the pure π-calculus abstract from the structure of the names. 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. Another problem with the match expression is that it does not allow to specify the process that should be executed in case the condition is not true. To overcome these problems, we introduce the process expression $\omega?P:Q$ that behaves like P if the condition ω is true, and otherwise like Q.

The condition ω can be any predicate in the ontology of actor that checks the condition. These include concept names, relation names defined in the T-Box, rule heads of DL-safe rules in the R-Box of the ontology and any predicate symbols, the implementations of which lie outside the description logic reasoner. This allows to model very expressive conditions the check for whose truth value is still decidable.

2.3 Semantics

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 (Milner, Parrow, & Walker 1992). In the following, we present the semantics of our formalism that we call π-DL by defining mappings from its process expressions to a labeled transition system (LTS) in a similar fashion.

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 $(s, a, t) \in \rightarrow$.*

As we have mentioned earlier, a business process can be in general a complex process involving many actors. In our model, we consider the behaviors and the knowledge bases of the involved actors. So, we described a state of an LTS with a set of π-calculus process expressions and the set of DL A-Boxes of the actors involved in the business process. Intuitively, a state describes the current snapshot of the knowledge bases of the actors and the actions that the actors can perform in the state. 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.

We now give the formal semantics of our formalism for modeling functional properties of business processes. The semantics specifies how the behaviors of the involved actors in a business process evolve by performing actions. Note,

that in our approach every actor corresponds to a π-calculus agent identifier with its behavior corresponding to the defining process expression of the agent identifier.

Local Operations A process $P = l(x_1, \ldots, x_n).Q$ can evolve to the process Q without any preconditions. This means that an actor that can perform a local operation in current state, can perform the local operation independent of what other actor can do.

$$\frac{-}{l(x_1, \ldots, x_n).Q \overset{l(x_1,\ldots,x_n)}{\rightarrow} Q}$$

Communication Operation Two processes can communicate with other only if they are running in parallel and one of them performs an input activity and the other an output activity. Note, that the processes running in parallel may belong to the same actor or to different actors. Furthermore, the type of data that is sent by the output process should be a sub type of the data type expected by the input process. Since, we model the involved data with DL ontologies and the data types as ontology concepts, it corresponds to the sub-concept relationship between the concepts.

$$\frac{P \overset{\overline{x}y}{\rightarrow} P' \quad Q \overset{x(z)}{\rightarrow} Q' \quad \mathsf{typeOf}(y) \sqsubseteq \mathsf{typeOf}(z)}{P \parallel Q \overset{\tau}{\rightarrow} P' \parallel Q'\{y/z\}}$$

Deterministic Choice A process with deterministic conditional branching can evolve to any of the successors, depending on whether the condition is true of false.

$$\frac{\omega?P':P'' \overset{\alpha}{\rightarrow} P' \quad \omega = true}{\omega?P':P'' \overset{\alpha}{\rightarrow} P'} \qquad \frac{\omega?P':P'' \overset{\alpha}{\rightarrow} P'' \quad \omega = false}{\omega?P':P'' \overset{\alpha}{\rightarrow} P''}$$

Composition The parallel composition of two processes can evolve to the any of the successors of the two processes.

$$\frac{P \overset{\alpha}{\rightarrow} P'}{P \parallel Q \overset{\alpha}{\rightarrow} P' \parallel Q} \qquad \frac{Q \overset{\alpha}{\rightarrow} Q'}{P \parallel Q \overset{\alpha}{\rightarrow} P \parallel Q'}$$

Non-Deterministic Choice A process with non-deterministic choice evolves to either one or the other choice.

$$\frac{P \overset{\alpha}{\rightarrow} P'}{P + Q \overset{\alpha}{\rightarrow} P'} \qquad \frac{Q \overset{\alpha}{\rightarrow} Q'}{P + Q \overset{\alpha}{\rightarrow} Q'}$$

3 Modeling Services

In the previous section, we have seen how executable business processes can be modeled. As we have mentioned in Section 1, organizations offer their business processes to other organizations as services. Services are interfaces between the business processes of different organizations and can be seen as means to access the corresponding business processes. Web services can be seen as a special case of services, that can be invoked by using standard Web protocols like HTTP. A Organization uses a business process of another organization by embedding the corresponding service offered by the latter organization.

In order to embed a service in an executable process, the communication pattern the service offers must match with the communication pattern the business process expects. Furthermore, the types and properties of the resources that are exchanged between the service and the business process must have the desired characteristics. The properties of a service on the basis of which it can be decided whether it offers the desired functionality and can be integrated in a business process as *functional properties*.

In a large service market where millions of services are offered, there will be more than one service providing the same functionality. This is analogous to having more than one shop where one can buy clothes. So, the credentials of 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. We call such properties *non-functional properties*. 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 service providers describe their services, the description of the credentials of the services will be hardly of any practical use. Rather, there is a need for techniques in which parties different from the providers issue credentials to services and users can build trust in services on the basis of such credentials.

As we have mentioned above, services are basically means for accessing a business process. For technical, economical and legal reasons, the access to a business process of an organization needs to be restricted. So, describing access control policies is an important part of the service description, so that potential users can automatically check, whether they have access to a service or not.

3.1 Modeling Non-Functional Properties

Simple Public Key Infrastructure (SPKI) The credential-based public key infrastructure SPKI/SDSI (Ellison *et al.* 1999a; 1999b) 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 me "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, A, S, V)

- The *issuer* K is a public key; the certificate is signed by K.

- The *identifier* 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.

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 by viewing SPKI names as DL concept descriptions and public keys as DL individuals. This means that in a name certificate, we use a DL concept expression at the place of the identifier A. This makes it possible to issue complex properties, 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 $S \sqsubseteq C$.

Modeling Non-Functional Properties with Semantic-SPKI Having seen, how actors can certify expressive properties to other actors in a distributed environment with semantic-SPKI, it is rather straightforward to use this technique to model non-functional properties of services. We only need to assign a public key with each service and each actor that wishes to act as a certification authority. Actors can then issue certificates to services certifying them quality of service properties. The set of certificates issued to a service represents the non-functional properties of the service. Other actors can build their trust in a service on the basis of its non-functional properties.

Definition 7 (Trust Policy) *A trust policy P is either a public key or a concept expression. If it is a public key only service with public key $K = P$ satisfies the trust policy P. If it is a concept expression, service with public key $K \in P$ satisfies the trust policy.*

Given a trust policy and a set of credentials, finding a chain of certificates that satisfies the trust policy can be done by the so called certificate chain discovery algorithm (Clarke *et al.* 2001).

3.2 Modeling Functional Properties and Access Control Policies

A service is essentially a process that mediates between a client business process and the provider business process. The business process of the provider can be seen as the functionality that the service offers. Thus, the functional properties of a service can be modeled by the formalism presented in Section 2.

We model checking of access eligibility as a condition by using a predicate symbol $CCD(C, P)$, that is true iff the set C of certificates fulfills the trust policy P according to the certificate chain discovery algorithm (Clarke *et al.* 2001).

Recall the process expression $\omega?P{:}Q$ for deterministic choice from Section 2. In such an expression, ω can be any n-ary predicate. In order to support credential based access control, we simply use the predicate CCD in place of ω in the process expression for deterministic choice. According to the semantics of deterministic choice, a process $CCD(C, P)?Q{:}R$ means that the process evolves to Q if the actor mentioned in the policy P can be trusted, otherwise the process evolves to R.
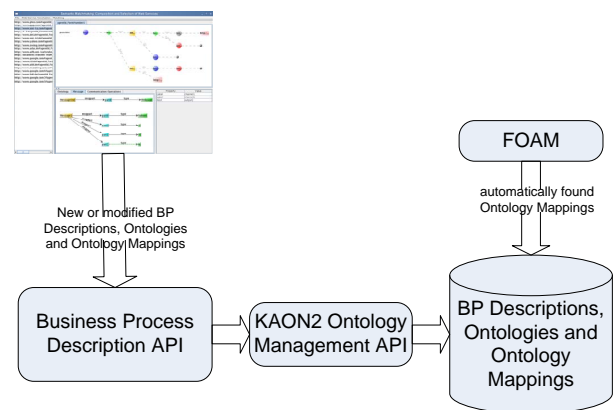
4 Implementation

Figure 3: Architecture of Karlsruhe Business Process Modeling Framework

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

4.1 Business Process Ontology

The prototype needs to maintain a large number of business process descriptions in our formalism. In general, every business process 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 KAON2[2] to manage the repository. KAON2 is an infrastructure for managing OWL-DL ontologies with DL-Safe rules. The Business Process Descriptions API allows to work with the business process descriptions in the repository.

While the domain ontologies associated with business processes can be directly managed by KAON2, we developed an extra component to manage the behavioral descriptions of business processes. 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 behavior 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 behavior of business processes 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 maintain descriptions of business processes and services with KAON2.

The concept AgentIdentifier represents a named process. An executable business process, BusinessProcess and a service Service are special type of processes. An AgentIdentifier has an ontology of type Ontology that represents its knowledge base. The behavior of an AgentIdentifier is represented by the property definition with range Process. The behavior of an AgentIdentifier is further defined by describing the π-calculus syntax with concepts Null, Prefix, Composition, Summation, and IfThenElse. The sequential process (Prefix) can either contain a communication or a local operation followed by another process modeled via the property next. The communication operation is modeled with the concept Communication and a local operation is modeled with the concept Local. A communication operation can either be an input or an output operation, modeled as sub-concepts Input and Output of the concept Communication. Communication takes place over a channel (Channel), that has a message type (MessageType), a protocol (Protocol) and an address of type String. A message type consists of many parts, called message parts messageParts. A type of a message part is a concept from the domain ontology and hence the process description is connected to the domain ontology via message parts. A condition in the IfThenElse expression is any predicate the truth

value of which can be calculated by the reasoner local to the agent. A Service has non-functional properties represented by the property has with range Credential. A credential is described by three properties, namely issuer, recipient and about, that show to the issuing agent, receiving agent and the certified property respectively.

4.2 Business Process Description API

While the above described business process and service ontology defines the structure of business process descriptions modeled as instances, description logics and consequently description logic modeling tools like KOAN2, do not provide any mechanism for enforcing the structure on the ABox instances. To overcome this problem, we have developed the business process modeling API, a Java API for modeling the business processes. The API roughly contains

- A Java class `RepositoryFactory` for creating a new repository or loading an existing repository of business process descriptions. A repository is an OWL ontologies that contains instances representing the business process descriptions. For this purpose the class `RespositoryFactory` provides methods `createRepository` and `openRepository` with parameters `String physicalURI` and `String logicalURI` and return type `Repository`.

- The Java class `Repository` contains methods for creating and retrieving process descriptions. For this purpose, it contains methods `createX` and `getAllX` for each concept X in the process description ontology with return types X and `Set<X>` respectively. The `createX` and `getAllX` methods call the appropriate KAON2 ontology management API methods in order to create and retrieve instances of corresponding concepts.

- A Java interface X for each concept X defined in the business process and service ontology. Each such interface has `getP` and `setP` methods for each property P the concept X is a domain concept for. The return type of a `getP` method is Y if P is a functional property with range concept Y, otherwise the return type is `Set<Y>`. Similarly, the parameter of a `setP` method is Y or `Set<Y>`. The implementations of the interfaces call the appropriate KAON2 ontology management API methods to retrieve and set the property instances.

The business process descriptions API encapsulates the KAON2 domain independent ontology management API and thus provides a domain specific Java API which the software engineers can work with as they are used.

4.3 Graphical User Interface

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

- It allows to describe a new business process including the corresponding domain ontology graphically and save it in the knowledge base.

- It allows to load an existing business process description along with the domain ontology and modify or delete it.
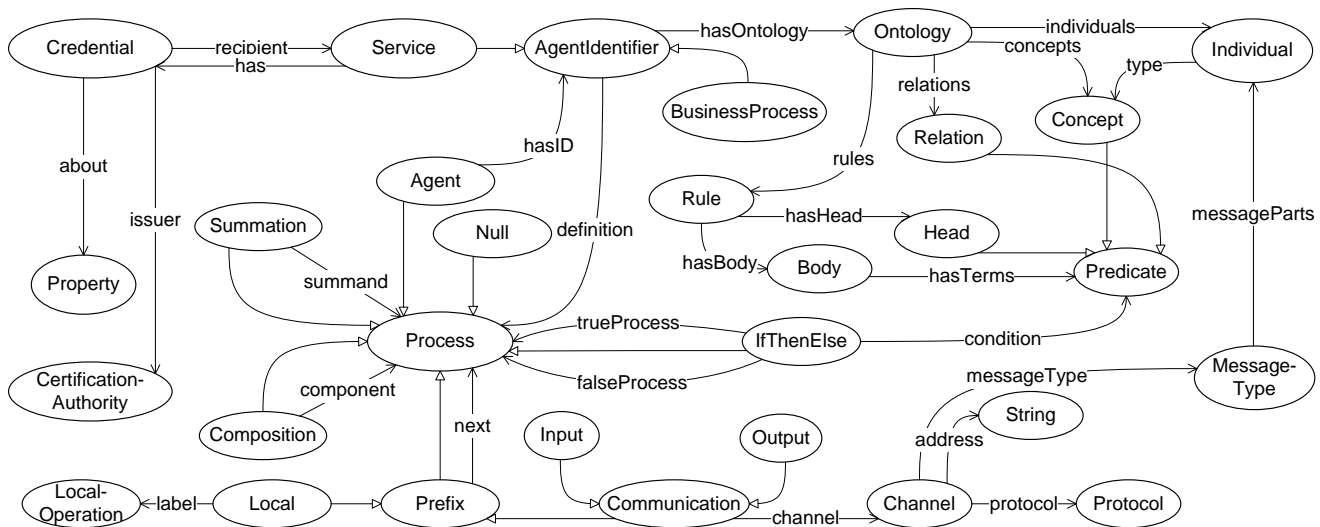
Figure 4: Karlsruhe Business Process and Service Modeling Ontology

- Since the ontology mappings found automatically by FOAM can be semantically faulty, the GUI allows to manually edit the ontology mappings and add new more expressive ones (Haase & Motik 2005).

5 Related Work

There are quite a few approaches to describe business processes and services declaratively. The earlier approaches focus on the execution of business processes inside an organization and are best known as workflow systems. Later the workflow systems has been extended to so-called inter-organizational workflow systems, that can describe processes involving across organizational boundaries.

BPEL4WS (Andrew *et al.* 2003) is a popular formalism for modeling business processes in the Web. It combines XLANG and WSFL. However, it still lacks formal semantics and reasoning procedures. Therefore, our work is complementary to BPEL4WS as our formal model may be used in its extended form to specify formal semantics for BPEL4WS, which is needed in order to prove certain properties of reasoning algorithms based on BPEL4WS. Note, that all the attempts to define formal semantics of BPEL4WS cover only the dynamic behavior of BPEL4WS. Resource schemas that are an essential part of multi party business processes have not been considered. Furthermore, BPEL4WS views Web services as black boxes with one input and one output activity. In our model, we do not have this restriction. In our view, services can have multiple interactions with the user. BPML[3] is similar to BPEL4WS in the sense that it focuses more on the execution of a business process than on reasoning about properties of a process.

OWL-S formerly known as DAML-S (Sycara *et al.* 2003) 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 (Ankolekar, Huch, & Sycara 2002) 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 (Sycara *et al.* 2003) actually matches the subsumptions among the input and output parameter of a Web service, which is, not sufficient for automation. OWL-S however has identified and modeled some important non-functional properties of business processes. We believe, that our formal model can be used to provide formal execution semantics to OWL-S, since the OWL-S process model seems similar to our business process ontology. Furthermore, OWL-S though addresses the need for pre- and postconditions of a Web service, it does not provide with any concrete formalism for modeling them. We have shown how preconditions and effects can be modeled with the help of deterministic choice and local operations respectively.

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 (Roman *et al.* 2005). WSMO is more a formalized bird-eye view than a concrete Web service description language. There are reference implementations like WSML that technically realize WSMO. Our work is complementary to WSML in many ways. Firstly, WSML does not support the description of behavior Web services. Secondly, it does not provide with a concrete technique to model non-functional properties and access control policies. Furthermore, similar to OWL-S though WSMO addresses the issue of pre-conditions and effects, WSML does not provide with language elements to model them.

---

[3]http://www.bpmi.org

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 (Berardi *et al.* 2005), 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 (Berardi *et al.* 2005) 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.

6 Conclusion and Outlook

In this paper, we have presented formalisms for describing distributed business processes and services semantically. The formalism for modeling executable distributed business processes is a novel combination of the well known process algebra the π-calculus and description logic $\mathcal{SHOIN}(\mathbf{D})$ with DL-safe rules extensions. We have also presented the formal semantics of this formalism. For modeling services, We differentiated between functional and non-functional properties. Resources, behavior and changes in resources constitute the functional properties whereas the quality attributes build the non-functional properties. Furthermore, we argued that access control policies are an important aspect of the behavior of a service. We argued that it is not practical to abstract from the issuer of the 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 a semantic extension of SPKI for modeling non-functional properties of services that allows to annotate properties certified via SPKI/SDSI certificates with description logic concepts. We showed how credential based access control policies can be described and embedded as conditions in the behavioral description of services. Finally, we presented a prototype for describing distributed business processes and services and a repository for managing such descriptions.

The formalisms provided in this paper are different from existing formalisms mainly due to their expressivity, formal semantics and unified nature. Existing approaches with formal semantics either cover only the behavioral part or consider services as black boxes and cover only the types of input and output parameters. Note, that in real world business processes, there dependencies between behavior and data

and therefore, considering only one of the aspects is not sufficient for practical purposes. On the other hand, approaches that can describe both the aspects lack formal semantics so that it is difficult to develop comprehensible algorithms. Finally, to the best of our knowledge, ours is the only work that has shown how the quality attributes can be modeled in a way, such that the users can build their trust in them and how credential based access control policies can be modeled and integrated in the behavior of a service.

Semantic descriptions of business processes and services is a necessary step to enable automated reasoning about them. In this paper, our focus was on the descriptions of business processes and services with formal semantics. The development of algorithms for verification, discovery, composition and selection etc. that make use of such rich descriptions is still an open and interesting research issue. In (Agarwal 2007c), we have presented a preliminary version of model-checking based discovery. In (Agarwal 2007a), a language for specifying constraints on service properties has been presented, that together with the formalisms presented in this paper can build the basis for reasoning algorithms about distributed business processes and services.

References

Agarwal, S., and Studer, R. 2006. Automatic Matchmaking of Web Services. In Zhang, L.-J., ed., *IEEE 4th International Conference on Web Services*, 45–54. Chicago, USA: IEEE Computer Society.

Agarwal, S. 2007a. A Goal Specification Language for Automated Discovery and Composition of Web Services. In *International Conference on Web Intelligence*.

Agarwal, S. 2007b. *Formal Description of Web Services for Expressive Matchmaking*. Ph.D. Dissertation, University of Karlsruhe (TH).

Agarwal, S. 2007c. Model Checking Expressive Web Service Descriptions (Short Paper). In *IEEE 5th International Conference on Web Services*. Salt Lake City, Utah, USA: IEEE Computer Society.

Andrew, T.; Curbera, F.; Dholakia, H.; Goland, Y.; Klein, J.; Leymann, F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Trickovic, I.; and Weerawarana, S. 2003. Business Process Execution Language for Web Services. Technical report, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems.

Ankolekar, A.; Huch, F.; and Sycara, K. 2002. Concurrent Execution Semantics for DAML-S with Subtypes. In Horrocks, I., and 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)*, 14–21. Sardinia, Italy: Springer.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory Implementation and Applications.* Cambridge University Press.

Berardi, D.; Calvanese, D.; Giacomo, G. D.; Hull, R.; and Mecella, M. 2005. Automatic Composition of Transition-Based Semantic Web Services with Messaging. In Böhm, K.; Jensen, C. S.; Haas, L. M.; Kersten, M. L.; and Per-ke Larson, B. C. O., eds., *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, 613–624. Trondheim, Norway: ACM.

Clarke, D. E.; Elien, J.-E.; Ellison, C. M.; Fredette, M.; Morcos, A.; and Rivest, R. L. 2001. Certificate Chain Discovery In SPKI/SDSI. *Journal of Computer Security* 9:285–322.

Ellison, C. M.; Frantz, B.; Lampson, B.; Rivest, R. L.; Thomas, B. M.; and Ylonen, T. 1999a. Simple Public Key Certificate. http://world.std.com/cme/html/spki.html.

Ellison, C. M.; Frantz, B.; Lampson, B.; Rivest, R. L.; Thomas, B. M.; and Ylonen, T. 1999b. SPKI certificate theory. Internet RFC 2693.

Haase, P., and Motik, B. 2005. A Mapping System for the Integration of OWL-DL Ontologies. In Hahn, A.; Abels, S.; and Haak, L., eds., *IHIS 05: Proc.s of the 1st Int. Workshop on Interoperability of Heterogeneous Information Systems*, 9–16. ACM Press.

Horrocks, I., and Patel-Schneider, P. F. 2004. A Proposal for an OWL Rules Language. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, 723–731. ACM.

Milner, R.; Parrow, J.; and Walker, D. 1992. A Calculus of Mobile Processes, Part I+II. *Journal of Information and Computation* 1–87.

Motik, B.; Sattler, U.; and Studer, R. 2004. Query Answering for OWL-DL with Rules. In McIlraith, S. A.; Plexousakis, D.; and van Harmelen, F., eds., *Proc. of the 3rd. Int. Semantic Web Conference (ISWC 2004)*, volume 3298 of *LNCS*. Hiroshima, Japan: Springer.

Roman, D.; Keller, U.; Lausen, H.; de Bruijn, J.; Lara, R.; Stollberg, M.; Pollers, A.; Feier, C.; Bussler, C.; and Fensel, D. 2005. Web Service Modeling Ontology. *Applied Ontology* 1(1):77–106.

Sangiorgi, D., and Walker, D. 2001. *PI-Calculus: A Theory of Mobile Processes*. New York, NY, USA: Cambridge University Press.

Sycara, K.; Paolucci, M.; Ankolekar, A.; and Srinivasan, N. 2003. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics* 1(1):27–46.

van der Aalst, W., and van Kees, H. 2002. *Workflow Management: Models, Methods and Systems*. The MIT Press.