Expressing Semantic Web Service Behavior using Description Logics[⋆]

Markus Fronk and Jens Lemcke

SAP Research, Karlsruhe, Germany
{markus.fronk, jens.lemcke}@sap.com

Abstract. For the *automation* of major tasks of the traditional Web service (WS) usage, semantic Web services research has identified the need for (1) the provision of additional aspects of Web services, as well as (2) the formalization of these descriptions. This work focuses on the behavioral aspects of Web services and proposes to use Description Logics (DL) for their formalization. We provide DL constructs for describing interactions in sequences and parallel splits. This yields several advantages for the tasks of WS retrieval and composition. (1) The development of the retrieval and composition software becomes *simplified* which therefore results in more *robust* code. (2) The Web service description can be *extended* by additional features without touching the evaluating code. (3) In combination with the ontology language OWL-DL, the Web service description can directly be *integrated* with existing semantic Web efforts, e. g. domain ontologies.

1 Introduction

In today's industries, coping with a growing number of software artifacts in a flexible and efficient manner becomes a more and more important issue. In addition, faster changes of market situations force companies to be able to quickly adopt their business processes and set up interoperations with other parties.

Service-oriented technology arose to address some of the challenges posed by this development. Using the Web Service Description Language (WSDL)[1] in conjunction with the Simple Object Access Protocol (SOAP)[2] and the "Universal Description, Discovery and Integration" (UDDI),[3] software components can be accessed through and communicate via standardized interfaces and protocols. This increases the flexibility of the companies, but every task of the Web Service (WS) usage process still remains a manual integration activity.

The aim of Semantic Web Services (SWS) research is to automate major tasks of this usage process. These are discovery, selection, composition, execution and monitoring. Major contributions in this area are OWL-S[4] and the Web Service Modeling

⋆ This material is based upon work partially supported by the EU funding under the project DIP (FP6 - 507483). This paper reflects the author's views and the Community is not liable for any use that may be made of the information contained therein.

[1] http://www.w3.org/TR/wsdl, also: "Web Service Definition Language"

[2] http://www.w3.org/TR/soap12-part0/

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

[4] http://www.daml.org/services/owl-s/1.1/overview/

Ontology (WSMO [1]). In contrast to traditional Web service descriptions, Web service automation requires (1) some more information to be specified, and (2) a formal representation of all information given to facilitate their automatic processing.

In current approaches for Semantic Web service description, languages bearing a formal semantics are being used to add further information to "traditional" service descriptions. However, the capabilities of formalism are only used for defining the Web service description languages itself, rather than exploiting them to draw conclusions over the semantic of descriptions. For each task of the Web service usage process, specialized software is needed to work with the respective part of the formalized Web service description. This result in additional algorithms that have to be developed to interpretate the semantics described. In addition, different SWS approaches using different description formalisms can not easily be intergated. We therefore require a semantic Web service description whose formal capabilities can be further exploited. This means, that given Web service descriptions, standard software should be able to draw conclusions about which services are close to the evaluations needed to be performed during the tasks of the Web service usage process. This facilitates the separation of the model of aspects of the Web service from its interpreting application. Since we then can rely on standard reasoning software to realize standard parts, the *robustness* of software improves.

Further requirements for a general Semantic Web service description are its ability to be *extendable* for later enhancements and *integrable* with other aspects of Web service descriptions whose modelings were independently developed from each other. As an example, the "business semantics" aspect is one of these aspects which could be expressed in Description Logics (DL [2]) [3]. Other efforts are concerned with modeling Web service policies using DL [4]. Since these different aspects developed independently from each other adhere to the same formal semantics (the way inferences are drawn), they can potentially be integrated in a single analysis module considering all aspects as a whole.

In this paper, we propose to use Description Logics for the representation of Web service descriptions. A semantic Web service description consisting of a (1) technical, (2) behavioral and (3) contextual part provides enough information for the automation of the major WS usage tasks as named above. The common feature of all these tasks is finding matching service descriptions to a request description. We therefore design our Web service descriptions in such a way that through the standard subsumption reasoning this common task can be accomplished. For the implementation, we can therefore rely on complete and correct reasoner implementations.

Although all of the three aspects of semantic Web service descriptions should be expressed using DL, this paper focuses for demonstration on the behavioral aspect, i. e. the constraints between a service's operations that define the allowed order of execution. We choose DL, because it comes with a formal semantics, brings sufficient expressivity for our purposes, and is decidable. By using OWL-DL, we demonstrate its ability to serve as a general semantic Web service description language satisfying the previously described requirements of facilitating *robustness* of software, and ensuring *extendability* and *integrability* of WS descriptions.

2 Related Work

There are quite a few approaches that deal with the semantic annotation of Web Service technologies and standards to enable an automatic discovery and matchmaking process. The current UDDI discovery mechanism only insufficiently fosters the objective of automation. Most of the related work uses for the description of behavioral aspects of services either WSBPEL or language descriptions using ontologies. For matching of requests and services special algorithms have to be developed. The major difference of our approach is that we use DL to describe service behavior. Automated discovery and matchmaking can hence be realized using standard reasoner such as Racer[5] or Pellet[6]. Services matching a request can easily be determined by the subsumption mechanism when described with the DL expressions we suggest.

OWL-S Process Model The Web Ontology Language for Services (OWL-S) utilizes an ontology to describe Web services. Its concept is to provide markup language constructs to describe Web services in a semantic and thus computer-interpretable form. OWL-S builds an upper ontology for service description that consists of three main parts, namely the service profile, the service model and the service grounding.[7] The process model defines a subset of workflow features to describe a service as a process. In contrast to our solution special algorithms have to be developed to exploit the process descriptions characterized in OWL-S for matchmaking or similarity comparisons. Such algorithms based on OWL-S are described for example in [5] and [6]. The main difference of our solution is that with describing the service process flow using description logic, automated reasoning and matchmaking with standard reasoners become possible.

METEOR-S Process Designer The Managing End-To-End OpeRations for Semantic Web Services (METEOR-S) project at the Large Scale Distributed Information Systems (LSDIS) Lab at the University of Georgia annotates semantics to the complete Web service usage process. Its annotation framework is an approach to adding semantics to current industry standards such as WSDL. Finding an appropriate service for the composition is realized by a discovery engine querying an enhanced UDDI registry. The semantic descriptions published in this registry are annotated source code that is later transformed into either WSDL, WSDL-S or OWL-S.The OWL-S process model generally allows to semantically describe service behavior but the transformation made by the Semantic Description Generator however only considers the service profile and the service grounding. The process model is to the best of our knowledge not yet integrated in the transformation [7]. WSDL and WSDL-S anyway do not provide constructs to express service behavior. Behavioral aspects are hence not published in the registry. Therefore the METEOR-S approach in contrast to our solution does not consider the behavioral aspects of services in the discovery of adequate matches.

[5] http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[6] http://www.mindswap.org/2003/pellet/
[7] http://www.daml.org/services/owl-s/1.0/owl-s.pdf

WSMO Choreography The Web Service Modeling Ontology (WSMO [1])[8] is a *conceptual* specification for describing ontologies, Web services, goals, and mediators—called WSMO entities. The behavioral aspects of a Web service are called its "choreography". The choreography in WSMO is described by an adoption of Abstract State Machine (ASM [8]) statements. Roughly spoken, these statements are of the form "$\mathbf{if}\ condition\ \mathbf{then}\ updates\ \mathbf{endif}$". In the $condition$, the existence of instances of an ontological concept can be queried, which may refer to a message that was just received. In the $updates$, instances of the internal ontology can be manipulated which may trigger the sending of respective messages. This description of causal dependencies between single communications is very similar to our approach. However, the ASMs are very expressive and do not define when, e. g., a service matches a certain request with respect to their behavioral constraints.

WSBPEL Abstract Processes The Web Services Business Process Execution Language (WSBPEL)[9] can be used to describe the implementation and the observable behavioral interface of Web services. This information could then be used for automating tasks of the Web service usage process—e. g. WS composition [9]. However, there is no decidable algorithm for the representation of service behavior in WSBPEL that can be used for reasoning about relevant properties. The conversion to a suitable representation is needed.

CoBPIA Particles The Collaborative Business Processes based on Intelligent Agent Technology (CoBPIA [10]) project uses Constraint Satisfaction Problem (CSP) techniques in order to describe process steps—called particles—that are to be composed to executable WSBPEL processes. To our knowledge, particles can only be atomic processes steps that are going to be composed into workflows. Web services could be interpreted as these particles. However, the interdependencies of Web service operations that we describe are not addressed in the CoBPIA particle representation.

3 Elements of behavioral aspect

In this section, we point out the behavioral patterns that can appear with regard to services and thus have to be represented in the DL. The elements in this section are besides being introduced described in terms of their meaning with regard to the service behavior. This means that each element has another sense in terms of the allowed order of messages. This is subsequently be described. The present work focuses on the basic control patterns. After having shown that these patterns can be represented in DL future work with regard to more complex patterns such as, e.g., loops can be motivated.
Messages exchanged by services can be distinguished in incoming and outgoing messages containing either parameters processed (inputs) or provided (outputs) by the service. The service behavior, as we understand it, is made up of three constitutive aspects that have to be considered. (1) The existence of inputs and outputs (interactions), (2) The sequence in which inputs and outputs occur and (3) Control constructs representing the allowed order of interactions.

[8] http://www.wsmo.org/
[9] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

Existence of interactions The fact that interactions occur in a service, we call existence. From a service requestor's point of view, describing a service environment[10], inputs *can* be provided and outputs *must* be received. From a provider's point of view, the described inputs are *required* and the outputs are *provided*. The semantics of these different interpretations of the existence of interactions is discussed in Sect. 4. For modeling purposes, the existence of inputs and outputs is in a first step considered individually before being consolidated. A service can consist of several inputs, outputs or most commonly a combination of both. The mere existence of interactions makes no statement about the ordering of interactions.

Sequence The order of interactions is specified defining a sequence of their appearance. Thinking of public and private processes, a special order of interactions needs for example to be defined when an internal state (that is not obvious for the external observer) triggered by an interaction has to be reached *before* another interaction can be processed. Although this implicit order of interactions is possibly not obvious in the first instant only seeing the external visible public process, it is however necessary for the successful execution of the operation. This requirement has to be met by services in order to match the request. Hence it has to be made explicit. A sequence can be defined either between inputs or outputs or between both interactions.

Control constructs The third behavioral aspect besides the mere existence of interactions and the sequence are control constructs describing special derivations from the basic sequence. In [11], 20 workflow patterns based on analysis of existing workflow management systems and workflow languages are identified and described. Our solution focuses on the basic control patterns to show how service behavior can be described with DL. These have been identified to model a common set of service behavior constructs. The basic patterns consist of *sequence* (described in Sect. 3), *parallel split* along with its *synchronization* and *exclusive choice* along with the *simple merge*.[11] Advanced patterns such as *multiple choice* and *cycles* are generally imaginable, they are however neglected in this initial approach. The *parallel split* expresses the concurrency of interactions. This means that the interactions described as concurrent can appear in any order. They only have to meet other sequential requirements that could be specified, e.g. when another interaction is defined to occur *before* the *parallel split*. An example use case from a service provider's point of view could be as follows: After entering necessary login information a user has to enter both credit card and address information. The sequence of entering the credit card and address information is however of no importance for the service functionality. Both only have to appear after the login. The *exclusive choice* describes the case when only one branch of interactions is executed. At present, the modeling of the *exclusive choice* in DL has not been realized. The potential combinations of the realized concepts as there are *existence and sequence*, *existence and parallel split*, *sequence and parallel split* and combinations of all three have been modeled and introduced in the design of the DL expressions. This is described in Sect. 5

[10] Environment in this regard describes the system or service landscape in which the searched or requested service is to be embedded with its functionality

[11] *Parallel split* and *exclusive choice* always include its *synchronization* or *simple merge* pattern. An explicit distinction between these patterns is waived.

4 Service matching

After having described in the previous section the elements that represent the behavioral aspects of services and their meaning to the order of interactions, this section focuses on the semantics of these elements with regard to the matching of services. We understand the matching of services as the major and common task in the SWS usage process. In this section we will outline the semantics due to which a service matches a request given the previously introduced elements. The decision whether Web services can be matched, depends on the previously described behavioral aspects. It can however not easily be claimed by just comparing the existence of these constructs as the semantic behind these is quite complex with regards to Web services. This will be described in more detail in the following sections. Each aspect described by a service request R has to be analyzed with regard to its effects on the fulfillment by a service S. We use DL in such a way that a service S fulfills the requirements of a request R when

$$R \sqsupseteq S \tag{1}$$

can be asserted. The semantic of the behavioral aspects has been modeled in DL to satisfy this equation. This means that the behavioral constructs are described in a way that matching services can be identified by satisfying (1).

Existence As indicated in Sect. 3, the existence of *inputs* described in a request has a different effect on a services' ability to assert $R \sqsupseteq S$ than the existence of *outputs*. For the DL expression representing inputs of a request R and a service S we write I_R and I_S, respectively. For outputs we write O_R and O_S. I_R describes the maximum set of inputs that *can* be provided by the environment. A service however does not have to use all the inputs provided. A service delivering the requested goal by using less inputs still fulfills the request. This is described by (2). The set of outputs specified by the environment defined as O_R is on the other hand mandatory. It specifies the minimum set that *must* at least be *provided* by the service's set of outputs O_S. This relationship is defined by (3).

$$\text{Inputs:} \quad I_R \sqsupseteq I_S \tag{2}$$

$$\text{Outputs:} \quad O_R \sqsubseteq O_S \tag{3}$$

Sequence The sequence describing the order of inputs and outputs specified by a service request is defined as Seq_R. It has to be met by the sequence of a matching service Seq_S in order to satisfy $R \sqsupseteq S$. Equations (2) and (3) must still hold. This means that a service does not necessarily have to consume an input, even if it is described in Seq_R. This is discovered by the standard reasoning mechanism. However, when a service consumes an input that is described in a special order in Seq_R, it has to appear with the same sequential constraints in Seq_S. Outputs, in contrast, always have to be provided. Additional outputs provided by the service S that are not necessarily required by R (i.e. they are not defined in O_R) can be part of the sequence Seq_S at any position. The same applies for interactions specified without any special order (cp. Sect. 3)in R.

$$\text{Sequence:} \quad Seq_R \sqsupseteq Seq_S \tag{4}$$

Control constructs The semantics related to a *parallel split* is as follows: A request that defines a parallel split between interactions (for example I_1 and O_1)[12] explicitly states, that the order of these interactions is not relevant. (cp. Sect. 3). This means for a service to fulfill the request that it can either specify the parallelism as well, or define an order of the interactions involved. A service stating the sequence "I_1 *and then* O_1" or vice versa also matches the request. The identification of possible matches has to consider this specialty since it can not just be derived from a one-to-one mapping of control constructs. A service does not necessarily has to have the same control construct as the request to match. The aspects of existence, sequence and the control constructs can be combined in any combinations. The previously described characteristics of each of the aspects is still considered for the matching when combined to a more complex behavior.

5 DL for behavioral aspects

In the previous sections we described the elements of service behavior and their semantics with regard to the matching of services. In this section we will introduce the DL-constructs that have been developed to represent the previously discussed elements and their semantics. This will outline the basis for automated reasoning and matching. The following proof of concept using the subsequently introduced DL-constructs will finally show the obtained advantages.

The Description Logics approach, and more precisely OWL-DL, was chosen because of its decidability and its ability to be integrable with other semantic web technologies facilitating the expandability of the WS descriptions. The matching of requests and services is represented by a subsumption relation of their DL constructs. (cp. (1)) Given several services annotated with the language constructs subsequently defined for describing the behavioral aspects, standard reasoners can be exploited to infer this classification. The following constructs are modeled and developed in order to be used with the conception of classification. Matching services in terms of the aspects previously described in Sect. 4 are *automatically* identified and classified as subordination of the service request. The required DL constructs are succedingly introduced. Requests, services and interactions are modeled as classes. The behavioral aspects are expressed by special properties representing the relationships between classes.

Existence For describing the semantics of the existence of interactions (specified in Sec. 4) we introduced the existence property "has". This can colloquially be interpreted as: "*A service has the following inputs and outputs!*" Due to the fact that inputs and outputs have different semantics for fulfillment, they are also modeled in a different way. The DL expression for a set of inputs $I_1, I_2, \ldots, I_n$ for a request R or service S is represented through the inputs combined by the *union of* specifier. Thereby, (2) is satisfied. The set for a service S and request R, respectively, is hence defined as follows.

$$I_{S/R} \equiv (I_1 \sqcup I_2 \sqcup \ldots \sqcup I_n) \qquad (5)$$

[12] The parallel split can contain only inputs or only outputs or all kind of combinations

A request R *providing* several inputs is then described through defining a has-relationship with the relevant set:

$$R \equiv \exists\,\mathrm{has}.I_R \equiv \exists\,\mathrm{has}.(I_1 \sqcup I_2 \sqcup \ldots \sqcup I_n) \tag{6}$$

A service S described as $S \equiv \exists\,\mathrm{has}.(I_1 \sqcup I_2 \sqcup \ldots \sqcup I_m)$ where $m \leq n$ is hence identified as being adequate for R, because of satisfying $R \sqsupseteq S$. Services *requiring* more inputs ($m > n$) do not satisfy this condition. Outputs, in contrast, are enumerated using the *intersection of* operator for representing the semantics in (3). The DL expression for a set of outputs $O_1, O_2, \ldots, O_n$ is defined as follows.

$$O_{S/R} \equiv (O_1 \sqcap O_2 \sqcap \ldots \sqcap O_n) \tag{7}$$

A request *relying on* several outputs is then described, through defining a has-relationship with the relevant set:

$$R \equiv \exists\,\mathrm{has}.O_R \equiv \exists\,\mathrm{has}.(O_1 \sqcap O_2 \sqcap \ldots \sqcap O_n) \tag{8}$$

A service S described as $S \equiv \exists\,\mathrm{has}.(O_1 \sqcup O_2 \sqcap \ldots \sqcap O_m)$ with $m \geq n$ is identified as match, because $R \sqsupseteq S$. Services *providing* less outputs ($m < n$) are not considered to fulfill the request R. The more common case that services consist of both inputs and outputs is accommodated by the combination of (5) and (7). These concepts are combined using the *intersection of* operator. For the DL construct of the combined interactions we write $IO_{S/R}$. It is described with following equation.

$$IO_{S/R} \equiv I_{S/R} \sqcap O_{S/R} \equiv (I_1 \sqcup I_2 \sqcup \ldots \sqcup I_n) \sqcap (O_1 \sqcap O_2 \sqcap \ldots \sqcap O_m) \tag{9}$$

Requests *providing* several inputs and *relying on* several outputs are then described through defining a relationship similar to the cases (6) and (8):

$$R \equiv \exists\,\mathrm{has}.((I_1 \sqcup I_2 \sqcup \ldots \sqcup I_n) \sqcap (O_1 \sqcap O_2 \sqcap \ldots \sqcap O_m)) \tag{10}$$

This expression facilitates reasoning over both the constraints defined in (2) and (3).

Sequence The sequence of inputs and outputs is described through an ordering then-property. The succession of several interactions is represented by nesting the ordering property. Inputs and outputs are in a first step again treated independently. The nested expressions are combined similar the existence of inputs and outputs. Consecutive inputs are combined with the *union of*, consecutive outputs with the *intersection of* operator. Requests R and services S with a sequence of interactions are described as shown in (11) and (12).

$$Seq_R \equiv \exists\,\mathrm{then}.(I_1 \sqcup (\exists\,\mathrm{then}.(I_2 \sqcup (\exists\,\mathrm{then}.(\ldots \sqcup (\exists\,\mathrm{then}.(I_n))))))) \tag{11}$$

$$Seq_R \equiv \exists\,\mathrm{then}.(O_1 \sqcap (\exists\,\mathrm{then}.(O_2 \sqcap (\exists\,\mathrm{then}.(\ldots \sqcap (\exists\,\mathrm{then}.(O_n))))))) \tag{12}$$

The combination of inputs and outputs in a single sequence poses a complication that could not be completely handled yet. The difference to the combination of interactions with the DL-expression realized within the existence aspect is that within the sequence inputs and outputs have to be combined in the nested sequentiell expression. Combinations of the concepts such as inputs following other inputs, as well as outputs following other outputs can be matched correctly using the combined expression shown in (11) and (12). Combinations of the concepts such as inputs following outputs and outputs following inputs however are not correctly matched in every aspect. This means that some services are identified as match although they are not and vice versa. Goal is to have a DL representation, that expresses the combined sequence in a way, that all kinds of cases are correctly classified. In Sect. 6 we provide an example that show a case that correctly distinguishes appropriate and not appropriate services for a defined request, given the present DL-constructs.

Control constructs The semantics of a *parallel split*, as it is described in Sec. 4, is expressed as follows considering as example three existing interactions.

$$Seq_R \equiv \exists\,\mathrm{then}.(I_1 \sqcup (\exists\,\mathrm{then}.((I_2 \sqcup (\exists\,\mathrm{then}.I_3)) \sqcup (I_3 \sqcup (\exists\,\mathrm{then}.I_2))))) \quad (13)$$

$$Seq_R \equiv \exists\,\mathrm{then}.(O_1 \sqcap (\exists\,\mathrm{then}.((O_2 \sqcap (\exists\,\mathrm{then}.O_3)) \sqcup (O_3 \sqcap (\exists\,\mathrm{then}.O_2))))) \quad (14)$$

Equations (13) and (14) describe the concurrency of the second and the third interaction. The sequence of occurance is not relevant. Both a service S_1 that has the second *before* the third[13] and a service S_2 that has the third *before* the second[14] interaction are identified as match for the request R. ($R \sqsupseteq S$ is satisified)

Service behavior description A service behavior consists (as outlined in the previous sections) of the existence part (cp. Sec. 5) and the sequential ordering (cp. Sec 5 and Sec. 5). The DL expression for the service behavior description is hence the combination of both (represented by the *intersection of* operator). The behavioral aspects of requests R and services S are described as follows. Assuming a request that describes the three inputs (I_1, I_2, I_3) sequentially ordered (*first* I_1 *then* I_2 *then* I_3) is represented by the DL construct:

$$R \equiv \exists\,\mathrm{has}.(I_1 \sqcup I_2 \sqcup I_3) \sqcap \exists\,\mathrm{then}.(I_1 \sqcup (\exists\,\mathrm{then}.(I_2 \sqcup (\exists\,\mathrm{then}.(I_3))))) \quad (15)$$

Services described with the same DL expressions can be identified as a match by verifying the satisfaction of $R \sqsupseteq S$. This inference can be done using standard reasoners. Outputs are described accordingly, combining the existence and sequence constructs for outputs.

[13] $Seq(S_1) \equiv \exists\,\mathrm{then}.(I_1 \sqcup (\exists\,\mathrm{then}.(I_2 \sqcup (\exists\,\mathrm{then}.I_3))))$
[14] $Seq(S_2) \equiv \exists\,\mathrm{then}.(I_1 \sqcup (\exists\,\mathrm{then}.(I_3 \sqcup (\exists\,\mathrm{then}.I_2))))$

6 Proof of Concept

In the following we show a scenario applying the previously described DL-constructs. The behavioral aspects of a request and possible services are subsequently described with OWL-DL and existing inference mechanisms are used to draw conclusions about the match of the described services. This shows how code *robustness* can be improved exploiting existing standard reasoners, because the formal semantics of DL facilitates the separation of the model of aspects of the Web service from its interpreting application.
The first example only considering interactions is then extended to also consider behavior showing the simple *extendability* of our approach. Different aspects developed independently from each other that adhere to the same formal semantics can be integrated in a single analysis module considering all aspects as a whole. The use of OWL-DL constructs further allows for the later the non-intrusive integration with other ontological models using the same ontology language.
The scenario described in the subsequent section is as follows. A user requests an ordering service that requires a login, an order and user data as inputs and provides an order confirmation as output. Furthermore two services, (1) a store service, appropriate for our request, and (2) a fraud service spying user data are described. The corresponding DL-constructs are defined as follows.

$$\begin{aligned}
R &\equiv \mathit{Ordering} \equiv \exists\,\mathrm{has}.((\mathit{login} \sqcup \mathit{order} \sqcup \mathit{userData}) \sqcap (\mathit{conf}))\\
S_1 &\equiv \quad \mathit{Store} \quad \equiv \exists\,\mathrm{has}.((\mathit{login} \sqcup \mathit{order}) \sqcap (\mathit{wMsg} \sqcap \mathit{conf}))\\
S_2 &\equiv \quad \mathit{Fraud} \quad \equiv \exists\,\mathrm{has}.((\mathit{login} \sqcup \mathit{order} \sqcup \mathit{userData}) \sqcap (\mathit{conf}))
\end{aligned}$$

The store service requires only a login and an order and provides both a welcome message and a confirmation for the order. The fraud service in contrast has the same interactions as the ordering request. The reasoner given these service descriptions has as a result the inference shown in Fig. 1. Considering only the interactions existent within the services, both services are identified as match to the request R. ($R \sqsupseteq S$) Considering in addition the service behavior for the identification of appropriate services, we just extend the previous expressions with the introduced DL-constructs to define the allowed ordering of interactions. The most important constraint is that the order and user data is not provided until a login has occured. The store service satisfies this requirement, the fraud service however requires the user data before the login. The according DL-constructs for the request and the services are defined as follows.

$$\begin{aligned}
&R \equiv \mathit{Ordering} \equiv \exists\,\mathrm{has}.((\mathit{login} \sqcup \mathit{order} \sqcup \mathit{userData}) \sqcap (\mathit{conf})\\
&\quad \sqcap \exists\mathrm{then}.(\mathit{login} \sqcup (\exists\mathrm{then}.(\mathit{order} \sqcup (\exists\mathrm{then}.(\mathit{userData} \sqcap (\exists\mathrm{then}.(\mathit{conf}))))))))
\end{aligned}$$

$$\begin{aligned}
&S_1 \equiv \mathit{Store} \equiv \exists\,\mathrm{has}.((\mathit{login} \sqcup \mathit{order}) \sqcap (\mathit{wMsg} \sqcap \mathit{conf}) \sqcap \exists\mathrm{then}.(\mathit{login}\\
&\quad \sqcap (\exists\mathrm{then}.(\mathit{wMsg} \sqcup (\exists\mathrm{then}.(\mathit{order} \sqcup (\exists\mathrm{then}.(\mathit{userData} \sqcap (\exists\mathrm{then}.(\mathit{conf}))))))))))
\end{aligned}$$

$$\begin{aligned}
&S_2 \equiv \mathit{Fraud} \equiv \exists\,\mathrm{has}.((\mathit{login} \sqcup \mathit{order} \sqcup \mathit{userData}) \sqcap (\mathit{conf})\\
&\quad \sqcap \exists\mathrm{then}.(\mathit{userData} \sqcup (\exists\mathrm{then}.(\mathit{login} \sqcup (\exists\mathrm{then}.(\mathit{order} \sqcap (\exists\mathrm{then}.(\mathit{conf}))))))))
\end{aligned}$$

Given these service descriptions, the result inferenced by the reasoner is shown in Fig. 2. Considering as well the behavioral aspects only the store service is inferred as a match for the request. The fraud service does not follow the allowed order of interactions.

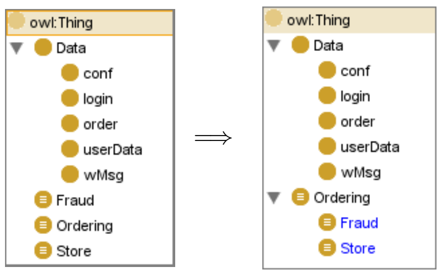

Fig. 1. Inference considering Interactions

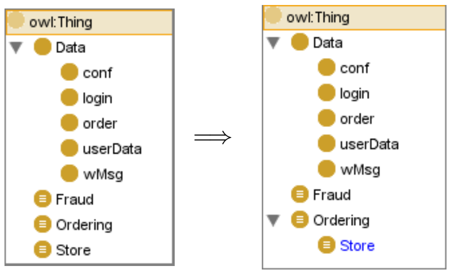

Fig. 2. Inference considering Behavior

7 Conclusions & Future Work

In this paper we have introduced a way to describe service behavior using OWL-DL. We described the different constructs for defining the mere existence of interactions, the sequence and the concurrency. With the use of DL as descriptive element our approach accomplishes the requirements of *robustness*, *extendability* and *integrability* often lacking in related work that describes service behavior aspects. On the basis of a simple example we finally showed the application of the constructs demonstrating the exploitation of standard reasoners. We see the solution described in this paper especially adaptable for the automation of the discovery and the composition of web services.

Based on the expressions given, and their evaluation by the subsumption-reasoning of a standard reasoner (also called "classification" [2, p. 48]), the task of finding Web services matching the technical (inputs and outputs) and behavioral aspects (causal constraints over inputs and outputs) of a Web service request can be automatically executed. Using the technique described for the matchmaking task, there is no additional application logics needed.

In addition, we understand our definition of a formal description of Web service behavior suitable for automatic matchmaking of requests and Web services as an important step towards the creation of an extensible, robust, automatic semantic Web service composer. The task of semantic Web service composition, amongst other jobs, mainly bases on well-known "syntactic" Web service composition as well as "semantic" discovery. Traditional Web service composition uses, e. g., planning [12] or configuration techniques [13] to come up with the composed workflow of Web services. Semantic composition mainly differs in the way the composer finds services being candidates for addition to the final workflow. For this step, it uses the additional information that is given in a semantic Web service description. This information may be the subsumption relations of different input and output elements, behavioral constraints, policy requirements and properties, or other arbitrary aspects of Web service descriptions. The described approach can therefore additionally be exploited for the tasks existent in the service composition as it supports the semantic description of services. Further research however has to occur in this area.

Future work will focus on three aspects of the service behavior description. (1) The combination of interactions in the sequence expression to cover all special cases in order to provide a set of DL constructs that allows to describe the main spectrum of service behavior. (2) An expression for modeling the *exclusive choice*. (3) The representation of cycles and advanced workflow patterns as extension to the basic expressions for describing even special service behavior. The future research, related to these aspects of describing the service behavior with the presented DL-constructs, will finally enable us to understand whether the expressibility of DL is enough for expressing the constraints that characterize service behavior.

References

1. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. In: Applied Ontology 1. Volume 1. IOS Press (2005) 77–106
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. In Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: Description Logic Handbook, Cambridge University Press (2003)
3. Preist, C., Cuadrado, J.E., Battle, S., Grimm, S., Williams, S.K.: Automated business-to-business integration of a logistics supply chain using semantic Web services technology. In Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A., eds.: International Semantic Web Conference. Volume 3729 of Lecture Notes in Computer Science., Springer (2005) 987–1001
4. Grimm, S., Lamparter, S., Abecker, A., Agarwal, S., Eberhart, A.: Ontology based specification of Web service policies. In Dadam, P., Reichert, M., eds.: GI Jahrestagung (2). Volume 51 of LNI., GI (2004) 579–583
5. Bansal, S., Vidal, J.M.: Matchmaking of web services based on the daml-s service model (2003)
6. Ankolekar, A., Paolucci, M., Sycara, K.: Spinning the owl-s process model, toward the verification of owl-s process models (2004)
7. Rajasekaran, P., Miller, J., Verma, K., Sheth, A.: Enhancing web services description and discovery to facilitate composition (2004)
8. Börger, E., Stärk, R.F.: Abstract State Machines—A Method for High-Level System Design and Analysis. Springer-Verlag (2003)
9. Trainotti, M., Pistore, M., Calabrese, G., Zacco, G., Lucchese, G., Barbon, F., Bertoli, P., Traverso, P.: Astro: Supporting composition and execution of web services. In Benatallah, B., Casati, F., Traverso, P., eds.: ICSOC. Volume 3826 of Lecture Notes in Computer Science., Springer (2005) 495–501
10. Wahl, T.: Konzeption und realisierung einer ontologie zur modellierung und ableitung von geschaeftsprozessen. Diplomarbeit, Technische Universitaet Berlin, DEUTSCHLAND (2005)
11. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns (2002)
12. Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: AIMSA. (2004) 106–115
13. Stumptner, M.: Configuring web services. In: Proceedings of the Configuration Workshop at the 16th European Conference on Artificial Intelligence (ECAI). (2004) 10–1/10–6