

---

## Functional Description of Services

Where the previous chapter defined the semantics of WSML ontologies (the static knowledge component of Web service descriptions), this chapter describes how WSML can be used to capture the functionality of Web services. We consider both set-based and state-based capabilities, describing the models underlying these kinds of description, as well as the formal relations that can be defined between goals and Web services based on these models.

In general it is important to understand the conceptual model and the particular assumptions underlying each specific approach to service description. Different approaches vary greatly in the level of detail that can be expressed. In terms of functional descriptions we are interested in modelling what a service does as opposed to how a certain functionality is achieved. If we consider for example a Web service capable of processing credit card payments, we are interested in the details of this functionality, i.e., which credit cards are accepted (Visa, Master Card, JCB, ...), which currencies and what kind of fraud checking is provided. In this chapter we are not concerned with non-functional aspects such as the supported transport layer security or the guarantees in terms of service availability.

Describing the functionality of a service explicitly has many advantages. Most notably it allows potential users to easily find a certain Web service. In terms of discovery the Web service life cycle starts once it has been created and published by a provider (cf. the Web service usage process in Figure 3.2 on page 3.2). Web service technology allows to the invoking of services over the internet, however without knowing about a service a potential user will not be able to use it. So if someone is in the need of a certain functionality he needs to perform some kind of search in order to find a list of services that is able to provide the functionality desired. In order to decide whether a service does actually provide a certain functionality or not a matchmaking procedure based on some kind of description has to be performed. Semantic annotations can reduce the amount of manual labor involved in this process by providing accurate descriptions of client requests and the services offered.

Consider, in contrast, a discovery service that only allows to query services by keyword will require a human to manually assess a potentially large list of matches to determine which are actually providing the functionality desired. When both the request and the response are semantically described a matching engine can guarantee that all results indeed fulfill the desire expressed in the request. For example a request for a payment service to handle Visa cards can be matched against a service offering all *major* credit cards by using the appropriate background knowledge (in an ontology). However, performing a keyword query for “visa creditcard” will not necessarily return only payment services. In fact using an existing search engine for Web services<sup>1</sup> one quarter of the results of this query have been validation services.

In the remainder of the chapter we will first give an overview of the most prominent approaches to the functional description of Web services, in Section 6.1. We then proceed with a description of the models underlying capability descriptions, as well as notions of consistency and capability matching, both for set-based and state-based capability descriptions, in the Sections 6.2 and 6.3, respectively.

## 6.1 Approaches to Functional Description

When describing functionality we can do this in various ways. The simplest possible way is to base it on natural language text, i.e., to use a set of keywords to capture the functionality. For the payment service this could be “payment”, “credit card”, etc. Obviously this type of description has the advantage of not requiring complex logical expressions and trained knowledge engineers, however the descriptions are not very precise and tend to be ambiguous. Imagine you want to express that a service that accepts all major credit cards, but only does business with merchants based in North America. With keywords alone one cannot capture these facts in a way that they can be easily understood and processed by a machine.

With an ontology language we can express this specific information. We can model the concept of a payment service that has numerous properties. An example of such a property would be “accepted credit cards”, whose values are the accepted credit cards. We refer to this kind of functional description as “set-based”. The functionality is described by an ontological concept that describes the service (i.e., what it delivers).

Although a set-based description using an expressive ontology language can be used to describe many aspects of a service it has limitations: we can not explicitly refer to constraints that must hold in order for the service to be able to execute, nor can we describe the concrete dependencies between the pre and post-states. For example this is required in order to describe that before some service is executed some service the available credit of the card

<sup>1</sup> <http://seekda.com>

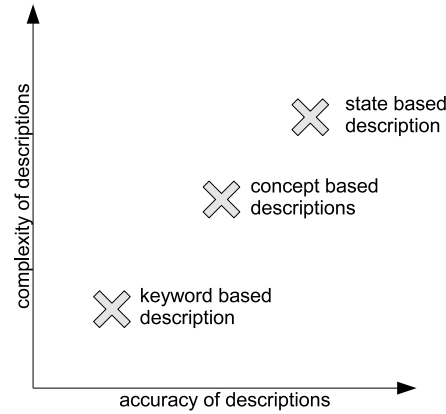
holder must be at least as high as the payment amount. We will refer to this type of description as “state-based”.

Still finer grained levels of description are sometimes required. Suppose we want to describe a service conducting bank transfers; besides single bank transfers the service offers processing of multiple transfers at once, in a batch processing fashion. Assuming that a batch can contain credits as well as withdrawals the lowest intermediate balance will depend on the order of how the transfers are internally executed. In case all withdrawals are performed first and only then all credits, the account will have a lower intermediate balance then if the order would be reversed. Although in general the functional description is not concerned about how a particular functionality is achieved we still might be interested in modeling execution invariants, i.e., some conditions that must hold in all states that exist between invoking the Web Service and after that invocation has completed. This might be for example that the account balance must be always greater than zero. However, in this chapter we are not concerned with this level of detail. Such constraints are part of the behavioral description of the service (see Chapter 7).

Every of these levels implies a different description of Web services, ranging from detailed characterizations of pre- and post-states to less detailed descriptions using (complex) concepts in an ontology and simple unstructured keywords. Consequently, the achievable accuracy of a result in the discovery process varies significantly, as more or less structure is reflected in the descriptions. On the other hand, the ease of providing the descriptions varies drastically between these levels as well. Whereas simple keywords are easy to provide, using ontological concepts already requires the publisher to either familiarize himself with the applicable domain ontologies or create their own ones. The provision of detailed state-based descriptions requires most effort. The more fine-grained the information, that the descriptions reveal, the more complex the algorithms must be that to deal with these descriptions. Therefore, there is a trade-off between the accuracy and complexity of descriptions. In turn, lower accuracy of a Web service descriptions leads to a lower precision of the discovery process and higher complexity of descriptions leads to increased effort and skills required for service description, as well as decreased efficiency of the discovery process. This trade off is illustrated in Figure 6.1.

In principle the different models (state- versus set-based) are not tied to particular representation formalisms. WSML, however allows writing service descriptions at all of these different levels of granularity. Moreover we can use different WSML variants to describe Web services at each of those levels. For example the set-based approach can be realized using WSML-DL. WSML leaves the choice of the concrete variant to the user, so that he can choose the expressivity suitable for his particular task.

Since a keyword based description (and the appropriate matchmaking) do not require a semantic language such as WSML we omit further discussion here. The interested reader can find additional information on how keyword



**Fig. 6.1.** Trade-off between accuracy and complexity for service descriptions

based technologies have been applied to the domain of Web service discovery in [47, 92].

## 6.2 Set-Based Web Service Description

In this section we present a formal modeling approach for Web services and goals that is based on set theory and exploits ontologies as formal, machine-processable representation of domain knowledge. We discuss Web service discovery based on this approach for simple semantic descriptions and describe how the set-based model in WSML is implemented.

The set-theoretic notions discussed in this section give rise to a set-based *discovery mechanism*.

### 6.2.1 The Model

One main characteristic of object oriented approaches is that the problem domain is understood as a collection of objects and objects with similar properties can be grouped together into sets or classes. Each class captures common (syntactic and semantic) features of their elements. Features can be inherited from one class to another by defining class hierarchies. In this way, the domain of discourse can be structured as a collection of classes and each class is a set of things. We apply this model to describe the functionality of a Web service as a set.

A Web service provides some value to the entity that invokes it. The invocation itself is based on Web service technologies like SOAP and WSDL, however these technical details of invocation are not necessarily relevant for discovery based on functional specifications. Only if one considers discovery together with the adaptation/invocation of the service does one need to cater for

the particularities of the invocation mechanism. To judge whether the functionality offered by a Web service matches the one requested it is not necessary to check compatibility on the interface level, since it might be possible to bypass mismatches by mediators. The execution of the Web service with particular input values generates certain piece of information as an output and achieves certain changes in the state of the world. An output as well as an effect can be considered as objects which can be embedded in some ontology.

Goals on the other hand specify the desire of a client that he wants to have resolved after invoking a Web service, that means they describe the information the client wants to receive as output of the Web service execution as well as the effects on the state of the world. This desire can be represented as sets of elements that are relevant to the client as the outputs and the effects of a service execution. Goals therefore refer to the desired state of the world.

Both Web services and goals are represented by sets of objects. These sets of objects are described in ontologies that capture general knowledge about the domain under consideration. Matching a Web service to a Goal requires that there exists some mutual relationship between the objects used in their descriptions. This relationship can be established with reference to the ontology (or ontologies) used to describe these objects. We define domain-independent notions of matching in Section 6.2.2.

An important observation is that the description of a set of objects for representing a goal or a Web service actually can be interpreted in different ways and thus the description by means of a set is not semantically unique: A modeler might want to express that either all of the elements that are contained in the set are requested or can be delivered ( $\forall$ ), or that only some of these elements are requested or can be delivered ( $\exists$ ). Consider for example someone who wants to find pricing information of some multimedia product ( $\exists$ ), versus someone who wants to know about the prices of all available products in a certain product category ( $\forall$ ).

Clearly, a modeler has some intuition in mind when specifying such a set of relevant objects for a goal or Web service description and this intention essentially determines whether we consider two descriptions to match or not. Thus, this intuition should be stated explicitly in the descriptions of service requests or service advertisements.

Given our previous considerations we can model goals and Web services as follows: A goal  $G$  and a Web service  $W$  are represented by sets of objects from some common universe  $U$  (i.e.,  $G, W \subseteq U$ ). These sets represent relevant objects for the description. In addition, both  $G$  and  $W$  have an associated *intention*  $I \in \{\forall, \exists\}$ .

### 6.2.2 Matching Web Services and Goals

In order to assert a goal  $G$  and a Web service  $W$  match on a semantic level, the sets  $G$  and  $W$  describing these elements have to be interrelated somehow; specifically, we expect that some set-theoretic relationship between  $G$  and  $W$

has to exist. The most basic set-theoretic relationships that one might consider are the following:

$G = W$	Set equality
$G \subseteq W$	Goal description is a subset of the Web service description
$W \subseteq G$	Web service description is a subset of the goal description
$G \cap W \neq \emptyset$	The goal and Web service descriptions have some elements in common
$G \cap W = \emptyset$	The goal and Web service descriptions have no elements in common

These set-theoretic relationships provide the basic means for formalizing our intuitive understanding of a match between goals and Web services. In fact, the above relationships have been considered in the context of Description Logic-based matching in [94] and [114]. The terminology for matching notions in these papers has been inspired by work done in the context of matching based on component specifications [139].

However, we have to keep in mind that in our model these sets only capture part of the semantics of goal and service descriptions, namely the relevant objects for the service requestor or service provider. The intentions of these sets in the semantic description of the goal or Web service is not considered but clearly affects whether particular set-theoretic relationships between  $G$  and  $W$  correspond to intuitive notions of matching. Hence, we have to consider the intentions of the respective sets as well. In the following we will discuss the set-theoretical relation for one notion of matching (namely, intersection match) as an example; we summarize all notions of matching in Table 6.1 on page 104.

### Example: Intersection Match

We provide a detailed discussion for the case where there exist common elements in goal and Web service descriptions. In this case the set of relevant objects that are advertised by the service provider and the set of relevant objects for the requester have a non-empty intersection, i.e., there is an object that is relevant for both parties. In a sense, this criterion can be seen as the weakest possible criterion for semantic matching in the set-based modeling approach.

Let us consider the goal in Listing 6.1 and the Web service in Listing 6.2. The Web service can be used to request the price of a product as well as to buy a product. Recall that concepts in ontology represent sets of instances. One can verify that the sets of instances of the concepts `GenericProductStore-Service` and `RetrievePriceInformationMedia` potentially intersect, i.e., it is possible that object is an instance of both concepts. Therefore, there is an *intersection match* between the goal and the service.

We now how an intersection match should be interpreted, depending of the intentions of the goal and the Web service.

---

```

namespace { "-" http://example.org/" ,
          tasks "-" http://example.org/ontologies/tasks/" }
goal GetPricelInformation
  annotations
    dc:description hasValue "Describes the desire of getting price information of any multi
      media product"
  endAnnotations
  importsOntology PricelInformation
  capability RetrievePricelInformationMedia

ontology PricelInformation
  importsOntology { "-" http://example.org/ontologies/tasks/Tasks", "-" http://example.org/
    ontologies/products/MediaProducts" }
  concept RetrievePricelInformationMedia subConceptOf tasks#RetrievePricelInformation
    forProduct impliesType media#MediaProduct

```

---

Listing 6.1. Goal for retrieving price information

---

```

namespace { "-" http://example.org/" ,
          tasks "-" http://example.org/ontologies/tasks/" }
webService GenericProductStoreService
  annotations
    dc:description hasValue "Describes a shop offering information about products and there
      prices"
  endAnnotations
  importsOntology PricelInformation
  capability GenericProductStore

ontology GenericProductStoreOntology
  importsOntology { "-" http://example.org/ontologies/tasks/Tasks", "-" http://example.org/
    ontologies/products/Products" }
  concept GenericProductStore
  axiom definedBy
    forall ?object (?object memberOf GenericProductStore equivalent
      (?object memberOf tasks#RetrievePricelInformation or ?object memberOf tasks#Buy)).

```

---

Listing 6.2. Web service offering price information and buying functionality

- $I_G = \forall, I_W = \forall$ : The service requester wants to get all of the objects and the service provider claims that the Web service is able to deliver all the objects specified. In this case, the requester needs can not be fully satisfied by the service. However, the service can contribute to meeting the requirements of the client. Thus, we consider this case a *partial match*.
- $I_G = \exists, I_W = \forall$ : The service requester wants to get some of the objects, whereas the service provider claims that the Web service is able to deliver all the objects specified. In this case, the requester needs are fully covered by the Web service. The requester might as well receive objects which are not relevant for him. We consider this case *match*.
- $I_G = \forall, I_W = \exists$ : The service requester wants to get all of the objects, whereas the service provider claims that the Web service is able to deliver only some of the objects specified. In this case, the requester needs are not fully covered. We are even not able to determine whether the service

	$I_W = \forall$ $I_G = \forall$	$I_W = \forall$ $I_G = \exists$	$I_W = \exists$ $I_G = \forall$	$I_W = \exists$ $I_G = \exists$
$G = W$	Match	Match	Partial Match	Match
$G \subseteq W$	Match	Match	Possible Match	Possible Match
$G \supseteq W$	Partial Match	Match	Partial Match	Match
$G \cap W \neq \emptyset$	Partial Match	Match	Possible Partial Match	Possible Match
$G \cap W = \emptyset$	Non-match	Non-match	Non-match	Non-match

**Table 6.1.** Set-theoretic criteria, intentions, and our intuitive understanding of matching

actually can deliver any of the objects desired by the requester and hence we consider this a *possible partial match*.

- $I_G = \exists, I_W = \exists$ : The service requester wants to get some of the objects and the service provider claims that the Web service is able to deliver some of the objects specified. In this case we have a *possible match*.

For a complete discussion of all possible matches we refer the reader to [82]. However, given the discussion for the case of an intersection match, it is straight-forward to apply it to the remaining cases. In the next subsection we give a brief summary of all possible combinations.

### Summary: Understanding of Matching

Given some goal  $G$  and some Web service  $W$ , Table 6.1 summarizes the discussion and shows under which circumstances the presence of which set-theoretic relationship between  $G$  and  $W$  is considered as a match, a partial match, a possible match, a possible partial match or a non-match.

In earlier Description Logic-based approaches to service discovery (e.g., [114, 94]) the notion of “intention” was not been reflected explicitly. As we have shown above, intentions capture an important aspect of goal and Web service descriptions and affect essentially the situations in which certain set-theoretic criteria represent our intuitive understanding of matches.

We believe that certain combinations of intentions will occur more often in practice than others: Web service providers for example have a strong interest in their Web services being discovered. If we compare the number of possible matchings with a given goal under existential and universal intentions, it seems most likely that providers tend to use universal intentions, even if the description does not necessarily model the actual functionality of the service accurately and promises too much. However, if a service provider wants to be more accurate with his Web service description then in many situations he would have to use the existential intention.

For service requesters (in particular in an e-Business setting) we expect that the existential intention will suffice in many situations, however the requester has the freedom to express stronger requests than existential goals



(using the universal intention) if he needs to and thus get more accurate results in these situations.

### 6.2.3 Consistency of Descriptions

What we have not considered so far is the possibility of inconsistent goal or Web service descriptions. In our situation, a set-based capability is inconsistent if it is necessarily interpreted as the empty set – that is, the concept representing the capability does not have instances in any model of the task ontology. Clearly such descriptions are not desirable: a requester who is asking for nothing and Web services that do not deliver anything are simply superfluous and undesired. Nonetheless, inconsistent descriptions might occur in cases where the descriptions are complex or refer to several complex ontologies which are not themselves designed by the modeler.

Additionally, when just being ignored they can have an undesired impact on matching and thus discovery: Consider for example an inconsistent goal description, i.e.,  $G = \emptyset$ . If we check  $G$  for matching Web services using the Plugin-criterion, i.e.,  $G \subseteq W$ , then obviously every Web service matches. For a user (who is not aware that his description is inconsistent, since otherwise he would usually not pose the query) the result would seem unintuitive and even incorrect because all checked services actually will match.

Checking for inconsistent goal and Web service descriptions is not a task that is only applicable at the design time. It is of course good practice to allow creating an inconsistent description, but consistency does not depend exclusively on the description itself; it also depends on the ontologies the description refers to. Hence, changes to such ontologies potentially lead to inconsistent descriptions. Moreover, since Web service and goal description may refer to different ontologies, their combination (necessary when performing matchmaking) may make a previously satisfiable goal description unsatisfiable. Thus before checking for a match the satisfiability of each description involved must be checked.

### 6.2.4 Ranking Matches

As shown in Table 6.1, we basically have for each pair of intentions for a goal and a Web service several formal criteria that capture actual matches, partial matches, possible matches, and non-matches. According to elementary set-theory the single criteria are not completely separated, but the following interdependencies hold. Notice that in some cases we require descriptions to be non-empty as discussed before.

$$\begin{aligned}
 G = W &\Rightarrow G \subseteq W \\
 G = W &\Rightarrow G \supseteq W \\
 G \subseteq W, G \neq \emptyset &\Rightarrow G \cap W \neq \emptyset \\
 G \supseteq W, W \neq \emptyset &\Rightarrow G \cap W \neq \emptyset
 \end{aligned}$$

That means that certain formal set-theoretic criteria that we consider here are stronger notions than others: if the stronger relationship holds than the weaker relationship must hold as well. These properties induce a partial order on the set-theoretic criteria:

$$(G = W) \preceq (G \subseteq W), (G \supseteq W) \preceq (G \cap W \neq \emptyset)$$

Given a goal and a Web service description let “subsumes match” be the criterion that captures the actual match, then a weaker criterion, such as “intersection match” does also hold. However one has to note that the stronger criterion provides additional knowledge about the relationship between goal and Web service. In this particular example a “subsumes match” also guarantees that no objects are delivered besides the one requested, since this property might be important for a requestor, it does make sense to allow the use of a particular criterion for the matching between goal and Web service descriptions by the requestor. A service requester basically can exploit this property during a discovery process in order to ensure certain convenient properties from the discovered Web services.

To sum up, we have seen that there are cases where a client could benefit from exploiting the additional semantics captured by matching criteria that are stronger than the weakest match. Hence, it makes sense to not only allow a request to demand a match (i.e., at least the weakest criterion to be fulfilled), but also to allow to specify the exact criteria and thus raise the semantic requirements that are captured by the criterion. In particular this makes sense for the case that a client does not want to accept that a Web service potentially delivers objects that have not been explicitly requested (in this case a subsumes or an exact match has to be requested).

We have seen as well that in our general framework there is only one such additional property that actually can be considered as useful, namely the property of a Web service to not deliver objects that are irrelevant to the user. This leads us to allow the client to specify what particular kind of match he is accepting, by specifying the following three dimensions:

- Intention of the goal
- match, partial match, possible partial match, possible match
- within each match it can be additionally specified if a service is allowed to deliver objects that are not requested.

#### *Partial Order on “Match”*

Similar to the partial order that is defined for the basic set theoretic matching criterion, we can also define a logically order on our intuitive understanding of the matching notion.

$$\text{Match} \preceq \text{PartialMatch}, \text{PossibleMatch} \preceq \text{PossiblePartialMatch}$$

	$I_W = \forall$ additional objects      no additional objects		$I_W = \exists$ additional objects      no additional objects	
$I_G = \forall$ Match	$G \subseteq W$	$G = W$	–	
$I_G = \exists$ Match	$G \cap W \neq \emptyset$	$G \supseteq W$	$G \supseteq W$	$G \supseteq W$
$I_G = \forall$ Partial Match	$G \cap W \neq \emptyset$	$G \supseteq W$	$G \supseteq W$	$G \supseteq W$
$I_G = \exists$ Partial Match	$G \cap W \neq \emptyset$	$G \supseteq W$	$G \supseteq W$	$G \supseteq W$
$I_G = \forall$ Possible Match	$G \subseteq W$	$G = W$	$G \subseteq W$	–
$I_G = \exists$ Possible Match	$G \cap W \neq \emptyset$	$G \supseteq W$	$G \cap W \neq \emptyset$	$G \supseteq W$
$I_G = \forall$ Possible Partial Match	$G \cap W \neq \emptyset$	$G \supseteq W$	$G \cap W \neq \emptyset$	$G \supseteq W$
$I_G = \exists$ Possible Partial Match	$G \cap W \neq \emptyset$	$G \supseteq W$	$G \cap W \neq \emptyset$	$G \supseteq W$

**Table 6.2.** Formal criteria for checking degrees of matching

The partial ordering can be exploited during matchmaking: in order to ensure that a property is satisfied when matching (e.g.,  $I_G = \exists, I_W = \forall$  and additional objects might be delivered), the discovery component has to apply only the weakest criterion still fulfilling the request. In the given example it is only required to check for an intersection match ( $G \cap W \neq \emptyset$ ) and not for all set theoretic relation separately. Table 6.2 represents the result of this discussion for all possible combinations. Matching criteria that are colored gray in the table indicate that the criterion does in fact not check the intuitive matching criteria specified (e.g., partial match or match), but one which also satisfies the requested criteria due to the partial order on the intuitive matching notions.

Matching in the set-based framework for Web service discovery is based on rather simple semantic annotations and thus can provide only limited guarantees on the actual accuracy of the results: A detected match between a goal and a Web service actually does not ensure that the Web service can really fulfill the user requirements depicted in the goal, since important information that affects this possibility is not specified in the descriptions; e.g., the requester's ability to satisfy the requirements of the Web service when invoking and interaction with the service, namely the pre-conditions as well as a prescribed choreography. In the next section we will present a model that allows to capture the relation between pre- and post-state.

### 6.3 State-Based Web Service Description

This section presents a model that allows a more precise definition of the functionality of a Web service, i.e., state-based capabilities. The model itself is independent of any specific logical formalism. In general the model can be formally represented in various logics of sufficient expressivity – for example, different WSMML variants – to enable reasoning with semantic descriptions of Web service capabilities. We will illustrate the model in an intuitive fashion, for the formal definitions we refer to [82].

---

```

...
concept CreditCard
  hasNumber ofType xsd#integer
  hasSecurityCode ofType xsd#integer
  hasExpiryDate ofType xsd#date
  hasOwner ofType Client
  hasLimit ofType xsd#decimal

relation creditCardCharge(ofType CreditCard, ofType xsd#decimal, ofType xsd#dateTime)

axiom noChargeBeyondLimit
  definedBy
    !— creditCardCharge(?creditCard, ?charge, ?date) and
      ?creditCard[hasLimit hasValue ?limit] memberOf CreditCard and
      ?charge > ?limit.
...

```

---

**Listing 6.3.** Excerpt from the commerce domain ontology

### 6.3.1 Abstract State Spaces

We consider the world as a set of entities that change over time. Entities that act in the world - which can be anything from a human user to some computer program - can affect how the world is perceived by themselves or other entities at some specific moment in time. At any point in time, the world is in one particular state that determines how the world is perceived by the entities acting therein. We need to consider some language for describing the properties of the world in a state. In the following we assume an arbitrary (but fixed) signature  $\Sigma$  that is based on some domain ontologies, and a language  $\mathcal{L}(\Sigma)$ .

Although the concept of ontologies has been already introduced we include a small excerpt of a commerce ontology in Listing 6.3 to illustrate a signature expressed in WSM. Within the listing we define the concept of an credit card and that no charge can be made that is beyond the credit card limit.

In the context of dynamics and properties of the world that can change, it is useful to distinguish between symbols in  $\Sigma$  that are supposed to have always the same, fixed meaning (e.g.,  $\geq, 0$ ) and thus can not be affected by any entity that acts in the world, and symbols that can be affected and thus can change their meaning during the execution a Web service (e.g., **memberOf**, **hasValue**). We refer to the former class of symbols as *static* (denoted by  $\Sigma_S$ ) and the latter as *dynamic* symbols (denoted by  $\Sigma_D$ ).

#### *Abstract State Spaces*

We consider an abstract state space  $\mathcal{S}$  to represent all possible states  $s$  of the world. Each state  $s \in \mathcal{S}$  completely determines how the world is perceived by each entity acting in  $\mathcal{S}$ . Each statement  $\phi \in \mathcal{L}(\Sigma)$  of an entity about the (current state of) the world is either true or it is false. We consider classical logic (and thus only true and false as truth values) here. However, the presented model can be used as it is in the context of non-classical logics (e.g.,

---

```

...
instance myVisaCard memberOf CreditCard
  hasNumber hasValue 4444111122223333
  hasLimit hasValue 2000,00

relationInstance creditCardCharge(myVisaCard, 100, _dateTime(2007, 1, 1, 12, 19, 10))
...

```

---

**Listing 6.4.** Possible state in an abstract state space

WSML-Rule) by considering a restricted class of models  $\mathcal{I}$ , e.g., stable models in the case of WSML-Rule. A state  $s \in \mathcal{S}$  in fact defines an interpretation  $\mathcal{I}$  (of some signature  $\Sigma$ ). However, not all  $\Sigma$ -Interpretations  $\mathcal{I}$  represent meaningful observations since  $\mathcal{I}$  might not respect some “laws” that the world  $\mathcal{S}$  underlies, e.g., that a credit card charge may not exceed the credit cards limit.

These laws are captured by a background ontology  $\Omega \subseteq \mathcal{L}(\Sigma)$  as for example in Listing 6.3. Listing 6.4 illustrates some meaningful observations according to our background ontology.

#### *Changing the World*

By means of well-defined change operations, entities can affect the world through state transitions over  $\mathcal{S}$ . In our setting, these change operations are single concrete executions of Web services  $W$ . Following [82], a change operation is represented by a service  $S$  that is accessed via a Web service  $W$ . The transition is achieved by executing  $W$  with some given input data  $i_1, \dots, i_n$  that determine the concrete service execution  $S$ , i.e.,  $S \approx W(i_1, \dots, i_n)$ .

Given some input data  $i_1, \dots, i_n$ , the execution of a Web service essentially causes a state transition  $\tau$  in  $\mathcal{S}$ , transforming the current state of the world  $s \in \mathcal{S}$  into a new state  $s' \in \mathcal{S}$ . A transition  $\tau$  will in general not be an atomic transition  $\tau = (s, s') \in \mathcal{S} \times \mathcal{S}$  but a sequence  $\tau = (s_0, \dots, s_n) \in \mathcal{S}^+$ , where  $s_0 = s$ ,  $s_n = s'$  and  $n \geq 1$ . Intermediate states can be useful if it is of interest to express invariants that must hold throughout an entire execution. Similarly intermediate states might be important when describing long-lasting transactions. However, for the purpose of functional descriptions we envision that it is not necessary to model those intermediate steps, since they are generally not of interest during analysis or use of capability descriptions. In the context of our model we describe the functionality of a Web service by the pre-state ( $s$ ) and the post-state ( $s'$ ). More fine grained statements about a service can be described using choreographies as detailed in Chapter 7.

#### *Outputs as Changes of an Information Space*

After the execution of a Web service, it can send some information as output to the requester. We consider these outputs as updates of the so-called information space of the requester of a service. The information space is a part of the state.

---

```

...
instance myVisaCard memberOf CreditCard
  hasNumber hasValue 4444111122223333
  hasLimit hasValue 2000,00

relationInstance creditCardCharge(myVisaCard, 100, _dateTime(2007, 1, 1, 12, 19, 10))
...

```

---

**Listing 6.5.** Statements describing a particular information space

Taking up our background ontology including the credit card we can model the output of an online purchase using transaction status and some id that uniquely identifies the transaction, as illustrated by the instance depicted in Listing 6.5.

#### *Observations in Abstract States*

Our aim is to describe all the effects of Web service executions for a requester. Obviously, a requester can observe in every state  $s \in \mathcal{S}$  related properties represented by statements  $\phi$  in  $\mathcal{L}(\Sigma)$  that hold in  $s$ . Additionally, he can perceive the information space, as described above. The abstract state space  $\mathcal{S}$  in a sense “corresponds” to the observations that can be made, both in the information space and the “real world”. Consequently, we represent the observations related to a state  $s$  by an observation function  $\omega$ , which assigns a  $\Sigma$ -interpretation  $\mathcal{I}$  to every state  $s \in \mathcal{S}$ . We denote that part of  $\mathcal{I}$  concerned with the real world by  $\omega_{rw}(s)$  and the part concerned with the information space by  $\omega_{is}(s)$ . However, we require the observation function  $\omega$  to be a (fixed) total function as it can not be arbitrary. This means that the observations  $\omega(s)$  of any entity are well-defined in every abstract state  $s$ .

#### *Web Service Executions*

Given some input  $i_1, \dots, i_n$ , a Web service execution induces a state transition  $(\omega(s), \omega(s'))$  that can be observed by the service requester. However, not all such transitions of abstract states represent meaningful state transitions caused by a Web service execution. For a transition to faithfully represent some service execution we need to require that between the states  $s$  and  $s'$  some change can be observed by the invoker. We need to require some further constraints on the transition such that we can interpret  $s, s'$  as a possible run  $W(i_1, \dots, i_n)$  of a Web service  $W$ , as we will discuss below. We call  $s$  the pre-state of the execution and  $s'$  the post-state of the execution.

Returning to our running example we illustrate in Listing 6.6 pre- and post-states by looking at how the statements relating to a single credit card might evolve during a simple online purchase.

---

```

... //statements in s
instance myVisaCard memberOf CreditCard
  hasNumber hasValue 4444111122223333
  hasLimit hasValue 2000,00
...

... //statements in s'
instance myVisaCard memberOf CreditCard
  hasNumber hasValue 4444111122223333
  hasLimit hasValue 1900,00

relationInstance creditCardCharge(myVisaCard, 100, _dateTime(2007, 1, 1, 12, 19, 10))
...

```

---

**Listing 6.6.** State transition of a bank transfer Web service

### Web Services

A Web service  $W$  then can be seen as a set of executions  $W(i_1, \dots, i_n)$  that can be delivered by the Web service in any given state of the world to a requester when being equipped with any kind of valid input data  $i_1, \dots, i_n$ . However, in order to keep track of the input data that caused a specific execution, we need to represent a Web service in terms of a slightly richer structure than a set, namely a mapping between the provided input values  $i_1, \dots, i_n$  and the resulting execution  $W(i_1, \dots, i_n)$ . This implies that we use a deterministic model for Web services here.

The corresponding Web service to our running example would be a payment service offered by a credit card company to a particular online shop that takes as input a credit card number, a expiration date and the amount to be charged. For all valid credit cards (given a sufficient initial limit) it will charge the amount requested to the card. Thus the actual Web services corresponds to a set of state transitions (and not only one), where each transition is determined by the concrete input values supplied.

Figure 6.2 illustrates the presented model. The Web service  $W$  provides three different concrete services, each of them having different pairs of input. Every single state is determined by the two components of  $\omega$  - the information space and the real world. The Web service is a set of possible transitions that is denoted by a dark green area inside the abstract state space.

The model presented gives a thorough mathematical model. For the formal definitions of this model we refer to [82]. For the purpose of this book the previous intuitive description should suffice. It is important to understand that this model is defined in order to allow an unambiguous interpretation of Web service and goal descriptions, i.e., that it provides a semantic to the syntactical description within a capability. In the following we outline some basic semantic analyzes that can be performed on top of this model.

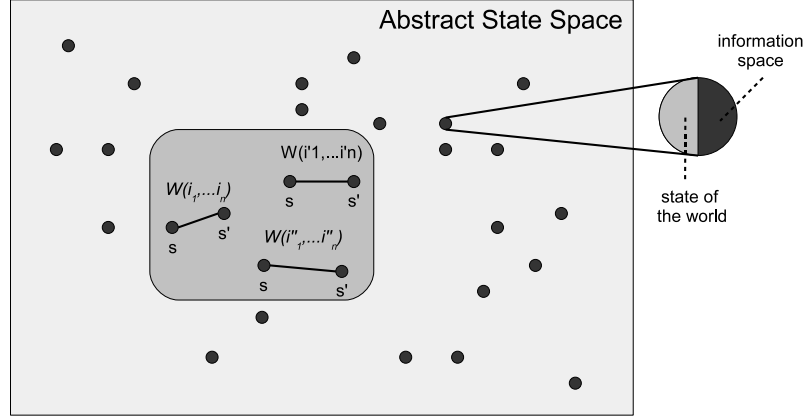


Fig. 6.2. Abstract model of Web services

---

```

namespace _ "http://example.org/"

webService creditCardCharge
importsOntology _ "http://example.org/ontologies/commerce/Commerce"
sharedVariables {?creditcard, ?amount}
capability
  precondition
    definedBy
      ?amount > 0 and
      ?creditcard memberOf CreditCard
  effect
    definedBy
      creditCardCharge( ?creditcard, ?amount, ?date)

```

---

Listing 6.7. A non-realizable credit card payment service

### 6.3.2 Semantic Analysis of State-Based Web Service Descriptions

For demonstrating the suitability of the proposed model, this section shows its beneficial application for semantic analysis of functional descriptions. Based on our model-theoretic framework, we can carry over several semantic standard notions from mathematical logic [52, 58] that refer to formal descriptions and are based on the model notion to our particular context in a meaningful way.

#### *Realizability*

We can now define realizability of a description as the corresponding notion to satisfiability in WSMML ontologies. A set of formulae is satisfiable if it has a model, i.e., if there exist an interpretation of the formulae that is true. The very same notion can be applied to Web services. Consider Listing 6.7, which contains the functional description of a Web service for credit card processing.



---

```

webService creditCardCharge
importsOntology "http://example.org/ontologies/commerce/Commerce"
sharedVariables {?creditcard, ?amount}
capability
  precondition
    definedBy
      ?amount > 0 and
      ?creditcard [hasLimit hasValue ?limit] memberOf CreditCard and
      ?amount < ?limit.
  effect
    definedBy
      creditCardCharge( ?creditcard , ?amount, ?date)

```

---

**Listing 6.8.** A realizable credit card payment service

At a first glance, the given description seems to be implementable within some Web service. However, taking a closer look at the domain ontology (cf. Listing 6.3) it becomes clear that this is not the case. The ontology defines that a charge might not exceed the limit of a particular credit card. Nonetheless the pre-condition does not prevent the amount being greater than the credit card limit.

Let us assume that there is a Web service realizing this description. When considering an input binding where the amount ( $?amount$ ) is greater than the limit ( $?creditcard[hasLimit hasValue ?limit]$ ) then the pre-condition is satisfied and thus the post-condition should hold in the final state of the respective execution is reached. However, this is inconsistent with the domain ontology since the charge would exceed the credit card limit. This is a contradiction and shows that no Web service exist that can adhere to above descriptions for all possible input bindings. To fix the description such that it becomes realizable, we need to extend the pre-condition, as illustrated in Listing 6.8.

The example illustrates the usefulness of the notion of realizability. It provides a tool for detecting functional descriptions that contain flaws that might not be obvious to the modelers. It is shown in [82] how the problem of realizability of a Web service description  $\mathcal{D} \in \mathcal{F}$  can be reduced to a well-understood problem in  $\mathcal{L}$  for which algorithms already exist.

#### *Functional Refinement*

Similar to the notion of satisfiability we can look at the notion of logical entailment, which is usually defined as follows: An formula  $\phi$  logically entails a formula  $\psi$  iff every interpretation  $\mathcal{I}$  which is a model of  $\phi$  (i.e.,  $\mathcal{I} \models_{\mathcal{L}} \phi$ ) is also a model of  $\psi$ . Substituting interpretations by Web services, formulae by functional descriptions and the satisfaction  $\models_{\mathcal{L}}$  by capability satisfaction  $\models_{\mathcal{F}}$ . In a similar way we can define the notion of functional refinement that corresponds to the notion of logical entailment: We use  $\mathcal{D}_1 \sqsubseteq \mathcal{D}_2$  to denote that description  $\mathcal{D}_1$  is a functional refinement of description  $\mathcal{D}_2$  in  $\mathcal{A}$ .

Intuitively speaking,  $\mathcal{D}_1 \sqsubseteq \mathcal{D}_2$  means that  $\mathcal{D}_1$  is more specific than  $\mathcal{D}_2$ : Every Web service (no matter which one) that provides  $\mathcal{D}_1$  can also provide

---

```

...
precondition
definedBy
...
    ?product memberOf mediaproduct#MediaProduct
...
effect
definedBy
...
    commerce#creditCardCharge(?creditcard,?price,?date) and
forall ?product,?entry (?products[hasEntry hasValue ?entry] and
    ?entry[hasProduct hasValue ?product] implies
    commerce#productDelivery(?product, 1, ?address,?dd)).
...

```

---

**Listing 6.9.** Excerpt of the mediaShoppingCapability

$\mathcal{D}_2$ . In other words,  $\mathcal{D}_1$  must describe some piece of functionality that always fits the requirements  $\mathcal{D}_2$  as well. However, Web services that provide  $\mathcal{D}_2$  do not have to satisfy  $\mathcal{D}_1$  and therefore, a Web service that provides  $\mathcal{D}_1$  can do something more specific than required by  $\mathcal{D}_2$ .

As an example, consider the excerpt of the `mediaShoppingCapability` in Listing 6.9. Using our formal definition we can examine another definition and check if it is a functional refinement of the previous description. Let us assume a second Web service has an identical capability, however the pre-condition is slightly broader, instead of requiring a `MediaProduct` the product only needs to be an instance of `Product`. The first Web service has a more specific pre-condition, thus every Web service fulfilling the second service automatically fulfills the first Web service.

This notion can beneficially be applied within functionality-based match-making. With the knowledge about the refinement a repository of services can be pre indexed to ease discovery at runtime.

### *Discovery*

We can use our model for discovery through checking functional refinement by looking for Web services that guarantee certain postconditions and/or effects. For instance, let us assume that some Person wants a particular CD shipped to his home. Since a CD is a media product we can infer that he can use both Web services presented.

To conclude this chapter we briefly summarize the set-based and state-based approaches. The set-based approach has in particular the following advantages

- This modeling approach is based on a very simple and intuitive perspective of the world where everything is considered in terms of sets (or concepts)
- In opposite to other approaches we start from building a model and analyzing the intuitive understanding of a match and then try to capture the intuitive semantics of match. This results in giving the modeler more freedom to express his/her desire, e.g., through the use of intentions.

- The approach represents a general framework which does not fix the language to be used for describing goals and Web services. In particular, it allows in general description which are not possible to express using Description Logics and thus provides increased expressiveness.
- Because of the same conceptual modeling style, this approach potentially allows a seamless integration of descriptions formalized in different languages, such as present in the WSML Family of languages.

Nevertheless it has to be noted that this approach does not capture the actual relation between service input and the corresponding outputs. Thus, the semantics of a Web service is only described in a conceptual manner. In fact, this can be too coarse-grained for enabling the automation of the discovery and later execution of a service.

The state-based description of Web services a more fine grained model to describe Web service descriptions. It requires significant skills to write those descriptions, but allows to capture a great deal of the functionality of a real world Web service. Still the model has some limitations about what can be expressed or captured:

- Only finite processes can be described. A specific limitation of pre- and post-condition style descriptions is that they are based on the assumption that there will be a final state of a computation, i.e., the computation terminates. Although, this might be a valid assumption for a wide variety of Web services, it does not allow the specification of non-terminating components which nonetheless deliver meaningful functionality. An example in a technical system would be an operation system of a computer which does not terminate or a Web service that implements a clock and periodically sends a the current time to a subscribed clients.
- Statements about intermediate states. Like in common specification frameworks, our model for the semantics of Web services considers a Web service as a set of atomic state-changes, i.e., possible intermediate states during an execution of a service are invisible for an external observer and can not be referred to in a formal specifications. For planing purposes it might be relevant or useful to allow to describe services in a more detailed way, for instance as a constraint on possible execution paths. For the same reason, it is not possible to express properties which do hold during the whole execution of the service, which have been studied in Dynamic Logics in the context of throughout modalities.
- Message format and order. We do not describe interface details such as the message format or the order of messages and thus even if the model is capable of automatically determining which service does semantically match, it does not provide guarantees on the syntactic level. Those could potentially be resolved by mediators as presented for example in [106].

In this chapter we have seen two means for describing Web service functionality: set-based and state-based capabilities, and associated discovery

mechanisms. In the case of set-based descriptions, discovery can be reduced to checking subsumption of concepts in an ontology (cf. Section 5.2.2). In the case of state-based descriptions, discovery can be reduced to checking logical entailment (cf. Sections 5.2.2 and 5.2.3).

A functional description is a somewhat abstract view of a Web service; it does not capture the behavior of the service. In the following chapter we describe the WSMML choreography language, which allows describing such behavioral and interaction patterns.

Modeling Semantic Web Services

The Web Service Modeling Language

de Bruijn, J.; Kerrigan, M.; Keller, U.; Lausen, H.;

Scicluna, J.

2008, XIV, 192 p. 28 illus., Hardcover

ISBN: 978-3-540-68169-4