On the Semantics of Functional Descriptions of Web Services

Uwe Keller, Holger Lausen, and Michael Stollberg

Digital Enterprise Research Institute (DERI)
University of Innsbruck, Austria
{firstname.lastname}@deri.org

Abstract. Functional descriptions are a central pillar of Semantic Web services. Disregarding details on how to invoke and consume the service, they shall provide a black box description for determining the usability of a Web service for some request or usage scenario with respect to the provided functionality. The creation of sophisticated semantic matchmaking techniques as well as exposition of their correctness requires clear and unambiguous semantics of functional descriptions. As existing description frameworks like OWL-S and WSMO lack in this respect, this paper presents so-called *Abstract State Spaces* as a rich and language independent model of Web services and the world they act in. This allows giving a precise mathematical definition of the concept of Web Service and the semantics of functional descriptions. Finally, we demonstrate the benefit of applying such a model by means of a concrete use case: the *semantic analysis* of functional descriptions which allows to detect certain (un)desired semantic properties of functional descriptions. As a side effect, semantic analysis based on our formal model allows us to gain a formal understanding and insight in matching of functional descriptions during Web service discovery.

1 Introduction

Enabling automated detection of Web services that adequately serve a given request or usage scenario is a main objective of Semantic Web service technology. Therefore, the functional description of a Web service specifies the provided functionality. Disregarding detailed information on how to invoke and consume the Web service the purpose of functional descriptions is to provide a black box description of *normal runs* of a Web service, i.e. without regard to technical or communication related errors that might occur during service usage.

In the most prominent overall description frameworks for Semantic Web services, functional descriptions are essentially *state-based* and use at least *prestate and poststate constraints* to characterize intended executions of a Web service. In OWL-S [13], *Service Profiles* encompass the functional description that is described by the in- and output, and by preconditions and results. Their counterpart in WSMO [11] are *capabilities* that are defined by preconditions, assumptions, postconditions, and effects. However, both models lack of clear and unambiguous semantics for functional descriptions [9]. This is essential for developing appropriate semantic matchmaking mechanisms for discovery, or for proving the correctness of functional descriptions - in general for any sort of symbolic computation based on functional descriptions in these frameworks.

With respect to this necessity, we present a rigorous formal model of Web services and the world they act as the basis for clear semantic definitions of functional descriptions. Addressing the most fine-grained perspective on Web services and their functional descriptions as identified in previous work [7, 8] (namely, the level of rich semantic descriptions) we aim at applicability of the presented model in *any* setting where *state-based functional descriptions* are used, e.g. in frameworks like OWL-S and WSMO. Since both do *not* restrict themselves to a *particular* language for describing states (i.e. preconditions and postconditions), a formal definition that is usable for these frameworks must be modular and independent of the language chosen for describing state-conditions.

The contribution of this paper is as follows:

- we present so-called *Abstract State Spaces* as a sufficiently rich, flexible, and language independent model for describing Web services and the world they act in (Sec. 2)
- based on the model we describe what a functional description actually is and properly specify their formal semantics (Sec. 2)
- we give concise, formal definitions of all concepts involved in our model (Sec. 3)
- we demonstrate the applicability of the introduced model by a specific use case: the semantic analysis of functional descriptions (Sec. 4). In particular, we clearly define desirable properties of functional descriptions like realizability and semantic refinement and show how to determine these properties algorithmically based on existing tools in a provably correct way. Hereby, we can reconstruct generalized versions of results on matching between component specifications that are well-known in the software component community [18], but (based on our formal model) get some additional insights the relation between the *semantic notion* of refinement and the *syntactic criterion* for checking semantic matches, that have can not be discussed in [18] (see [10] for a deeper discusson).

The Bigger Picture. The model presented in this paper can be considered as a small first step towards a mathematical model for *service-oriented architectures*. Based on a more emcompassing and rich mathematical model, we will be able to give semantics to formal descriptions of such *architectures* and (similarly to what we discussed for the simple case of capabilities here) to reason about such descriptions in a well-understood and verifiably correct way by extension and refinement of the presented basic model. We expect that the presented model provides a suitable and flexible foundation for such non-standard extensions.

Overview of the Solution. As a part of rich model description frameworks like OWL-S and WSMO, functional descriptions of Web services $\mathcal{D}$ are *syntactic expressions* in some specification language $\mathcal{F}$ that is constructed from some (non-logical) signature $\Sigma^{\mathcal{F}}$. Each expression $\mathcal{D} \in \mathcal{F}$ captures specific requirements on Web services W and can be used to constrain the set of all Web services to some subset that is interesting in a particular context. Hence, the set of Web services W that satisfy the functional description $\mathcal{D}$ (denoted by $W \models_{\mathcal{F}} \mathcal{D}$) can be considered as actual meaning of $\mathcal{D}$. This

way, we can define a natural *model-theoretic semantics* for functional descriptions by defining a satisfaction relation $\models_{\mathcal{F}}$ between Web services and functional descriptions. In comparison to most common logics, our semantic structures (i.e interpretations that are used to assign expressions $\mathcal{D}$ a truth value) are simply a bit more complex. In fact, they can be seen as generalizations of so-called Kripke structures [1, 17].

In general, various simpler syntactic elements are combined withing a functional description $\mathcal{D} \in \mathcal{F}$. State-based frameworks as the ones mentioned above use at least preconditions and postconditions. Whereas $\mathcal{D}$ refers to Web services, these conditions refer to simpler semantic entities, namely *states*, and thus in a sense to a "static" world. Such state conditions ϕ are expressions in (static) language $\mathcal{L}$ over some signature $\Sigma^{\mathcal{L}}$. Single states s determine how the world is perceived an external observer of the world and thus the truth value of these conditions. Formally, we have a satisfaction relation $\models_{\mathcal{L}}$ between states s and state expressions ϕ, where $s \models_{\mathcal{L}} \phi$ denotes that ϕ holds in state s. In essence, we can observe that on a syntactic level a language $\mathcal{L}$ for capturing static aspects of the world is *extended* to a language $\mathcal{F}$ that captures dynamic aspects of the world.

In order to define a similar extension on a semantic level, we *extend* the definition of the satisfaction $\models_{\mathcal{L}}$ (in $\mathcal{L}$) to a definition of satisfaction $\models_{\mathcal{F}}$ (in $\mathcal{F}$). This way, our definition is highly modular, language-independent to a maximum extent and focuses on the description of dynamics (i.e. possible *state transitions*) as the central aspect that the functional description language $\mathcal{F}$ adds on top of state description language $\mathcal{L}$. It can be applied to various languages $\mathcal{L}$ in the very same way as it only requires a model-theoretic semantics for the static language $\mathcal{L}$ (which almost all commonly used logics provide). Furthermore, our model-theoretic approach coincides to the common understanding of functional descriptions to be *declarative* descriptions *what* is provided rather than *how* the functionality is achieved.

2 Towards a Model of Web Services

The following introduces *Abstract State Spaces* as a flexible approach for defining a rigorous, formal model of Web services, the world they act in, and the meaning of functional descriptions. While introducing the model informally in the following, mathematically concise definitions are given in Section 3.

A changing world. We consider the world as an entity that changes 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. At each 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 Σ (that usually is based on domain ontologies), and some language $\mathcal{L}(\Sigma)$ derived from the signature.

We use classical First-order Logic for illustration purposes in the following, however we stress that we are by no means bound to this specific language, i.e. other languages such as WSML or OWL can easily be used instead in our framework. Consider a signature $\Sigma \supseteq \{isAccount(\cdot), balance(\cdot), \geq, 0, 1, 2, \ldots\}$ that allows to talk about

bank accounts and their balance. Then, $\mathcal{L}(\Sigma)$ allows comparing the respective values, for instance by expressions like $\forall ?x.(isAccount(?x) \Rightarrow balance(?x) \geq 0)$ stating that the balance of any account needs to be non-negative. In the context of dynamics and properties of the world that change, it is useful to distinguish between symbols in Σ 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. $isAccount(\cdot)$, $balance(\cdot)$). We refer to the former class of symbols by *static symbols* (denoted by Σ_S) and the latter by *dynamic symbols* (denoted by Σ_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 false. Thus, a state $s \in \mathcal{S}$ in fact *defines* an interpretation $\mathcal{I}$ (of some signature Σ). However, not all Σ-Interpretations $\mathcal{I}$ represent senseful observations since $\mathcal{I}$ might not respect some "laws" that the world $\mathcal{S}$ underlies, e.g. that the balance of any bank account is not allowed to be negative. In the following, we assume that these laws are captured by a background ontology $\Omega \subseteq \mathcal{L}(\Sigma)$ and denote the set of Σ-Interpretations that respect Ω (i.e. the models of Ω) by $Mod_\Sigma(\Omega)$. Considering our example signature from above and a background ontology Ω with $\{\forall ?x.(isAccount(?x) \Rightarrow balance(?x) \geq 0), isAccount(acc_1), isAccount(acc_2)\} \subseteq \Omega$, the following interpretation denotes a state $\mathcal{I} : balance(acc_1) = 10$. In contrast, the interpretation $\mathcal{I} : balance(acc_2) = -40$ does not denote a state in $\mathcal{S}$ (wrt. Ω).

Changing the World. By means of well-defined change operations, entities can affect the world and modify their current state. Such operations denote state transitions in $\mathcal{S}$. In our setting, these change operations are single concrete *executions* of Web services W. Following [8, 7], a change operation is represented by a *service* S that is accessed via a Web service W. S is achieved by executing W with some given input data $i_1, \ldots, i_n$ that specify (for a service provider) *what* kind of particular *service* S accessible via W is requested by the client, i.e. $S \approx W(i_1, \ldots, i_n)$.

Given input data $i_1, \ldots, i_n$, the execution of a Web service W essentially causes a state transition τ in $\mathcal{S}$, transforming the current state of the world $s \in \mathcal{S}$ into a new state $s' \in \mathcal{S}$. However, a transition τ will in general not be an *atomic transition* $\tau = (s, s') \in \mathcal{S} \times \mathcal{S}$ but a sequence $\tau = (s_0, \ldots, s_n) \in \mathcal{S}^+$, where $s_0 = s$, $s_n = s'$ and $n \geq 1$. In every intermediate state s_i in τ some effect can already be perceived by an enitity. This is especially relevant for Web services that allow accessing long lasting activities that involve multiple conversation steps between the requester and the Web service W. If we consider e.g. some international bank transfer having as concrete input data the information to transfer \$20 from acc_2 to acc_1 the web service execution might involve the following intermediate state between s_{pre} and s_{post} :

$$s_{pre} : balance(acc_1) = 10 \wedge balance(acc_2) = 100$$
$$s_1 : balance(acc_1) = 10 \wedge balance(acc_2) = 80$$
$$s_{post} : balance(acc_1) = 30 \wedge balance(acc_2) = 80$$

Outputs as Changes of an Information Space. During the execution $W(i_1, \ldots, i_n)$

of a Web service W, W 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 S. More precisely, we consider the *information space* of some service requester as a set $IS \subseteq U$ of objects from some universe U. Every object $o \in IS$ has been received by the requester from W during the execution $W(i_1, \ldots, i_n)$. During the execution the information space itself evolves: starting with the empty set when the Web service is invoked the execution leads to a monotonic sequence of information spaces $\emptyset = IS_0 \subseteq IS_1 \subseteq \ldots \subseteq IS_k$. Within our bank transfer example, during some financial transaction $tid891$ we might first receive a message acknowledgment $msgid23$ and then a confirmation that the transaction has been approved and initialized:

$$IS_1 = \{ack(20051202, msgid23, tid891)\}$$
$$IS_2 = \{ack(20051202, msgid23, tid891), confirm(acc_1, acc_2, 20, tid891)\}$$

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 $s \in \mathcal{S}$ world-related properties represented by statements ϕ in $\mathcal{L}(\Sigma)$ that hold in s. However, additionally there might be other aspects that an observer of the world can perceive in an abstract state s. In our model, this includes at least the information space $IS \subseteq U$ described above. Thus, an abstract state $s \in \mathcal{S}$ in a sense „corresponds" to *all observations* (relevant for the uses of our formal model) that can be made in s. For our purpose here, this means all pairs of Σ-interpretations $\mathcal{I} \in Mod_\Sigma(\Omega)$ and (possible) information spaces $IS \subseteq U$. Consequently, we represent the observations related to a state s by an observation function $\omega : \mathcal{S} \rightarrow Mod_\Sigma(\Omega) \times \mathcal{P}(U)$ that assigns every state $s \in \mathcal{S}$ a pair $(\mathcal{I}, IS)$ of a Σ-interpretation $\mathcal{I}$ (respecting the domain laws Ω) and an information space IS. We denote the first component of $\omega(s)$ by $\omega_{rw}(s)$ (*real-world properties*: how an entity perceives the world) and the second component by $\omega_{is}(s)$ (*information space*: how the invoker perceives the information space). However, we require the observation function ω to be a (fixed) total function as is can *not* be arbitrary. This means that the observations $\omega(s)$ of any entity are well-defined in *every* abstract state s. Moreover, any perception representable in terms of $\mathcal{L}(\Sigma)$ and U that is consistent with the domain model Ω should actually corresponds to some abstract state $s \in \mathcal{S}$ by means of ω, so that ω is surjective[1]. By considering abstract states as *abstract* objects without a predefined or fixed structure, and the separated assignement of a formal structure representing the actual observations that can be made in a state (by means of ω), we are able to address extensions of our model that might be needed in future extensions in a clean and modular way (e.g. when formally describing conversations between a group of agents in the world). Moreover, we can easily include representation of contextual aspects [3] in our model (e.g. that a state s is observed differently by various agents).

Web Service Executions. Given some input $i_1, \ldots, i_n$, the Web service execution

[1] However, since we assume a fixed signature Σ and thus a limited language for describing observations about the world, we do not assume that ω is injective, i.e. there could be distinct states s, s' of the world which can *not* be distinguished by the (limited) language $\mathcal{L}(\Sigma)$, i.e. $\omega_{rw}(s) = \omega_{rw}(s')$.

$W(i_1, \ldots, i_n) = (s_0, \ldots, s_m)$ starting in state s_0 induces a sequence of observations $(\omega(s_0), \ldots, \omega(s_m))$ which can be made by the service requester during the execution. However, not all such sequences τ of abstract states actually do represent a meaningful state-transition caused by an execution of W. For τ to faithfully represent some $W(i_1, ..., i_n)$ we need to require at least that for any two adjacent states s, s' in $W(i_1, \ldots, i_n)$ some change can be observed by the invoker, and that objects which are in the information space (i.e. have been received by the invoker) at some point in time during the execution can not disappear until the execution is completed. As discussed later, in general we need to require some further constraints on a sequence τ such that we can interpret τ as a possible run $W(i_1, \ldots, i_n)$ of a Web service W. We call s_0 the pre-state of the execution, s_m the post-state of the execution, and all other states in $W(i_1, ..., i_n)$ intermediate states.

Web Services. A Web service W then can be seen as a set of executions $W(i_1, \ldots, i_n)$ 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 $i_1, \ldots, i_n$. 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 $i_1, \ldots, i_n$ and the resulting execution $W(i_1, \ldots, i_n)$. Figure 1 illustrates the proposed model.

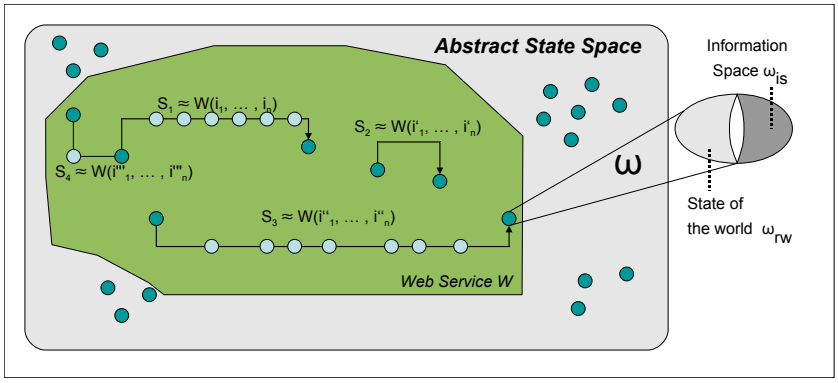

Fig. 1. An abstract Model of the World and Web services therein.

Functional Description of Web Services. Combining state-related descriptions with a functional description (or capability) essentially creates a constraint on possible Web Service executions. Executions of a Web service W whose capability has been described in terms of a capability description $\mathcal{D}$ (where $\mathcal{D}$ contains a prestate-constraint ϕ^{pre} and a post-state constraint ϕ^{post}) can no longer be arbitrary possible executions τ in an abstract state space $\mathcal{S}$, but whenever the prestate s_0 of τ respects ϕ^{pre} then the final state s_m of τ must respect ϕ^{post}. Otherwise, τ is not considered to represent an actual execution of a Web service W with capability $\mathcal{D}$.

3 Abstract State Spaces and Web Services

We will now give a series of definitions which capture the preceding semi-formal discussion in a rigorous way.

In the following, let Σ be some signature, $\mathcal{L}(\Sigma)$ be some logic over signature Σ and $\Omega \subseteq \mathcal{L}(\Sigma)$ be some background theory capturing relevant domain knowledge. Let $\Im(\Sigma)$ denote the set of Σ-interpretations in $\mathcal{L}(\Sigma)$ and $\Im(\Sigma,\mathcal{U})$ denote the set of Σ-interpretations such that for all $\mathcal{I} \in \Im(\Sigma)$ the universe considered in $\mathcal{I}$ (denoted by $universe(\mathcal{I})$) is a subset of $\mathcal{U}$. For a set U we use $\mathcal{P}(U)$ to denote the powerset of U. Let $Mod_\Sigma(\Omega,\mathcal{U})$ denote the Σ-Interpretations $\mathcal{I} \in \Im(\Sigma,\mathcal{U})$ which satisfy the domain model Ω (i.e. $\mathcal{I} \models_{\mathcal{L}(\Sigma)} \Omega$). We denote the meaning of a symbol $\alpha \in \Sigma$ that is assigned by an interpretation $\mathcal{I}$ by $meaning_\mathcal{I}(\alpha)$.

Given a signature $\Sigma_0 = \Sigma_D \cup \Sigma_S$ that is partitioned into a set Σ_D of dynamic symbols and a set Σ_S of static symbols, we extend Σ_0 to a signature Σ by adding a (new) symbol α_{pre} for each $\alpha \in \Sigma_D$. The set of these pre-variants of symbols is denoted by Σ_D^{pre}. Furthermore, add a new symbol out to Σ_0. The intention is as follows: Σ_S contains symbols that are interpreted always in the same way (static symbols), Σ_D contains symbols whose interpretation can change during the execution of a Web service (dynamic symbols), and Σ_D^{pre} contains symbols that are interpreted during the execution of a Web service as they have been right before starting the execution. Finally, out denotes the objects in the information space. The symbols that have been added to Σ_0 can be used when formulating post-state constraints to describe changes between pre-states and post-states in a precise way.

Definition 1 (Abstract State Space). *An **abstract state space** $\mathcal{A} = (\mathcal{S},\mathcal{U},\Sigma,\Omega,\omega)$ is a 5-tuple such that (i) $\mathcal{S}$ is a non-empty set of **abstract states**, (ii) $\mathcal{U}$ is some non-empty set of objects called the **universe** of $\mathcal{A}$ (iii) $\Omega \subseteq \mathcal{L}(\Sigma)$ is consistent (iv) $\omega : \mathcal{S} \to Mod_\Sigma(\Omega,\mathcal{U}) \times \mathcal{P}(\mathcal{U})$ is a total surjective function that assigns to every abstract state s a pair of a Σ-interpretation $\omega_{rw}(s)$ satisfying Ω and an information space $\omega_{is}(s)$ and (v) for all $s,s' \in \mathcal{S}$ and $\alpha \in \Sigma_S$: $meaning_{\omega_{rw}(s)}(\alpha) = meaning_{\omega_{rw}(s')}(\alpha)$.* □

In $\mathcal{A}$, Ω can be considered as a domain ontology representing (consistent) background knowledge about the world. It is used in any sort of descriptions, like preconditions etc. Clause (v) captures the nature of static symbols. In the following, $\mathcal{A}$ always denotes an abstract state space $\mathcal{A} = (\mathcal{S},\mathcal{U},\Sigma,\Omega,\omega)$.

For interacting with a Web service W, a client can use a technical interface. When abstracting from the technical details, every such interface basically provides a set of values as input data. The required input data represent the abstract interface used for interaction with the Web service from a capability point of view.

Definition 2 (Web Service Capability Interface, Input Binding). *A **Web service capability interface** IF of a Web service W is a finite sequences of names $(i_1,\ldots,i_n)$ of all required input values of a W. An **input binding** β for a Web service capability interface IF in $\mathcal{A}$ is a total function $\beta : \{i_1,\ldots,i_n\} \to \mathcal{U}$. The set of all input bindings for IF in $\mathcal{A}$ is denoted by $In_\mathcal{A}(IF)$.* □

An input binding essentially represents the input that is provided by the invoker of a Web service W during the *entire* execution of W.

Definition 3 (Web Service Execution). *A (possible)* ***Web service execution*** *in $\mathcal{A}$ is finite sequences $\tau = (s_0, \ldots, s_m) \in \mathcal{S}^+$ of abstract states such that for all $0 \leq j < m$ and $0 \leq i, k \leq m$ (i) $\omega(s_j) \neq \omega(s_{j+1})$, (ii) $\emptyset = \omega_{is}(s_0) \subseteq \omega_{is}(s_1) \subseteq \ldots \subseteq \omega_{is}(s_m)$, (iii) $universe(\omega_{rw}(s_i)) = universe(\omega_{rw}(s_k))$, (iv) $\omega_{is}(s_i) \subseteq universe(\omega_{rw}(s_i))$, (v) for all $\alpha \in \Sigma_D : meaning_{\omega_{rw}(s_0)}(\alpha) = meaning_{\omega_{rw}(s_i)}(\alpha_{pre})$ and (vi) $meaning_{\omega_{rw}(s_i)}(out) = \omega_{is}(s_i)$. We denote the set of all possible Web service executions in $\mathcal{A}$ by $Exec(\mathcal{A})$.* □

This definition gives detailed conditions under which a sequence τ can be considered as a Web service execution. Clause (iii) requires that within an execution the universes which are related to abstract states s_j are the same[2]. In other words, universes (which are used to interpret state-based expression) that are related by an execution are not arbitrary, but specifically related to each other. In particular, (iii) ensures that within a functional description $\mathcal{D}$ postconditions can talk about *every object* that the precondition can refer to as well. Hence, precise comparisons between various states of an execution becomes possible. Clause (iv) requires that for every abstract state that involved in the execution its information space is part of the universe of the abstract state. This allows to relate and compare information space objects with real-world objects in state-based expressions. Finally, clauses (v) and (vi) ensure that in all intermediate and final states, the pre-versions α_{pre} of dynamic symbols α are interpreted as α in the prestate s_0 of the execution and that the symbol out represent the respective information space.

Definition 4 (Web Service, Web Service Implementation). *A* ***Web service implementation*** *W of some Web service capability interface $IF = (i_1, \ldots, i_n)$ in $\mathcal{A}$ is a total function $\iota : In_{\mathcal{A}}(IF) \times \mathcal{S} \rightarrow Exec(\mathcal{A})$ that defines for all accepted input bindings in $In_{\mathcal{A}}(IF)$ and abstract states $s \in \mathcal{S}$ the respective Web service execution of W in $Exec(\mathcal{A})$. Formally, we require for ι that $\iota(\beta, s) = (s_0, ..., s_m)$ implies $s_0 = s$ for all $s \in \mathcal{S}, \beta \in In_{\mathcal{A}}(IF)$. A* ***Web service*** *$W = (IF, \iota)$ is a pair of a Web service capability interface IF and a corresponding Web service implementation ι of IF.* □

One can consider the mapping ι as a marking of execution sequences in $\mathcal{A}$ by the input data that triggers the execution. Since we define a Web service implementation in terms of a function which maps to single Web service executions, we consider *deterministic* Web services, i.e the execution is fully determined by the input binding β and the intial state s_0 only. Any sort of uncertainty about what is going to happen when executing W (e.g. unexpected failures due to the environment the Web service is embedded in) is not considered in our model. In being a total function on $In_{\mathcal{A}}(IF) \times \mathcal{S}$, the definition reflects the fact that ι represents an (abstract) *implementation*, i.e. (unlike for specifications) every possible effect in every situation is *fully determined* by ι.

Based on this formal machinery, we can now formalize the meaning of functional descriptions $\mathcal{D} \in \mathcal{F}$ that are based on a state-description language $\mathcal{L}(\Sigma)$. In the following, we write $\mathcal{I}, \beta \models_{\mathcal{L}(\Sigma)} \phi$ to express that formula $\phi \in \mathcal{L}(\Sigma)$ is satisfied under Σ-interpretation $\mathcal{I}$ and variable assignment β. We assume that a functional description $\mathcal{D} = (\phi^{pre}, \phi^{post}, IF_{\mathcal{D}})$ consists of a precondition $\phi^{pre} \in \mathcal{L}(\Sigma_0)$, and a postcondition

[2] In order to model dynamic universes (e.g. object creation and deletion) one needs to model object existence in the state-description language $\mathcal{L}$ itself, for instance by a dynamic unary relation `existing`.

$\phi^{post} \in \mathcal{L}(\Sigma)$. $IF_{\mathcal{D}} \subseteq FreeVars(\phi^{pre}, \phi^{post})$ denotes the set of (free) variable names in $\mathcal{D}$ which represent inputs for the Web service under consideration. The logical expressions ϕ^{pre} and ϕ^{post} usually refer to some backgound ontology $\Omega \subseteq \mathcal{L}(\Sigma)$.

Definition 5 (Extension of an Input Binding, Renaming). *Let β be an input binding for some Web service capability interface $IF = (i_1, \ldots, i_n)$, V be a set of symbol names and $U \subseteq \mathcal{U}$. A total function $\beta' : \{i_1, \ldots, i_n\} \cup V \rightarrow U$ is called a V-**extension of** β **in** $\mathbf{U}$ if $\beta'(i_j) = \beta(i_j)$ for all $1 \leq j \leq n$.*

*Let π be some function and β an input binding for IF. Then we denote by $rename_{\pi}(\beta)$ the input binding β' for IF' that is derived from β by replacing all pairs $(n, v) \in \beta$ with $n \in dom(\pi)$ by $(\pi(n), v)$. We call $rename_{\pi}(\beta)$ **renaming of** β **by** π.* □

An extension of an input binding β is used in the next definition to ensure that every variable that occurs free in a precondition or postcondition can be assigned a concrete value. Otherwise, no truth-value can be determined for these statements. The renaming represents on a technical level the effect of renaming input names in a Web service interface by the corresponding names in the interface used in the Web service description.

Definition 6 (Capability Satisfaction, Capability Model). *Let $W = (IF, \iota)$ be a Web service in $\mathcal{A}$ and $\mathcal{D} = (\phi^{pre}, \phi^{post}, IF_{\mathcal{D}})$ be a functional description of a Web service. Let FV denote the set of free variables in ϕ^{pre} and ϕ^{post} and U denote $universe(\omega_{rw}(s_0))$. W **satisfies capability** $\mathcal{D}$ in $\mathcal{A}$ if and only if (i) there exists a subset $IF' \subseteq IF_{\mathcal{D}}$ of the inputs of $\mathcal{D}$ and a bijection $\pi : IF \rightarrow IF'$ between IF and IF' such that (ii) for all input bindings $\beta \in In_{\mathcal{A}}(IF)$ and abstract states $s \in \mathcal{S}$: for all FV-extensions β' of $rename_{\pi}(\beta)$ in U: if $\iota(\beta, s) = (s_0, \ldots, s_m)$ for some $m \geq 0$ and $\omega_{rw}(s_0), \beta' \models_{\mathcal{L}(\Sigma)} \phi^{pre}$ then $\omega_{rw}(s_m), \beta' \models_{\mathcal{L}(\Sigma)} \phi^{post}$*

*In this case we write $W \models_{\mathcal{F}} \mathcal{D}$ and call the Web service W a **capability model** (or simply model) of $\mathcal{D}$ in $\mathcal{A}$.* □

Clause (i) essentially requires (interface) compatibility between the Web service and the inputs refered to in Web service description. Note, that we do not require syntactic equality between these names, but only equivalence up to some renaming π. Moreover, it is perfectly fine for models of $\mathcal{D}$ to only use a proper subset IF' of the inputs $IF_{\mathcal{D}}$ mentioned in capability $\mathcal{D}$. Clause (ii) defines the meaning of preconditions and postcondition. Please note, that free variables in these expressions are implicitly universally quantified by our definition.

4 Applying the formal Model for Semantic Analysis

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 [2, 4] that refer to formal descriptions and are based on the *model* notion to our particular context in a meaningful way. For a deeper and extended discussion of the topic, we refer the interested reader to [10].

Realizability. We define *realizability* of a description $\mathcal{D}$ as the corresponding notion to

satisfiability in a logic $\mathcal{L}$: A functional description $\mathcal{D}$ is **realizable in an abstract state space** $\mathcal{A}$ iff. there is a Web service W in $\mathcal{A}$ that satisfies $\mathcal{D}$, i.e. $W \models_{\mathcal{F}} \mathcal{D}$.

Consider the following functional description $\mathcal{D} = (\phi^{pre}, \phi^{post}, IF_{\mathcal{D}})$ describing Web services for account withdraws: $IF_{\mathcal{D}} = \{?acc, ?amt\}$

$$\phi^{pre} : ?amt \geq 0 \qquad \phi^{post} : balance(?acc) = balance_{pre}(?acc) - ?amt$$

At a first glance, the given description seems to be implementable within some Web service W that satisfies $\mathcal{D}$. However, taking a closer look at the respective domain ontology it becomes obvious that this actually is not the case. The ontology defines that a balance might not be negative, but the precondition does not prevent the balance being less then the withdraw. Let's assume that there is a Web service W realizing $\mathcal{D}$. When considering an input binding β with $\beta(?amt) > balance_{pre}(?acc)$, then the precondition is satisfied and thus the postcondition should hold in the final state of the respective execution, i.e. $\omega_{rw}(s_m), \beta \models \forall ?acc.balance(?acc) < 0$. However, this is inconsistent with the domain ontology since $\Omega \models balance(?acc) \geq 0$ and thus s_m can not exist in $\mathcal{A}$. This is a contradiction and shows that no Web service W with $W \models_{\mathcal{F}} \mathcal{D}$ can exist. To fix the description such that it becomes realizable, we need to extend the precondition to $\phi^{pre} : 0 \leq ?amt \wedge ?amt \leq balance(?acc)$.

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. Moreover, we as we will see soon, we can often rephrase the problem of realizability of a description $\mathcal{D} \in \mathcal{F}$ to a well-understood problem in $\mathcal{L}$ for which algorithms already exist. We first turn to an important other notion of which realizability turns out to be a special case (in conformance as with the original notions in mathematical logic).

Functional Refinement. The notion of logical entailment is usually defined as follows: An formula ϕ logically entails a formula ψ iff every interpretation $\mathcal{I}$ which is a models of ϕ (i.e. $\mathcal{I} \models_{\mathcal{L}} \phi$) is also a model of ψ. Substituting interpretations by Web services, formulae by functional descriptions and the satisfaction $\models_{\mathcal{L}}$ by capability satisfaction $\models_{\mathcal{F}}$ we derive a criteria that captures *fuctional refinement*: Let $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{F}$ be functional descriptions. $\mathcal{D}_1$ **is a functional refinement of** $\mathcal{D}_2$ **in** $\mathcal{A}$ (denoted by $\mathcal{D}_1 \sqsubseteq \mathcal{D}_2$) iff. for each Web service W in $\mathcal{A}$, $W \models_{\mathcal{F}} \mathcal{D}_1$ implies $W \models_{\mathcal{F}} \mathcal{D}_2$. 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 $\mathcal{D}_2$. 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 $\mathcal{D}_2$.

For illustration, consider some Web service description $\mathcal{D}_1 = (\phi_1^{pre}, \phi_1^{post}, IF_1)$ with $IF_1 = \{?prs, ?acc\}$ that advertises the ability to provide access credentials for a particular web site ($http://theSolution.com$). A domain ontology specifies that if some web site has some content and someone can access the web site, then he (is able to) know about the content. Furthermore, $http://theSolution.com$ is a web site providing the ultimate answer to life (the universe and everything) and some constant $accessFee$

has a value less then 42.[3]

$$\begin{aligned}
\phi_1^{pre} &: account(?p, ?acc) \wedge balance(?acc) \geq accessFee \\
\phi_1^{post} &: balance(?acc) = balance_{pre}(?acc) - accessFee \\
&\quad \wedge \mathbf{out}(password(?prs, http://theSolution.com)) \\
&\quad \wedge isValid(password(?prs, http://theSolution.com)) \\
\Omega \models &\forall ?ws, ?co, ?prs.\ content(?ws, ?co) \wedge access(?prs, ?ws) \Rightarrow knows(?prs, ?co) \\
&\quad content(http://theSolution.com, answer2Life), accessFee \leq 42 \\
&\quad \forall ?prs, ?ws.\ isValid(password(?prs, ?ws)) \Rightarrow access(?prs, ?ws))
\end{aligned}$$

Using our formal definition we now can examine another definition $\mathcal{D}_2 = (\phi_2^{pre}, \phi_2^{post}, IF_2)$ with $IF_2 = \{?prs, ?acc\}$ and check if it is a functional refinement of the previous description.

$$\phi_2^{pre} : account(?prs, ?acc) \wedge balance(?acc) \geq 100 \quad \phi_2^{post} : knows(?prs, answer2Life)$$

This notion can beneficially be applied within functionality-based matchmaking. For instance, let's assume that a Person me is seeking for the ultimate answer to life ($knows(me, answer2Life)$); me has an account $acc123$ with a current balance of 174 USD. Given this information (and our domain ontology Ω) and considering the specific input binding $\beta(?prs) = me, \beta(?acc) = acc123$, we can infer that any Web service W that is advertised to provide capability $\mathcal{D}_2$ can serve for me's purpose as the precondition ϕ_2^{pre} is satisfied for the input β. In consequence, for the specific input β the service delivers what is described the postcondition ϕ_2^{post}; therefrom, we can infer $knows(me, answer2Life)$. However, since $\mathcal{D}_1 \sqsubseteq \mathcal{D}_2$ we know as well, that any Web service W' that is advertised to provide capability $\mathcal{D}_1$ is perfectly suitable for me and his endeavor as well. The notion of functional refinement can then be used to pre-index some set of Web service description, such that for a given request it is not necessary to consider all available description but only a subset identified by the pre-indexing.

Our framework allows to proof the following theorem (see [10]) , which is especially useful for reducing the problem of determining functional refinement (and eventually all other semantic analysis notions we discuss in this section) to a well-defined proof obligation in the language $\mathcal{L}$ underlying $\mathcal{F}$.

Theorem 1 (Reduction of Functional Refinement from $\mathcal{F}$ to $\mathcal{L}$). *Let $\mathcal{D}_1 = (\phi_1^{pre}, \phi_1^{post}, IF_1)$ and $\mathcal{D}_2 = (\phi_2^{pre}, \phi_2^{post}, IF_2)$ be functional descriptions in $\mathcal{F}$ with the same interfaces, i.e. $IF_1 = IF_2$. Let $[\phi]_{\Sigma_D^{pre} \to \Sigma_D}$ denote the formula ϕ' which can be derived from ϕ by replacing any dynamic symbol $\alpha \in \Sigma_D$ by its corresponding pre-variant $\alpha_{pre} \in \Sigma_D^{pre}$. Then $\mathcal{D}_1 \sqsubseteq_{\mathcal{F}} \mathcal{D}_2$ if $\Omega \cup [\Omega]_{\Sigma_D^{pre} \to \Sigma_D} \models_{\mathcal{L}} ([\phi_2^{pre}]_{\Sigma_D^{pre} \to \Sigma_D} \wedge [\phi_1^{pre}]_{\Sigma_D^{pre} \to \Sigma_D} \wedge \phi_1^{post} \Rightarrow \phi_2^{post})$* □

This gives us the following: If there is an algorithm or an implemented system that allows us to determine logical entailment in $\mathcal{L}$, then we can use the very same

[3] Note that we do not expect such knowledge in one central domain ontology, but a number of knowledge bases (generic, provider- and requester-specific). For simplicity we assume Ω being already aggregated

system or algorithm to determine functional refinement for descriptions of the capability language $\mathcal{F}$, i.e. in principle no new calculus for dealing with $\mathcal{F}$ is needed (at least for the purpose semantic analysis). However, the algorithm which can be derived from Theorem 1 is no longer a heuristic, but *provably correct*. For further discussion, variants and generailzations of the theorem, we refer to [10].

To be able to formulate the next corollary (which is an immediate consequence of the definition of realizability and functional refinement), we use $\bot^{IF}$ to denote a description $D \in \mathcal{F}$ that is trivially unrealizable, i.e. $D = (true, false, IF)$.

Corollary 1 (Realizability vs. Refinement). *A functional description $\mathcal{D} = (\phi^{pre}, \phi^{post}, IF)$ is not realizable iff. $\mathcal{D} \sqsubseteq \bot^{IF}$* □

The corollary simply states that any description which is more specific than the trivially unrealizable functional description must be unrealizable as well. In the light of Theorem 1, it shows that we can reduce realizability of $\mathcal{D}$ to a well-defined proof obligation in $\mathcal{L}$ as well. Hence we can deal with realizability algorithmically based on existing tools.

Omnipotence. For any functional description $\mathcal{D}$ we can consider the dual notion of being not realizable at all, i.e. having every Web service W in $\mathcal{A}$ as a model. This notion corresponds to the classical notion of validity and obviously represents another form of ill-defined or unacceptable type of description. It matches all possible Web services, no matter what they actually do. Service providers could use such (non-trivially) omnipotent descriptions to advertise their Web services in some registry to get maximal visibility. A trivially omnipotent functional description in $\mathcal{F}$ is $\top^{IF} = (true, true, IF)$.

As an immediate consequence we can derive the following corollary which shows that we can reduce omnipotence of $\mathcal{D}$ to a well-defined proof obligation in $\mathcal{L}$ as well and thus deal with it algorithmically based on existing tools:

Corollary 2 (Omnipotence vs. Refinement). *A functional description $\mathcal{D} = (\phi^{pre}, \phi^{post}, IF)$ is omnipotent iff. $\top^{IF} \sqsubseteq \mathcal{D}$* □

The corollary simply states that any description which is more general than the trivially omnipotent functional description must be omnipotent as well.

Summary. Semantic analysis can be seen as both, (i) a concrete example of symbolic computation with functional descriptions that we can formally ground in our formal model, and (ii) as a problem that is interesting in itself. Using our model, we are able to rigorously define various useful notions that enable us to analyze and relate functional descriptions semantically. We have shown that we can reduce the various relevant notions to well-defined proof obligations in the underlying language $\mathcal{L}$ without making severe restrictions or assumptions on that language. Using our framework, we are able to proof the correctness of the reduction. Given the a wealth of different languages that co-exist on the Semantic Web (and the ones that might still be invented), our uniform treatment provides a universal approach to the semantics of functional description independent of the language used.

5 Related Work

By defining the semantics of functional description we provide a basis for applications like semantic Web service repositories and discovery engines (as illustrated in the use case for our formalism and the corresponding examples). Work in this area has previously leveraged a different (less detailed) formal view on the concept of a Web Service: Web services there have been formally considered as *sets of objects* (input, outputs). On a description (language) these sets allow for a natural representation by means of concepts in Description Logics. Matching then has been reduced to standard reasoning tasks in the language [15, 12], however the dynamics associated with a detailed (state-based) perspective on Web services, can not be represented in such a setting. Until recently, it seemed to be a common practice in the Semantic Web Community when considering semantic descriptions of Web service, to strictly focus on languages (e.g. description logics) rather than an adequate (language-independent) mathematical model of the objects of invest igation that underlies such descriptions. The latter question is conceptually interesting and compatible with various concrete representation languages such as Description Logics, First-order Logics, etc. as we have demonstrated in this paper.

In the area of software specification, functional descriptions of are a well studied phenomena. Hoare [5] introduced the approach describing a component by its pre- and post-conditions. Numerous systems have been developed since then [14, 6, 16] that follow the same line of description. They have significant commonalities with our framework, such as constructs for identifying inputs and outputs as well as means to reference symbols in pre-state formulae from the post-state. However our framework is different in two dimensions: (1) we do not fix the underlying language and therefore address the current situation in the Semantic Web with various different languages used in various formalisms, and (2) we explicitly take the existence of background knowledge (represented by some Ontology Ω) and the notion of side effect in the real world modelled into account. In particular, Theorem 1 in Section 4, represents a generalization of a well-known criterion proposed in the software component community for specification matching [18]. The *Guarded Plugin Match* defined by

$$match_{guarded-pm}(\mathcal{D}_1, \mathcal{D}_2) = (\phi_2^{pre} \Rightarrow \phi_1^{pre}) \wedge (\phi_1^{pre} \wedge \phi_1^{post} \Rightarrow \phi_2^{post})$$

is the equivalent to the necessary condition presented in Theorem 1. However, [18] covers a much simpler scenario, where specifications do not contain *dynamic* functions. Futhermore, our criterion explicitly deals with a background ontology Ω on which the functional descriptions $\mathcal{D}_1, \mathcal{D}_2$ are based. In contrast to our work (i.e. Theorem 1), [18] gives no formal investigation of how the criterion called *Guarded Plugin Match* actually relates to the *semantic notion* of functional refinement, which is to be detected by means of a well-defined proof obligation.

6 Conclusions and Future Work

We have defined Abstract State Spaces as a formal model for appropriately describing how Web services act in the world and change it. The main features of the proposed

model are: (i) language independence to a maximum extent, and (ii) modular and flexible definitions that can easily be extended to fit the needs for specific applications.

Language independence, in particular, means that our approach is applicable to a variety of static description language (capturing properties of single states). Thus, it is especially suitable for application in frameworks like OWL-S and WSMO that describe the functionality provided by Web services in a state-based manner. On basis of our model, we have rigorously defined the semantics of functional descriptions. We demonstrated the applicability and benefit of our model in terms of a concrete use case, namely the semantic analysis of functional descriptions. Therein, we have illustrated how to capture several interesting and naturally arising properties of functional descriptions, in particular *functional refinement* and *realizability*. We have given mathematically concise definitions and exemplified how to device a provably correct algorithm for semantic analysis based on existing algorithms and systems. The use case followed throughout the explications supports our thesis: the correctness of any sort of symbolic computation based on functional descriptions of Web services can be analyzed and exposed in our framework.

While this paper presents the basic model, we plan to apply it to frameworks like WSMO and OWL-S that strive for genericity and independence of specific static languages for state descriptions. In particular, we plan to develop a matching mechanism following the defined notion of functional refinement in order to provide a component with clear defined functionality for functional Web service discovery. Furthermore, we consider several extensions of the model, namely integrating *execution invariants* as properties that are guaranteed not to change during execution of a Web service (see [10] for details), the distinction between complete and incomplete functional description (i.e. some sort of closed-world modelling), as well as integrating behavioral descriptions like choreography and orchestration interfaces that are concerned with the intermediate states in order to consume, respectively achieve the functionality of a Web service.

The model presented in this paper can be considered as a first small first step towards an adequate mathematical model for service-oriented architectures. For this, one needs to consider and represent a lot more aspects of the world and its states e.g. multiple agents interacting in a distributed setting and communicating with each other in a concurrent fashion and integrate respective elements in the mathematical model. We expect that the presented model provides a flexible and extensible foundation for such non-standard extensions. Based on a concise and rich model, we will be able to give semantics to formal descriptions of such *architectures* and (similarly to what we discussed for the simple case of capabilities here) to reason about such descriptions in a well-understood and verifiably correct way by extension and refinement of the presented basic model.

Acknowledgements. This material is based upon works supported by the EU within the Knowledge Web Network of Excellence (FP6-507482), the DIP project (FP6-507483), and by the Austrian Federal Ministry for Transport, Innovation, and Technology under the project **RW**^{2} (FFG 809250). The authors would like to thank the members of the WSMO working group (www.wsmo.org) for fruitful input and discussion to the presented work.

References

1. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
2. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, second edition edition, 2000.
3. G. F. and B. P. Introduction to Contextual Reasoning. An Artificial Intelligence Perspective. Technical report, ITC-IRST, Technical Report #9705-19, May 1997.
4. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, second edition edition, 1996.
5. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.
6. C. B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1990.
7. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic Location of Services. In *Proceedings of 2nd European Semantic Web Conference (ESWC)*, pages 1–16, 2005.
8. U. Keller and R. Lara (eds.). WSMO Web Service Discovery. Deliverable D5.1v0.1 Nov 12 2004, WSML Working Group. online: http://www.wsmo.org/TR/.
9. R. Lara, D. Roman, A. Polleres, and D. Fensel. A Conceptual Comparison of WSMO and OWL-S. In *Proc. of the 2nd European Conference on Web Services*, 2004.
10. H. Lausen. Functional Description of Web Services. Deliverable D28.1v0.1 Jan 13 2006, WSML Working Group, 2006. online: http://www.wsmo.org/TR/.
11. H. Lausen, A. Polleres, and D. Roman (eds.). Web Service Modeling Ontology (WSMO). W3C Member Submission 3 June 2005, 2005. online: http://www.w3.org/Submission/WSMO/.
12. L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In *WWW'03*, Budapest, Hungary, May 2003.
13. D. Martin (ed.). OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 November 2004, 2004. online: http://www.w3.org/Submission/OWL-S.
14. B. Meyer. *Eiffel: the Language*. Prentice Hall PTR, 1992.
15. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *ISWC*, pages 333–347. Springer Verlag, 2002.
16. J. Spivey. *The Z Notation, A Reference Manual*. Prentice-Hall International, second edition edition, 1992.
17. J. van Benthem. *Handbook of logic in artificial intelligence and logic programming: epistemic and temporal reasoning*, volume 4, chapter Temporal logic, pages 241–350. Oxford University Press, Oxford, UK, 1995.
18. A. M. Zaremski and J. M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.