Matching Semantic Service Descriptions with Local Closed-World Reasoning

Stephan Grimm[1], Boris Motik[1], and Chris Preist[2]

[1] FZI Research Center for Information Technologies at the University of Karlsruhe
Karlsruhe, Germany
{grimm,motik}@fzi.de
[2] HP Laboratories
Bristol, UK
chris.preist@hp.com

Abstract. Semantic Web Services were developed with the goal of automating the integration of business processes on the Web. The main idea is to express the functionality of the services explicitly, using semantic annotations. Such annotations can, for example, be used for service discovery—the task of locating a service capable of fulfilling a business request. In this paper, we present a framework for annotating Web Services using description logics (DLs), a family of knowledge representation formalisms widely used in the Semantic Web. We show how to realise service discovery by matching semantic service descriptions, applying DL inferencing. Building on our previous work, we identify problems that occur in the matchmaking process due to the open-world assumption when handling incomplete service descriptions. We propose to use autoepistemic extensions to DLs (ADLs) to overcome these problems. ADLs allow for non-monotonic reasoning and for querying DL knowledge bases under local closed-world assumption. We investigate the use of epistemic operators of ADLs in service descriptions, and show how they affect DL inferences in the context of semantic matchmaking.

1 Introduction

Semantic Web Services have been recently proposed as a technology for the automated integration of business processes. The key idea is to represent the functionality of a Web Service explicitly, using so-called *semantic annotations*. These are useful for numerous purposes, such as, for example, *service discovery*—the process of locating Web Services capable of fulfilling a business request.

In the Semantic Web, annotation is a piece of machine-interpretable meta data based on ontological vocabularies formulated by means of an ontology language. The Web Ontology Language (OWL) [20] is a W3C recommendation language for building ontologies in the Semantic Web. As part of the Web Service Modelling Ontology initiative (WSMO), the Web Service Modelling Language (WSML) [4] was recently proposed as an ontology language specifically tuned to annotating Web Services. Certain variants of both languages, namely OWL-DL

and WSML-DL, are based on description logics (DL), a family of knowledge representation formalisms with a clearly defined semantics and well-understood computational properties [3].

Several approaches to service discovery based on description logics have already been proposed in [26, 18, 17]. Along these lines, in this paper we extend our work from [10] and present a DL-based approach for modelling semantics of Web Services. We build on establishing a clear correspondence between the DL modelling primitives and the modeller's intention. In this way, we explain the intuition behind the DL constructs in the service context and thus give guidelines for their application.

Furthermore, we identify problems that occur when DL inferencing is applied to matching semantic service annotations. Namely, DLs are monotonic logics with open-world semantics: the inability to prove a fact does not imply its contrary 'by default'. This often requires a modeller to *overspecify* a situation and to include information that humans take for granted by common-sense. Sometimes, it is not even possible to completely specify semantic service annotations without making some default assumptions. Our analysis shows that the lack of common-sense information in the domain model or in service annotations leads to false matches, thus significantly degrading the quality of the discovery platform.

To address these deficiencies in a systematic way, we propose to base service discovery on a non-monotonic logical formalism that allows for local closed-world reasoning by referring to facts which are *explicitly known*. This compensates for imprecision in domain ontologies and service annotations.

Numerous non-monotonic formalisms have already been developed, such as default logic, circumscription or various extensions of logic programming with negation-as-failure [2]. However, we base our service discovery framework on an autoepistemic extension to description logic (ADL) from [5], namely the logic $\mathcal{ALCK}$. ADLs are proper extensions of description logics, so the same principles can be applied to obtain autoepistemic extensions of OWL-DL or WSML-DL. Moreover, the reasoning algorithm from [5] extends the well-known tableau algorithm implemented in DL reasoning systems, such as RACER [11], FaCT [13] or Pellet [25]. Therefore, we believe that ADLs are a good fit with the existing technological Semantic Web infrastructure. To verify the practicability of our approach for matching semantic descriptions, we have implemented a simple ADL reasoner, as a testing environment to verify our examples for service discovery.

2 The Service Discovery Problem

We introduce the problem of service discovery by means of an example taken from the travelling domain. Let us assume that a company needs to frequently book business trips for its employees. To stay competitive, for any single booking this company wants to contact several travel agencies and pick the one providing the best offer. In such a business transaction, the company plays the role of the *requester* and the travel agencies play the role of *providers* of a travelling service.

In order to allow this process to be automated, the electronically available travel agencies provide access to their booking services via Web Service interfaces. Furthermore, both the requester company and the travel agencies need to specify the functionality of the services they request or provide in a declarative way, using semantic annotations.

In [21] the notion of a *concrete service* has been introduced, which represents a particular business transaction. An example of a concrete service, offered by some travel agency A, is 'selling a flight ticket from Frankfurt to London at a particular date and time for 50 Euro'. However, A also provides other concrete services, which vary depending on the cities, date or price. The set of all concrete services is approximated as an *abstract service* [21]. For example, A might advertise "selling flight tickets between cities in Europe". Similarly, another travel agency B advertises an abstract service for 'selling flight tickets from Europe to the US', which includes concrete services such as 'selling a flight ticket from Frankfurt to New York'.

In the same way, the business needs of the requester company correspond to concrete services, such as 'selling a ticket from Frankfurt to London for November 5^{th}'. Similar to the agencies, the company summarises all intended concrete services in an abstract service, such as 'selling a ticket from Germany to the UK'.

We introduce the notion of *capability description* as a formal specification, used by requesters and providers, to represent an abstract service. In their capability descriptions the requester company and the travel agencies intend to capture the set of all concrete services they are willing to accept. Here, capability descriptions are expressed informally, however, in Section 4 we show how to express them in a formal language to make them machine-processable. Our capability descriptions are similar in functionality to WSMO Web Service capabilities [16] or OWL-S service profiles [1]. However, they are different in that they base on an abstract ontological description of service functionality rather than on a state transition model with pre- and postconditions.

In [21], the process of selecting a service to fulfil a request is split into two consecutive phases. The *service discovery* phase is concerned with the identification of abstract services relevant for the request. This is done by matching the capability description of the requester to the capability descriptions of providers, to determine whether they are compatible. In Section 4 we show how description logic inferences can be used for this purpose. In this sense, service discovery is based on the capability descriptions of requesters and providers, and does not involve information that is obtained by invoking any Web Service.

The service discovery phase is followed by the *service definition* phase, where the set of potential providers is further narrowed, and the concrete service to be performed is specified in detail. As discussed in [14], this process often includes negotiation and requires information which is not captured in the capability descriptions for abstract services (such as preferences or additional business constraints that are not publicly available). In this paper we focus on the service discovery phase, and leave the service definition phase to our future work.

3 Description Logics and their Autoepistemic Extension

In this section we describe an autoepistemic extension to the description logic (DL) formalism that we will use throughout the paper. We start with an intuitive view on the basic DL $\mathcal{ALC}$ and its autoepistemic extension $\mathcal{ALCK}$. Then we revisit the formal syntax and semantics of $\mathcal{ALCK}$ and introduce epistemic queries and the satisfiability of an $\mathcal{ALCK}$ concept w.r.t. an $\mathcal{ALC}$ knowledge base.

Description Logics

DLs [3] are a family of knowledge representation formalisms that provide the formal underpinning of certain ontology languages for the Semantic Web, such as WSML-DL [4] or OWL-DL [20]. The basic syntax elements of DLs are *concepts*, such as *City* or *Airplane*, *roles*, such as *transportationMeans* or *from*, and *individuals*, such as *Frankfurt* or *Airbus380*. Primitive concepts can be combined into complex concepts using concept constructors. In this paper, we consider the basic DL $\mathcal{ALC}$, which provides the propositional connectives and restricted existential and universal role quantification. For example, a complex concept $Journey \sqcap \exists from.UKCity \sqcap \forall transpMeans.\neg Airplane$ intuitively represents a journey from somewhere in the UK by a transportation means different from an airplane.

A DL knowledge base consists of axioms and is split into a TBox and an ABox. *Concept inclusion axioms* in the TBox state subset relationship between concepts; for example, $Airplane \sqsubseteq Vehicle$ states that airplanes are kinds of vehicles. *Assertion axioms* in the ABox describe the state of the world; for example, $UKCity(London)$ states that London is a city in the UK, and $train(Berlin, Hamburg)$ states that Berlin is connected by train to Hamburg.

Autoepistemic Description Logics

Autoepistemic logic is a formalism concerned with the notion of 'knowledge' and allows introspection of knowledge bases—that is, asking what a knowledge base *knows*. In [5], the basic DL $\mathcal{ALC}$ has been extended by the autoepistemic knowledge operator $\mathbf{K}$, yielding the autoepistemic description logic $\mathcal{ALCK}$. The $\mathbf{K}$-operator can be applied as a constructor to both concepts and roles, and can intuitively be paraphrased as 'known to be'.

To understand the intuition behind the $\mathbf{K}$-operator, consider the knowledge base $KB = \{City(Frankfurt), train(Frankfurt, Paris)\}$, and the concept $D = City \sqcap \exists train.\neg GermanCity$, which can be paraphrased as 'cities which are connected by train to some city outside Germany'. Since KB does not say whether *Paris* is a German city or not, *Frankfurt* is not in the extension of D. On the contrary, consider the autoepistemic concept $D' = City \sqcap \exists \mathbf{K} train.\neg \mathbf{K} GermanCity$, which can be intuitively paraphrased as 'cities which are *known* to be connected by train to something which is not *known* to be a German city'. Based on the facts in KB, we cannot derive that *Paris* is a German city. Therefore, *Paris* is *not known* to be a German city, and thus *Frankfurt* is in the extension of D'.

These autoepistemic extensions allow for local closed-world reasoning [7] and a logical reconstruction of non-monotonic features of frame-based knowledge representation systems, such as concept and role closure, defaults, integrity constraints and procedural rules [23]. In Section 5 we apply local closed-world reasoning to the matching of service capability descriptions.

The Language $\mathcal{ALCK}$

We now formally introduce the syntax and semantics of $\mathcal{ALCK}$ [5]. The following rules define the syntax of this language, where C, D denote concepts, A denotes a primitive concept, r denotes a role and p denotes a primitive role:

$$\begin{array}{rcl} C, D & \longrightarrow & A \mid \top \mid \bot \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \forall r.C \mid \exists r.C \mid \mathbf{K}C \\ r & \longrightarrow & p \mid \mathbf{K}p \end{array}$$

An *epistemic interpretation* is a pair $(\mathcal{I}, \mathcal{W})$ where $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a *first-order interpretation* with interpretation domain $\Delta^{\mathcal{I}}$ and interpretation function $\cdot^{\mathcal{I}}$, and $\mathcal{W}$ is a set of first-order interpretations, seen as possible worlds. The following equations define how the syntax elements of $\mathcal{ALCK}$ are epistemically interpreted.

$$\begin{array}{rcl} \top^{\mathcal{I},\mathcal{W}} = \Delta^{\mathcal{I}} \quad , \quad \bot^{\mathcal{I},\mathcal{W}} = \emptyset & & \\ A^{\mathcal{I},\mathcal{W}} = A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \quad , \quad p^{\mathcal{I},\mathcal{W}} = p^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} & & \\ (C \sqcap D)^{\mathcal{I},\mathcal{W}} = & C^{\mathcal{I},\mathcal{W}} \cap D^{\mathcal{I},\mathcal{W}} & \\ (C \sqcup D)^{\mathcal{I},\mathcal{W}} = & C^{\mathcal{I},\mathcal{W}} \cup D^{\mathcal{I},\mathcal{W}} & \\ (\neg C)^{\mathcal{I},\mathcal{W}} = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I},\mathcal{W}} & \\ (\forall r.C)^{\mathcal{I},\mathcal{W}} = & \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in r^{\mathcal{I},\mathcal{W}} \rightarrow b \in C^{\mathcal{I},\mathcal{W}}\} & \\ (\exists r.C)^{\mathcal{I},\mathcal{W}} = & \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in r^{\mathcal{I},\mathcal{W}} \wedge b \in C^{\mathcal{I},\mathcal{W}}\} & \\ (\mathbf{K}C)^{\mathcal{I},\mathcal{W}} = & \bigcap_{\mathcal{J} \in \mathcal{W}} C^{\mathcal{J},\mathcal{W}} & \\ (\mathbf{K}r)^{\mathcal{I},\mathcal{W}} = & \bigcap_{\mathcal{J} \in \mathcal{W}} p^{\mathcal{J},\mathcal{W}} & \end{array}$$

Primitive concepts are interpreted as subsets of $\Delta^{\mathcal{I}}$, and primitive roles are interpreted as subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The boolean connectives and existential and universal role quantification are interpreted in terms of set operations on $\Delta^{\mathcal{I}}$, as in $\mathcal{ALC}$ [3]. An epistemic concept $\mathbf{K}C$ is interpreted as the set of all individuals which belong to the concept C in all first-order interpretations in $\mathcal{W}$, i.e. in all possible worlds. Thus, applying $\mathbf{K}$ to concept C produces the set of objects that are members of C in all possible worlds; in other words, these objects are definitely *known to be* members of C. Similarly, an epistemic role $\mathbf{K}p$ is interpreted as the pairs of individuals that belong to the role p in all possible worlds.

An epistemic interpretation satisfies an inclusion axiom $C \sqsubseteq D$ if $C^{\mathcal{I},\mathcal{W}} \subseteq D^{\mathcal{I},\mathcal{W}}$, and it satisfies an assertion axiom $C(a)$ or $r(a,b)$ if $a^{\mathcal{I}} \in C^{\mathcal{I},\mathcal{W}}$ or $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I},\mathcal{W}}$, respectively. An *epistemic model* for an $\mathcal{ALCK}$ knowledge base *KB* is a maximal non-empty set $\mathcal{W}$ of first-order interpretations such that, for each $\mathcal{I} \in \mathcal{W}$, the epistemic interpretation $(\mathcal{I}, \mathcal{W})$ satisfies all axioms in *KB*. The maximality condition for $\mathcal{W}$ ensures that there is no other first-order interpretation $\mathcal{I} \notin \mathcal{W}$ which also satisfies all the axioms in *KB*. In this way, the $\mathbf{K}$-operator allows to refer to definitely known facts by intersecting *all* possible worlds of *KB*.

Epistemic Queries and Concept Satisfiability

In this paper we assume that KB does not contain occurrences of the **K**-operator; that is, KB is an $\mathcal{ALC}$ knowledge base. Then, KB has at most one epistemic model $\mathcal{M}(KB)$, comprising all its first-order models. An *epistemic query* [5] over an $\mathcal{ALC}$ knowledge base KB is an $\mathcal{ALCK}$ concept assertion of the form $C(a)$. We say that KB entails $C(a)$, written $KB \models C(a)$, if, for every first-order interpretation $I \in \mathcal{M}(KB)$, the epistemic interpretation $(\mathcal{I}, \mathcal{M}(KB))$ satisfies $C(a)$.

A tableaux calculus for answering epistemic queries has been presented in [5]. However, in Section 5 we require checking satisfiability of epistemic concepts with respect to a knowledge base KB, which we define next. This inference can be performed by a straightforward extension of the calculus from [5].

Definition 1 (Concept Satisfiability). *For a satisfiable $\mathcal{ALC}$ knowledge base KB, an $\mathcal{ALCK}$ concept C is satisfiable w.r.t. KB if there is a first-order interpretation $I \in \mathcal{M}(KB)$ such that $C^{(I,\mathcal{M}(KB))} \neq \emptyset$.*

4 Modelling Service Capabilities in Description Logics

We now show how to use description logics to model capability descriptions for services. In particular, we focus on mapping the notions introduced in Section 2 into the description logic framework, based on our previous work in [10]. Furthermore, we identify incomplete capability descriptions as a key problem for service discovery.

4.1 From Concrete Services to Capability Descriptions

We map a concrete service, representing a specific business transaction, to the relational structure in a first-order interpretation $\mathcal{I}$. Such an interpretation can be understood as a directed labelled graph, which represents various properties of services. For example, the bottom left part of Figure 1 shows a relational structure which corresponds to a concrete service for travelling between Frankfurt and London on an Airbus 380.

We express a capability description using a DL concept. Under a first-order interpretation, such a concept is mapped to a set of individuals. Thus, concepts provide a natural way of modelling sets of concrete services. For example, at the top of Figure 1 we show a capability description S, which describes 'travelling between EU cities'. In $\mathcal{I}$, this concept is interpreted as a set $S^{\mathcal{I}}$ of individuals, representing the concrete services accepted by S. Since the service description does not specify the actual cities, $S^{\mathcal{I}}$ contains concrete services for different pairs of cities, such as Frankfurt and London, or Berlin and Hamburg.

Capability descriptions usually refer to commonly used domain ontologies. These ontologies define the background knowledge in a certain domain of interest in form of DL axioms. For example, in the travelling domain, they define terms such as 'City', 'Journey' or 'Airplane'.

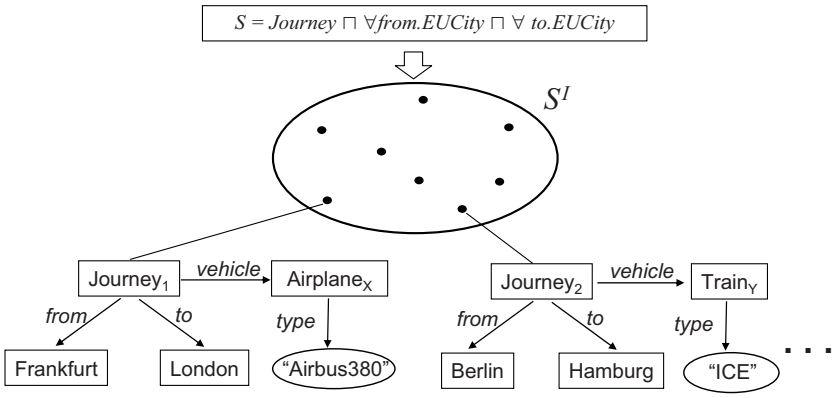


Fig. 1. A Capability Description Specifying Several Concrete Services

4.2 Variance and Incompleteness in Capability Descriptions

Recall from Section 2 that the main purpose of a capability description is to describe a set of concrete services which vary on several parameters. Hence, we say that capability descriptions introduce *variance due to intended diversity* [10], which manifests itself by allowing the capability description to specify several concrete services, each having different parameter values. The fact that the capability description S allows concrete services for travelling between Frankfurt and London, and Hamburg and Berlin, is an example of variance due to intended diversity. Using DL concept expressions as a description technique allows us to express this variance in a compact form, without listing all possible pairs of cities explicitly.

Moreover, we also identify *variance due to incomplete knowledge* [10], which is caused by the fact that capability descriptions do not completely specify all parameters. For example, a travel agency might not explicitly specify the types of payment it is willing to accept. This detail may be off-loaded from the service discovery to the service definition phase. However, this does not mean that a concrete service would not have any payment information; it simply means that the type of payment has not been specified. Each concrete service will still contain a certain type of payment. Variance due to incomplete knowledge is captured by assuming different *possible worlds*. In each of these possible worlds unspecified information is resolved in a particular way. In DL, variance due to incomplete knowledge is reflected by the fact that a knowledge base can have several different first-order interpretations, each corresponding to a particular possible world. Notice that this actually coheres to open-world semantics in description logics.

In Section 5 we show how epistemic operators can be incorporated into capability descriptions in order to close off parts of the domain model and to control and reduce variance due to incomplete knowledge by ruling out some of the possible worlds.

4.3 Matching Capability Descriptions

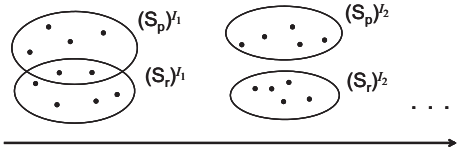
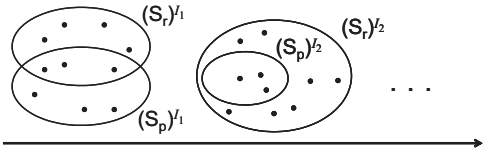
We now define a matching function $match(KB, S_r, S_p)$, which returns *true* if the capability descriptions S_r and S_p match, *false* otherwise. The basic idea behind matching is to check if two capability descriptions, issued by a requester and a provider, respectively, specify any common concrete service [26]. Such concrete services might then be taken as a basis to enter into the service definition phase.

Technically, matching is reduced to checking the non-emptiness of the intersection of both capability descriptions. However, when performing this check, there are two ways to resolve variance due to incomplete knowledge, as shown in [10]. The first one is to check if the intersection is non-empty in *some* possible world, as presented in the upper part of Table 1. In other words, we check if there is a way to resolve incompleteness in the capability descriptions such that they specify a common concrete service.

Another possibility is to check if the intersection of concept extensions is non-empty in *each* possible world, as shown in the lower part of Table 1. This is a stronger check: regardless of how we resolve incompleteness in capability descriptions, we need a concrete service that is common to both descriptions.

Related approaches to service discovery use similar matching techniques. 'Satisfiability of concept conjunction' was first proposed in [8, 26] and [24], and was subsequently considered in [19, 18, 17, 14, 10]. Furthermore, many of these works use 'entailment of concept subsumption', which checks if one of the sets of accepted concrete services is a subset of the other in each possible world. How-

Table 1. Using DL Inferences for Matching Service Capabilities

Inference:	*Satisfiability of Concept Conjunction*
Function:	$match_{int}(KB, S_r, S_p)$
Formula:	$S_r \sqcap S_p$ is satisfiable w.r.t. KB
Situation:	
Intuition:	Is there a way to resolve unspecified details such that S_r and S_p specify some common concrete service?
Inference:	*Entailment of Concept Non-Disjointness*
Function:	$match_{ndj}(KB, S_r, S_p)$
Formula:	$KB \cup \{S_r \sqcap S_p \sqsubseteq \bot\}$ is unsatisfiable
Situation:	
Intuition:	Do S_r and S_p specify some common concrete service, regardless of how unspecified details are resolved?

ever, as discussed in [10], this inference has not shown to be beneficial for our notion of compatibility between two descriptions S_r and S_p: we treat concrete services as alternative specifications of service parameters, and thus, having a single concrete service in the extension of the sets is already sufficient for our capability descriptions to be compatible. Subsumption and equivalence matching applied to pairs (S_r, S_p) of descriptions has been used for establishing a ranking among service providers in [19, 17, 14]. However, in our setting the partial subsumption check between two provider descriptions S_{p_A} and S_{p_B}, defined in [10], provides a more fine-grained ranking based on the options the requester has later on in the service definition phase. In this work, we do not consider ranking but focus on the characteristics of the matching inferences in a local closed-world setting.

In [15], and also partly in [14], the authors consider a different description approach based on specifying services in terms of state transitions. They base matching on transaction logic, a formalism capable of explicitly representing changes in the world. In this way, they do not only consider the discovery phase, but also address partly the service definition phase.

4.4 Problems in Matching Capability Descriptions

Both ways of matching, $match_{int}$ and $match_{ndj}$, cause problems in certain cases [10], which we illustrate next on our running example. Let us assume that the requester company asks for a flight from a city in the UK, and that two providers A and B offer flights from cities in the EU and the US, respectively.

Example 1 (Problems with Matching Service Descriptions).

$$KB = \{\ UKCity \sqsubseteq EUCity,\ \ Flight \sqsubseteq \exists\, from\,.\top\ \}$$

$$\begin{aligned} S_r &= Flight \sqcap \forall\, from\,.UKCity \\ S_{p_A} &= Flight \sqcap \forall\, from\,.EUCity \\ S_{p_B} &= Flight \sqcap \forall\, from\,.USCity \end{aligned}$$

First, consider matching capability description S_r against S_{p_A}. Since the requester asks for a flight from a UK city, and A offers flights from an EU city, we intuitively expect the two descriptions to match. However, by applying the DL inferences, we get that $match_{int}(KB, S_r, S_{p_A}) = true$, but $match_{ndj}(KB, S_r, S_{p_A}) = false$. In the second case, the unintuitive result is due to the fact that we never specified that A actually offers any services. Hence, there is a way to resolve this incompleteness in the specification by choosing a possible world in which the extension of S_{p_A} is empty. Therefore, matching fails, since it is not the case that the intersection of S_r and S_{p_A} is non-empty in each possible world.

Second, consider matching capability description S_r against S_{p_B}. Since the requester asks for flights from UK cities, but B offers flights from US cities, we would expect matching to fail. However, by applying the DL inferences, we get that $match_{int}(KB, S_r, S_{p_B}) = true$, but $match_{ndj}(KB, S_r, S_{p_B}) = false$. In the first case, the unintuitive result is due to the fact that we never said that UK

and US cities are disjoint. Therefore, matching succeeds, since there is a possible world in which some city is in the extension of both *UKCity* and *USCity*.

Both of these problems principally arise from the existence of unwanted possible worlds. To reduce the number of unintuitive matches, it would be desirable to reduce the variance due to incomplete knowledge, and to rule out those possible worlds which are 'obviously' wrong.

In the first case, this could be achieved by adding the assertion $\mathit{Flight}(a)$, for some new individual a, to the knowledge base before matching is performed, ruling out possible worlds in which *Flight* is empty.

In the second case, the false positive match with $match_{int}$ could be 'repaired' by adding the disjointness axiom $\mathit{EUCity} \sqcap \mathit{USCity} \sqsubseteq \bot$ to the knowledge base, eliminating possible worlds in which a city can be in both the EU and the US.

In any case, we would have to include additional facts, such as disjointness constraints, which in practice often has the drawback of overloading the specification with 'obvious' information. In general, domain ontologies in the Semantic Web cannot be expected to contain such additional information, since they are reusable domain vocabularies and different ontologies might have been developed for different purposes.

As we shall see in the following section, non-monotonic features and local closed-world reasoning allow us to address this important problem of *overspecification* by dealing with incompleteness in an alternative way. A pure closed-world system, on the other hand, would not equally support the desired variance. Since in [10] we identified other problems with successfully using $match_{ndj}$ when several restrictions on roles are combined, we will focus on $match_{int}$, which has already been applied in an industrial logistics scenario in [22] on service descriptions in OWL-DL.

5 Epistemic Operators in Capability Descriptions

The autoepistemic extension to DL provides a means to exclude unwanted first-order interpretations in a controlled way. In this section, we show how the problems described in Section 4.4 can be overcome by realising local closed-world reasoning using the **K**-operator in capability descriptions.

5.1 Locally Closing Worlds in Capability Descriptions

Description logics employ the *open-world* semantics, under which, if a fact is not derivable from the knowledge base, its contrary cannot be assumed 'by default'. This is considered appropriate for the Semantic Web, due to its open nature. However, in a controlled scenario such as service discovery, it is sometimes beneficial to assume that all relevant facts about a subset of the domain are known; this is known in the literature as *local closed-world assumption* [7, 12]. The local closed-world assumption can be applied to DL knowledge bases in form of *concept closure* and *role closure* [23], which enable us to assume that all individuals of a concept, or all pairs of individuals of a role are known.

Concept Closure
The **K**-operator can be used to restrict the extension of a concept to those individuals that belong to this concept in each possible world, which is denoted by *concept closure*. Recall from the definition of the semantics of $\mathcal{ALCK}$ in Section 3 that an expression $\mathbf{K}C$ is interpreted as the intersection of extensions over all first-order interpretations. Intuitively, this can be paraphrased by 'the set of individuals that are *known* to belong to C'. The following example extends Example 1 by applying the pattern of concept closure to city concepts.

Example 2 (concept closure).

$$KB = \{\ UKCity \sqsubseteq EUCity,\ Flight \sqsubseteq \exists from.\top,\ UKCity(London)\ \}$$

$$\begin{aligned} S_r &= Flight \sqcap \forall from.\mathbf{K}\,UKCity \\ S_{p_A} &= Flight \sqcap \forall from.\mathbf{K}\,EUCity \\ S_{p_B} &= Flight \sqcap \forall from.\mathbf{K}\,USCity \end{aligned}$$

The $\mathcal{ALC}$ knowledge base *KB* states that every UK city is also an EU city, that any flight must specify the property *from* and that the individual *London* is an explicitly asserted UK city. The $\mathcal{ALCK}$ concepts S_r, S_{p_A} and S_{p_B} are the service capability descriptions issued by a requester and providers A and B. The requester requires a flight from somewhere in the UK, whereas the providers advertise flights from EU and US locations, respectively. In contrast to Example 1, in this example all parties use the pattern of concept closure to restrict the property *from* to only those individuals that are known to be cities in the UK, the US or Europe, respectively.

We use the extended notion of concept satisfiability from Definition 1 in Section 3 to check satisfiability of the $\mathcal{ALCK}$ concept $S_r \sqcap S_p$ w.r.t. the $\mathcal{ALC}$ knowledge base *KB* in $match_{int}$. As we show next, the application of $match_{int}$ in Example 2 yields the intuitively desired matching behaviour.

First, consider matching the capability description S_r against S_{p_A} using $match_{int}$. The satisfiability of $S_r \sqcap S_{p_A}$ requires the existence of an individual which is both known to be a *UKCity* and known to be a *EUCity*. The individual *London* is explicitly stated to be a *UKCity* in *KB*, so it is known to be a *UKCity*. This individual is also known to be a *EUCity* because of the inclusion axiom in *KB*. Hence, $S_r \sqcap S_{p_A}$ is satisfiable w.r.t. *KB* and thus $match_{int}(KB, S_r, S_{p_A}) = true$. Notice that without the explicitly introduced individual *London*[3] this satisfiability would not hold because there would be no individual which meets the above mentioned conditions in each possible world.

Second, consider matching the capability description S_r against S_{p_B} using $match_{int}$. The satisfiability of $S_r \sqcap S_{p_B}$ requires the existence of an individual which is both known to be a *UKCity* and known to be a *USCity*. However, there is no such individual and therefore $match_{int}(KB, S_r, S_{p_B}) = false$. Of course there are first-order interpretations in which *UKCity* and *USCity* have common

[3] The individual *London* here can be seen as a representative for all explicitly modelled cities in some domain ontology with a geographic context.

individuals, namely, some in which *London* is both a *UKCity* and a *USCity*, but for such individuals this is not the case in each possible world.

Role Closure

The **K**-operator can also be used to restrict the extension of a role to those pairs of individuals that are connected by this role in each possible world, which is denoted as *role closure*. Recall from the definition of the semantics of $\mathcal{ALCK}$ in Section 3 that an expression $\mathbf{K}r$ is interpreted as the intersection of role extensions over all first-order interpretations. Intuitively, this can be paraphrased by 'all pairs of individuals that are *known* to be connected by r'. The following example applies the pattern of role closure to a role *train* that denotes the connection of two cities via the continental train network.

Example 3 (role closure).

$$\begin{aligned} KB = \{\ & GermanCity \sqsubseteq EUCity,\ UKCity \sqsubseteq EUCity,\ Flight \sqsubseteq \exists\, from\,.\top, \\ & UKCity(London)\ , GermanCity(Berlin)\ , GermanCity(Hamburg)\ , \\ & train(Berlin, Hamburg)\ , train(Hamburg, Berlin) \qquad \} \end{aligned}$$

$$\begin{aligned} S_r &= Flight \sqcap \forall\, from\,.(\mathbf{K}\, EUCity \sqcap \exists\, \mathbf{K}\, train\,.\top) \\ S_{p_A} &= Flight \sqcap \forall\, from\,.\mathbf{K}\, GermanCity \\ S_{p_B} &= Flight \sqcap \forall\, from\,.\mathbf{K}\, UKCity \end{aligned}$$

The $\mathcal{ALC}$ knowledge base *KB* states that both German cities and UK cities are cities in the EU and that the individuals *London*, *Berlin* and *Hamburg* are explicitly stated to be such cities. Furthermore, *KB* states that the individuals *Berlin* and *Hamburg* are connected via the train network. In the $\mathcal{ALCK}$ concept S_r the requester requires a flight from a known EU city which is known to be connected to the train network. This is achieved by applying concept closure to the concept *EUCity* and role closure to the role *train*. The providers advertise travelling from locations in Germany and in the UK, respectively. Also Example 3 shows the intuitively expected matching behaviour.

First, consider matching the capability description S_r against S_{p_A} using $match_{int}$. There are two known German cities, *Berlin* and *Hamburg*, which are both known to have connection to the train network. Furthermore, they both are also known to be EU cities due to the inclusion axiom. Therefore, $match_{int}(KB, S_r, S_{p_A}) = true$ and provider A matches the request.

Second, consider matching the capability description S_r against S_{p_B} using $match_{int}$. The only known UK city is *London* but it is not known to be connected to the train network. Therefore, $match_{int}(KB, S_r, S_{p_B}) = false$ and provider B fails to match the request, although *London* is known to be a EU city. Due to the local closure of worlds, only the known train connections explicitly modelled in the domain knowledge are taken into account.

Without the **K**-operator applied to the role *train* in S_r, provider B would also match the request because, due to the open-world assumption, *London* would have train connection in some possible world. Alternatively to the usage of **K**, one would have to complete the specification by listing all the cities which have no connection to the continental train network, by axioms like $\forall\, train\,.\bot(London)$.

5.2 Benefits of Locally Closing Worlds

By using the **K**-operator in capability descriptions to locally close off worlds, we have excluded unwanted first-order interpretations (that is, possible worlds), reducing variance due to incomplete knowledge. In this way, we have avoided overspecifying the domain, but have succeeded in removing false positive matches.

In Example 2, there is no need to explicitly state disjointness between non-related city concepts, since by the use of **K** we restrict their extensions to known cities only, for which EU and US do not overlap. In general, adding disjointness constraints is no real solution to the problem, since not all imaginable cities can be covered in the specification. A requester or provider might introduce a new city concept which is not explicitly related to the existing city concepts in any ontology. Intuitively, we would not want this unrelated city concept to match against any other. Hence, we avoid such additional constraints by locally closing off city concepts, assuming full knowledge about this part of the world.

In Example 3, we avoid to list all the cities that are *not* connected to the train network in addition to those that are. Here the use of **K** allows us to handle a partial and incomplete description of the state of the world by closing off the role *train*, assuming full knowledge about all train connections between cities.

Since the use of **K** makes matching dependent on the state of the world, it should only be applied to concepts or roles for which there is some ABox information present. For example, when requesting a flight carried out by a Star Alliance partner, **K** would most likely be applied as follows: $S = \mathit{Flight} \sqcap \forall\, \mathit{carriedOutBy}.\mathbf{K}\mathit{StarAlliancePartner}$. For the discovery system, assertions of air carriers to the Star Alliance, such as $\mathit{StarAlliancePartner}(\mathit{Lufthansa})$, is static ABox information present in some domain ontology. In combination with other such information about the state of the world, like $\mathit{GermanCarrier}(\mathit{Lufthansa})$, this request would match a provider that offers flights carried out by German carriers. Thus, **K** can safely be applied to *StarAlliancePartner* in S, preventing the specification from being overloaded with information about which airlines are no such partners. On the contrary, in settings similar to those from our Examples, no information about concrete flights and their carriers occurs in the domain knowledge. Domain ontologies rather speak of flights in more general terms, using TBox information such as $\mathit{Flight} \sqsubseteq \neg \mathit{ShipCruise}$ to distinguish them from other forms of travelling. Therefore, **K** is not applied to *Flight* or to *carriedOutBy* in S. The discovery system benefits from leaving this part of the world open, not requiring travel agencies to list all the concrete flights they offer.

5.3 An Implementation of the Matchmaking Framework

We have verified these examples with our prototypical implementation of a reasoner for $\mathcal{ALCK}$ according to the calculus presented in [5]. Based on this calculus, we implemented a decision procedure for satisfiability of $\mathcal{ALCK}$ concepts w.r.t $\mathcal{ALC}$ knowledge bases as well as for epistemic query answering. It can be used as a testing environment for small examples[4].

[4] The implementation is available at `http://www.fzi.de/downloads/wim/KToy.zip`

6 Summary and Outlook

In this paper, we have described a DL-based framework for discovery of services in the Semantic Web. We have presented an intuitive way to map the business needs of requesters and providers to the formal DL constructs. Thus, we have provided a basis for modelling guidelines which meet well the modeller's intuition. We have identified problems of DL-based matching related to open world semantics. We have shown how an autoepistemic extension to DL can be used to overcome those problems. In particular, we have shown how the application of epistemic operators in service capability descriptions can be used to realise local closed-world reasoning in a controlled way, preventing overspecification of capability descriptions and domain ontologies. We also implemented a testing environment for reasoning with $\mathcal{ALCK}$ concepts together with $\mathcal{ALC}$ knowledge bases, in order to verify our examples.

We plan to investigate the extension of $\mathcal{ALCK}$ with features of expressive description logics, such as number restrictions, nominals or inverse roles, which proofed to be useful in the context of describing service semantics [17, 10]. We also intend to investigate reasoning with arbitrary $\mathcal{ALCK}$ knowledge bases and to explore the formalism presented in [6, 23], which introduces an additional epistemic operator **A**, capturing the notion of 'assumption'. This formalism allows for the whole range of non-monotonic features, such as default rules and integrity constraints, which we have applied in a Semantic Web context in [9]. We intend to incorporate these features into our discovery framework to further improve the matching of capability descriptions. Moreover, we plan to systematise the use of epistemic operators in service capability descriptions to obtain intuitive modelling constructs that abstract from the underlying logical formalism.

References

1. The OWL Service Coalition. OWL-S 1.1 release. Available at http://www.daml.org/services/owl-s/1.1/, November 2004.
2. Grigoris Antoniou. *Nonmonotonic Reasoning*. MIT Press, 1997.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, January 2003.
4. J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The Web Service Modeling Language WSML: An Overview. In *Proceedings of the 3rd European Semantic Web Conference (ESWC)*, 2006.
5. F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An Epistemic Operator for Description Logics. *Artificial Intelligence*, 100(1-2):225–274, 1998.
6. F. M. Donini, D. Nardi, and R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.
7. O. Etzioni, K. Golden, and D. Weld. Tractable Closed World Reasoning with Updates. In *Proceedings of the 4th International Conference on Knowledege Representation and Reasoning (KR1994)*, pages 178–189. Morgan Kaufmann, 1994.
8. J. Gonzlez-Castillo, D. Trastour, and C. Bartolini. Description Logics for Matchmaking of Services. In *Proc. of the KI-2001 Workshop on Appl. of DL*, 2001.

9. S. Grimm and B. Motik. Closed-World Reasoning in the Semantic Web through Epistemic Operators. In *CEUR Proceedings of the OWL Experiences and Directions Workshop*, Galway, Ireland, 2005.
10. S. Grimm, B. Motik, and C. Preist. Variance in e-Business Service Discovery. In *Proceedings of the 1st Int. Workshop SWS'2004 at ISWC 2004*, November 2004.
11. V. Haarslev and R. Möller. Description of the RACER System and its Applications. In *International Workshop on Description Logics*, 2001.
12. J. Heflin and H. Munoz-Avila. LCW-based Agent Planning for the Semantic Web. In *Proc.of AAAI Workshop on Ontologies and the Semantic Web(WS-02-11)*, 2002.
13. I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proceedings of the 6th International Conference on Knowledege Representation and Reasoning (KR1998)*, pages 636–645. Morgan Kaufmann, 1998.
14. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic Location of Services. In *Proc. of the 2nd European Semantic Web Conference(ESWC)*, 2005.
15. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *Proceedings of the 1st International Workshop SWS'2004 at ISWC 2004*, November 2004.
16. H. Lausen, D. Roman, and U. Keller. Web service Modeling Ontology - Standard (WSMO-Standard). Working draft. Technical report, Digital Enterprise Research Institute (DERI), March 2004. http://www.wsmo.org/2004/d2/v0.2/.
17. L. Li and I. Horrocks. A Software Framework For Matchmaking Based on Semantic Web Technology. In *Proceedings of the 12th World Wide Web Conference*, 2003.
18. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mogiello. A System for Principled Matchmaking in an Electronic Marketplace. *Journal of E-Commerce vol.9*, 2004.
19. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, pages 333–347, 2002.
20. P. F. Patel-Schneider, P. Hayes, I. Horrocks, and F. van Harmelen. OWL Web Ontology Language; Semantics and Abstract Syntax, W3C Candidate Recommendation. http://www.w3.org/TR/owl-semantics/, November 2002.
21. C. Preist. A Conceptual Architecture for Semantic Web Services. In *Proceedings of the 3rd International Semantic Web Conference (ISWC)*, 2004.
22. C. Preist, J. Esplugas-Cuadrado, S. Battle, S. Grimm, and S. Williams. Automated B2B Integration of a Logistics Supply Chain Using Semantic Web Services Technology. In *Proc. of the 4th Int. Semantic Web Conference (ISWC)*, 2005.
23. R. Rosati. Autoepistemic Description Logics. *AI Communications, IOS Press*, 11(3–4):219–221, 1998.
24. E. Di Scasio, F. M. Dononi, M. Mongiello, and G. Piscitelli. A Knowledge Based System for Person-to-Person E-Commerce. In *Proc. of the KI-2001 Workshop on Applications of Description Logics*, 2001.
25. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A Practical OWL-DL Reasoner. Technical report, University of Maryland Institute for Advanced Computer Studies (UMIACS), 2005. http://mindswap.org/papers/PelletDemo.pdf.
26. D. Trastour, C. Bartolini, and C. Preist. Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In *Proceedings of the Eleventh International Conference on World Wide Web*, pages 89–98, 2002.