Deciding Semantic Matching of Stateless Services*

Duncan Hull†, Evgeny Zolin†, Andrey Bovykin‡, Ian Horrocks†, Ulrike Sattler†, and Robert Stevens†

† School of Computer Science,
University of Manchester, UK

‡ Department of Computer Science,
University of Liverpool, UK

Abstract

We present a novel approach to describe and reason about stateless information processing services. It can be seen as an extension of standard descriptions which makes explicit the relationship between inputs and outputs and takes into account OWL ontologies to fix the meaning of the terms used in a service description. This allows us to define a notion of matching between services which yields high precision and recall for service location. We explain why matching is decidable, and provide biomedical example services to illustrate the utility of our approach.

Introduction

Understanding the data generated from genome sequencing projects like the Human Genome Project is recognised as a "grand challenge" both for Computer Science and Biomedicine (Sleep 2004; Collins *et al.* 2003). Many of the tools and databases for analysing this data are available via Web Service interfaces, thereby allowing biomedical scientists to use the Web as a platform to perform so-called *in silico* experiments. Large numbers of these experiments are carried out by choosing some of these Web Services, composing them into a workflow, and running them (Stevens *et al.* 2004; Hull *et al.* 2006)—an approach which shows considerable promise for molecular biology (Stein 2002) whilst challenging current Web Service approaches. In contrast to other application areas, the structure of these workflows is mostly determined by the biologist designing the experiment, who also has a clear picture in mind of the kind of services he or she wants to use in each experimental step. Moreover, a variety of domain ontologies exist which capture the knowledge of biologists, see, e.g., `http://obo.sourceforge.net/`. Finally, the majority of these services are *stateless*, i.e., they provide information, but do not change the state of the world—apart from the knowledge of the biologist running the service, which we will ignore here. We restrict our attention to these kinds of services since they are quite common yet easier to represent, and since it turned out that defining a semantics or specifying automated reasoning algorithms for such stateful service descriptions is basically impossible in the presence of any expressive ontology (Baader *et al.* 2005). Statelessness implies that we do not need to formulate pre- and post-conditions since our services do not change the world.

The question we are interested in here is how to help the biologist to find a service he or she is looking for, i.e., a service that works with inputs and outputs the biologist can provide/accept, and that provides the required functionality. The growing number of publicly available biomedical web services, 3 000 as of February 2006, required better matching techniques to locate services. Thus, we are concerned with the question of how to describe a service request Q and service advertisements S_i such that the notion of a service S *matching the request* Q can be defined in a "useful" way. By useful, we mean the following: (1, precision) only those services should match the request that indeed provide the requested functionality; (2, recall) all services providing the requested functionality should match the request; (3) service advertisements and requests should be formulated using terms from existing (OWL) ontologies; and (4) such that the matching problem can be decided automatically.

Let us illustrate the first three points using a simple example from (Martin *et al.* 2004); realistic examples from molecular biology are discussed later. Consider a service provider who advertises a service S_1 with an input of type GeoRegion, an output of type Wine, and which returns a list of wines produced in the region with which it was called. Moreover, consider a service S_2 which has the same input and output, but S_2 returns a list of wines that are sold in the region with which it was called. Now, if the types of a service's input and output are the only information available to match a request to a service, then no matching algorithm can distinguish between S_1 and S_2: they have identical types, and thus matching cannot be precise—see (1) above. That is, S_1 will be matched to a request whenever S_2 is, regardless of whether the request Q is for a service that returns wines produced or sold in this region, i.e., regardless of the required functionality of the service. Next, assume that a user requests a service that takes, as input, a FrenchGeoRegion and returns a list of FrenchWines that are produced in this region. Even though our service S_1 returns, in general, wines that may not be FrenchWines, it returns only FrenchWines when called with a FrenchGeoRe-

*This work is supported by EPSRC grants GR/R67743/01 and GR/S63168/01

gion, and thus should be matched to this request—a matching algorithm that does this can clearly be viewed as having high recall, see (2) above. To determine this match, however, we need to take into account some background knowledge, namely that only FrenchWines are produced in a FrenchGeoRegion. To see this, consider the case where the request is for a service that takes a FrenchGeoRegion and returns a list of FrenchWines that are *sold* in this region: in this case, S_2 should not be matched since some shops in this region might sell wines that are not french.[1] In general, if a service description uses terms whose meaning is defined in an ontology, this not only enables the reuse of these definitions and thus makes service descriptions more succinct, but it also allows their semantics to be taken into account for matching.

Another way of describing a service's functionality would be by using terms from a fixed vocabulary of functionalities—such as "sold-wines"and "grown-wines". This does not integrate well with a background ontology: for example, it would not allow for the above mentioned matching of S_1 to the request for a service that takes a FrenchGeoRegion and returns a list of FrenchWines that are produced in this region.

In this paper, we will propose *a framework to describe stateless services that takes into account background ontologies and where matching of services is defined such that it (a) yields matchings with high precision and recall and (b) the matching problem is decidable*. Moreover, our framework allows to compute automatically, from the description of atomic services, a description of their composition.

Services as queries

In this section, we present our framework for describing and matching *stateless* web services. From a syntactic point of view, it can be viewed as an extension of the way services are described in the OWL-S Service Profile (namely, of its part concerning description of inputs and outputs). From a semantic viewpoint, the definition of "service matching" introduced below will allow for service matching with high-precision and recall.

As mentioned in the introduction, in addition to the types of inputs and outputs, our service descriptions explicate the *relationship* between inputs and outputs. Analysing numerous examples of services—including those in bioinformatics, see Section)—it was observed that the notion of *conjunctive query* can be adopted for these purposes. Before defining this "services as queries" approach, we illustrate it using our wine service examples. Intuitively, when run with a GeoRegion g, the produced-wine service S_1 returns all those wines w *for which there exists some winegrower* f *who produces* w *and who is located in* g. In our framework, this service can thus be described as follows:

```
INPUT: g GeoRegion
OUTPUT: w Wine
THERE IS SOME f  [WineGrower(f),
      LocatedIn(f,g), Produces(f,w)],
```

where the terms `Wine`, `LocatedIn`, etc., are defined in some ontology. In contrast, the sold-wine service S_2 returns all those wines w *for which there exists some shop* s *who sells* w *and who is located in* g, and can thus be described as follows:

```
INPUT: g GeoRegion
OUTPUT: w Wine
THERE IS SOME s
   [Shop(s), LocatedIn(s,g), Sells(s,w)]
```

Given service descriptions of this kind, matching of services can be reduced to query containment w.r.t. an ontology—a task whose decidability and complexity is relatively well understood; see, e.g., (Calvanese *et al.* 2005; Calvanese, De Giacomo, & Lenzerini 1998; Horrocks *et al.* 2000).

Describing services

We assume the reader to be familiar with OWL-DL and its semantics (Horrocks, Patel-Schneider, & van Harmelen 2003). Throughout this paper, we borrow the term *TBox* for a class-level ontology (i.e., a finite set of OWL-DL axioms) and *ABox* for a factual ontology (i.e., a finite set of OWL-DL facts). The union of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ is called a *knowledge base* and denoted by $\mathcal{KB} = \langle \mathcal{T}, \mathcal{A} \rangle$. We use $\mathcal{KB} \models \Psi$ to denote the fact that (the description logic or first order translations of) $\mathcal{KB}$ and imply Ψ, i.e., Ψ holds in every interpretation that satisfies $\mathcal{KB}$.

Definition 1 (Service syntax)**.** Let $\mathcal{X}$ be a set of variables. A *service description* $\langle\, \vec{x} : \vec{X};\ \vec{y} : \vec{Y};\ \Phi(\vec{x}, \vec{y})\,\rangle$ consists of

- a list $\vec{x} : \vec{X} = \langle x_1 : X_1, \ldots, x_m : X_m \rangle$ of pairs of variables x_i from $\mathcal{X}$ and classes X_i; this list enumerates input variables and their "types";
- a list $\vec{y} : \vec{Y} = \langle y_1 : Y_1, \ldots, y_n : Y_n \rangle$ of pairs of variables y_j from $\mathcal{X}$ and classes Y_j; this list enumerates output variables and their "types";
- a *relationship specification* $\Phi(\vec{x}, \vec{y}, \vec{z})$ of the form

  $$\mathit{term}_1(\vec{x}, \vec{y}, \vec{z}) \wedge \ldots \wedge \mathit{term}_k(\vec{x}, \vec{y}, \vec{z}),$$

  where each $\mathit{term}_i(\vec{x}, \vec{y}, \vec{z})$ is either an expression of the form $C(w)$ with C a class or $R(w_1, w_2)$ with R a property, and w, w_1, and w_2 variables from $\mathcal{X}$ that occur in $\vec{x}$, $\vec{y}$, or $\vec{z}$, or individual names.

In OWL-DL terms, $\Phi(\vec{x}, \vec{y}, \vec{z})$ is a set of facts of the form $\texttt{Individual}(w\ \ \texttt{type}(W))$ and $\texttt{Individual}(w\ \ \texttt{value}(R\ \ w_1))$ over variables, some of which may occur in the input or in the output. Using the syntax from Definition 1, our wine services can be formalised as follows:

$$S_1 = \langle\, g : \texttt{GeoRegion};\ w : \texttt{Wine};\ (\texttt{WineGrower}(f) \wedge \texttt{LocatedIn}(f, g) \wedge \texttt{Produces}(f, w))\,\rangle$$

$$S_2 = \langle\, g : \texttt{GeoRegion}; w : \texttt{Wine}; (\texttt{Shop}(s) \wedge \texttt{LocatedIn}(s, g) \wedge \texttt{Sells}(s, w))\,\rangle$$

We prefer the syntax given in Definition 1 to the one used informally above since it is shorter; it is clear, however, how to translate between these two representations, and we will not go into any further discussion of syntax here. To enhance readability, for a variable vector $\vec{z} = z_1, \ldots, z_\ell$, we use $\exists \vec{z}$ as an abbreviation for $\exists z_1, \ldots, z_\ell$. Next, we define what it means for a service to implement a service description.

[1] Obviously, S_1 should not be matched since it returns the wrong wines.

Definition 2 (Service semantics)**.** Let $\mathcal{T}$ be a TBox and S a service description as in Definition 1. A service s *implements* a service description S over $\mathcal{T}$ if, for any ABox $\mathcal{A}$ and any individuals $a_1, \ldots, a_m$ in $\mathcal{T}$ and $\mathcal{A}$, if $\mathcal{T}, \mathcal{A} \models X_i(a_i)$ for each $1 \leq i \leq m$, then

1. s accepts $\vec{a} = \langle a_1, \ldots, a_m \rangle$ as input and
2. when run with $\vec{a}$ as input, it returns the set of all those tuples of individuals $\vec{b} = b_1, \ldots, b_n$ from $\mathcal{A}$ such that

$$\mathcal{T}, \mathcal{A} \models \vec{Y}(\vec{b}) \wedge \exists \vec{z} : \Phi(\vec{a}, \vec{b}, \vec{z}).$$

Intuitively, an input $\vec{a}$ must be an instance of $\vec{X}$ w.r.t. the background ontology, and the service returns as its output the set of all tuples of objects $\vec{b}$ that are instances of $\vec{Y}$ w.r.t. the ontology and for which we can find some $\vec{z}$ such that $\vec{a}$, $\vec{b}$, and $\vec{z}$ satisfy the condition $\Phi()$.

Next, we develop the means of comparing service descriptions, i.e., a notion of one service matching another.

Matching services

Matching is the problem of determining whether a given service description S conforms to another service description Q. Matching algorithms can be used for the location of services, and we can think of S as being a service advertisement and of Q as being a service requested by a user. As a consequence, each definition of matching of services should express some reasonable conditions for a service S to be considered as an "appropriate" candidate to be returned by a search engine to a user who specified a request Q.

As mentioned above, we assume that services are described w.r.t. a terminological ontology (TBox) $\mathcal{T}$. We will first give a formal definition, and then provide explanations. We use $|\vec{x}|$ to denote the length of a vector $\vec{x}$.

Definition 3. Given two service descriptions:

$$\begin{array}{l} S = \langle\, \vec{x} : \vec{X};\ \ \vec{y} : \vec{Y};\ \ \Phi(\vec{x}, \vec{y}, \vec{u}) \,\rangle, \\ Q = \langle\, \vec{z} : \vec{Z};\ \ \vec{w} : \vec{W};\ \ \Psi(\vec{z}, \vec{w}, \vec{v}) \,\rangle, \end{array} \tag{1}$$

with $|\vec{x}| = m = |\vec{z}|$ and $|\vec{y}| = n = |\vec{w}|$, we say that the service S *matches* the request Q w.r.t. the TBox $\mathcal{T}$ if there exist two permutations

$$\begin{array}{l} \pi : \{1, \ldots, m\} \to \{1, \ldots, m\} \\ \rho : \{1, \ldots, n\} \ \to \{1, \ldots, n\} \end{array}$$

such that the following two conditions hold:

(i) $\mathcal{T} \models Z_{\pi(i)} \sqsubseteq X_i$, for all $1 \leqslant i \leqslant m$, i.e., for each input x_i in the advertised service S, we can find a matching input $z_{\pi(i)}$ in the requested service Q such that $Z_{\pi(i)}$ is a (possibly implicit) sub-class of X_i w.r.t. the ontology $\mathcal{T}$. Intuitively, this means that we can map the inputs from S to the inputs from Q such that all input data that the user intends to provide will be accepted by S.

(ii) for any ABox $\mathcal{A}$ and any individuals $\vec{a} = \langle a_1, \ldots, a_m \rangle$, $\vec{b} = \langle b_1, \ldots, b_n \rangle$ in the knowledge base $\mathcal{KB} = \langle \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{KB} \models \vec{Z}(\vec{a})$, then the equivalence holds:

$$\begin{array}{ll} \mathcal{KB} \models \vec{Y}(\rho(\vec{b})) \ \wedge\ \exists \vec{u}\ \Phi(\pi(\vec{a}), \rho(\vec{b}), \vec{u}) & \text{iff} \\ \mathcal{KB} \models \vec{W}(\vec{b}) \ \wedge\ \exists \vec{v}\ \Psi(\vec{a}, \vec{b}, \vec{v}), & \end{array}$$

where $\pi(\vec{a})$ and $\rho(\vec{b})$ are the permutations of $\vec{a}$ and $\vec{b}$ according to π and ρ.

Intuitively, this means that, modulo some re-arrangement of the input and output vectors, the services S and Q return the same answers on any input that conforms to the request Q.

Some remarks are in order here. The need to permute inputs and outputs of Q to "fit" the ones of S is by no means new—it is present in any reasonable definition of service matching. Thus, in order to check whether S matches Q, a reasoning system must "guess" two appropriate permutations or exhaustively explore all possible assignments. Condition **(i)** is quite standard; for example, it can be found in definitions for matching of OWL-S services (Payne, Paolucci, & Sycara 2001). In contrast, condition **(ii)** is—to the best of our knowledge—new, and it is not expressible in terms of OWL-S service profiles. As we will see in Section , this condition is in fact reducible to checking containment between two conjunctive queries w.r.t. a TBox, which is a standard reasoning task.

The above definition covers only the case when $|\vec{x}| = |\vec{z}|$ and $|\vec{y}| = |\vec{w}|$. We can easily extend this approach to assume that Q contains "redundant" input variables and S "redundant" output variables. In contrast, if S contains more input variables than Q, then one possible solution is to try to "instantiate" some of the inputs of S with individual names in order to decrease the number of inputs of S. Another option is to merge some inputs of S, provided that their types are compatible. Similarly, if Q has more output variables than S, it might be reasonable that a search engine still tries to match S and Q by trying to check whether S produces at least part of the requested outputs. All these cases are discussed in more detail and illustrated by examples in the accompanying technical report (Bovykin and Zolin 2005).

Service composition

In this section, we will show that, from the service descriptions $S_1, \ldots, S_n$, we can automatically construct a description of the sequence of services $S_1 \circ \ldots \circ S_n$. This is another important advantage of our approach since it decreases the annotation workload for the web service provider. For the beginning, suppose that we have two service descriptions in a repository, both having only one input and one output:

$$\begin{array}{l} S\ = \langle\, x : X;\quad y : Y;\ \ \Phi(x, y, \vec{u}) \,\rangle, \\ S' = \langle\, x' : X';\ y' : Y';\ \Phi'(x', y', \vec{u}\,') \,\rangle. \end{array}$$

Our task is to formulate reasonable conditions when the composition of services $S \circ S'$ (to be read as "first S runs, then S' runs on the output produced by S") is meaningful (i.e., when these services are compatible) and when it matches a user's request

$$Q = \langle\, z : Z;\ w : W;\ \Psi(z, w, \vec{v}) \,\rangle.$$

Definition 4. A composition of services $S \circ S'$ *matches* a request Q w.r.t. a TBox $\mathcal{T}$ if the following conditions hold:

(a) $\mathcal{T} \models Z \sqsubseteq X$. As usual, this ensures that S accepts all inputs described in the request Q.

(b) for any ABox $\mathcal{A}$ and any individuals a, b in the knowledge base $\mathcal{KB} = \langle \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{KB} \models Z(a) \wedge Y(b) \wedge \exists \vec{u} : \Phi(a, b, \vec{u})$, then $\mathcal{KB} \models X'(b)$.
This ensures that, if S runs on inputs provided by the user, then its outputs are accepted by S'.

(c) for any ABox $\mathcal{A}$ and any individuals a, c in the knowledge base $\mathcal{KB} = \langle \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{KB} \models Z(a)$, then $\mathcal{KB} \models W(c) \wedge \exists \vec{v} : \Psi(a, c, \vec{v})$ holds iff there exists an individual b in $\mathcal{KB}$ such that

$$\mathcal{KB} \models \exists \vec{u}\ \Phi(a, b, \vec{u}) \wedge Y(b) \wedge \exists \vec{u}'\ \Phi'(b, c, \vec{u}') \wedge Y'(c).$$

This means that, on the user's inputs, the application of S and then S' yields the same answers as Q.

Observe that the notion of service composition is *request dependent* in the sense that the composition of services $S \circ S'$ is built for a particular request Q. This is so because, in general, the services S and S' may not be compatible—S may return outputs that the S' cannot accept—yet on inputs that a user is intended to provide, S returns only outputs that are accepted by S'.

More generally, if we are given several services $S_1, \ldots, S_r$, all having only one input and one output, then we can easily modify Definition 4 accordingly. Condition **(b)** will say that the outputs of S_1 (on user's inputs) are accepted by S_2, that the outputs of S_2 (on inputs coming from S_1) are accepted by S_3, etc. Condition **(c)** is easier to modify: its first line is unchanged, whereas in its first line we state that $\mathcal{KB}$ entails the assertion that c has the type of the output of the last service S_r the conjunction of all formulas Φ_i with identification of their intermediate variables.

Finally, this notion of composition of services can be further generalised to the case of several services having multiple inputs and outputs. No fundamental difficulties arise here, but the notation becomes cumbersome due to permutations of variables.

Example Queries and Matches

Before describing how to decide matching between services, we discuss some examples from the biomedical domain to illustrate the usefulness of our approach. In general, we observe that most of these services take (one or more) strings as input and return (one or more) strings as output. Hence, any attempt to match services based on the types of input and output is doomed to be of rather low precision. Moreover, these services provide access to tools, databases, and algorithms developed in the public domain, and a large part of these services are related to the analysis of the biological molecule DeoxyriboNucleic Acid (DNA). There are at least 20 different syntaxes[2] for representing just four bases (A, C, T, G) of the DNA code: GenBank_Format and EMBL_ Format are two commonly used formats for representing DNA sequences. These DNA sequences and associated metadata are stored in a large number of public repositories, 858 databases (Galperin 2006) at the last count.

Consider the following advertisements for two "shim" services (Hull *et al.* 2004) which extract the DNA sequence from a GenBankRecord.

[2] http://emboss.sf.net/docs/themes/SequenceFormats.html

```
S1: INPUT: x GenBankRecord
    OUTPUT: y DNASeqRepresentation
    [ hasPart(x,y) ]

S2: INPUT: x GenBankRecord
    OUTPUT: y DNASeqRepresentation
    THERE IS SOME d,e [DNASequence(d),
      EMBLRecord(e), about(x,d), about(e,d),
      hasPart(e,y)]
```

They coincide on their inputs and outputs, yet they will behave in slightly different ways. The first service simply extracts the DNASequence from the input, whereas the second one first extracts the DNA sequence and then translates the syntax from GenBankForm to EMBLForm. Since there at least 20 different formats for representing DNA sequences, we have to distinguish between a DNA sequence and its representation in one of these formats.

Now consider the following request, which describes services taking a GenBankRecord and returning the corresponding DNA sequence in EMBL format:

```
Q: INPUT: x GenBankRecord
   OUTPUT: y DNASeqRepresentation
   THERE IS SOME d,e
   [ DNASequence(d), Record(e), about(x,d),
    about(e,d), hasPart(e,y), EMBLform(y)]
```

First, note that, according to our definition of matching, the service S1 does not match our request Q since it cannot guarantee that the output is in EMBL format. In contrast, if our TBox contains

$$\texttt{SubClassOf}(\texttt{EMBLRecord}\ \texttt{restriction}(\texttt{hasPart}\ \texttt{allValuesFrom}\ \texttt{EMBLform}))$$

which ensures that all entries in an EMBLRecord are in EMBLform, then service S2 matches our request—which is indeed useful. Similarly, in the presence of the above OWL axiom, S2 even matches the following request—despite the fact that the output of the request is declared to be more specific than that provided by the second service.

```
Q1: INPUT: x GenBankRecord
    OUTPUT: y IntersectionOf(
      DNASeqRepresentation EMBLform)
    THERE IS SOME d,e
    [ DNASequence(d), Record(e), about(x,d),
     about(e,d), hasPart(e,y)]
```

Let us point out again that our definition of matching yields both a higher precision and a higher recall than any comparison of inputs and outputs could possibly yield: it matches services whose inputs or outputs do not match in an obvious way (such as Q2 and S2 above), and it does not match services despite their in- and outputs matching (such as S6 and Q2). The latter point is especially important for biomedical Web Services since many take strings as in- and outputs—and thus all services would match on the grounds of in- and outputs.

Deciding service matching

In this section, we show that the problem of deciding whether a service S matches a service Q is reducible to a standard reasoning problem, namely to *containment* of conjunctive queries w.r.t. a TBox.

Definition 5. A *conjunctive query* is an expression of the form $q(\vec{x}) \leftarrow term_1(\vec{x},\vec{y}) \wedge \ldots \wedge term_k(\vec{x},\vec{y})$, where $\vec{x}$ and $\vec{y}$ are lists of variables and each $term_i$ is of the form $C(w)$ or $R(w,z)$, where C is a class, R a property, and w, z are either variables from the lists $\vec{x}, \vec{y}$ or individual names. We call $\vec{x}$ *distinguished* and $\vec{y}$ *non-distinguished* variables. As in our service descriptions, the existential quantification of non-distinguished variables ($\exists\vec{y}$) is only implicit in a query.

Given a knowledge base $\mathcal{KB}$, the *answer* to a query $q(x_1,\ldots,x_m)$ over $\mathcal{KB}$ is defined as follows:

$$q(\mathcal{KB}) = \{\langle a_1,\ldots,a_m\rangle \mid \text{all } a_i \text{ are individuals in } \mathcal{KB} \text{ and } \mathcal{KB} \models \exists\vec{y}\ \big(term_1(\vec{a},\vec{y}) \wedge \ldots \wedge term_k(\vec{a},\vec{y})\big)\}.$$

A query $q_1(\vec{x})$ *subsumes* a query $q_2(\vec{x})$ w.r.t. a TBox $\mathcal{T}$ if, for each ABox $\mathcal{A}$, $q_1(\mathcal{T},\mathcal{A}) \supseteq q_2(\mathcal{T},\mathcal{A})$. Two queries $q_1(\vec{x})$ and $q_2(\vec{x})$ are *equivalent* w.r.t. $\mathcal{T}$ if they subsume each other.

Theorem 1 (Reduction)**.** *Service matching w.r.t. a TBox is reducible to conjunctive query equivalence w.r.t. a TBox.*

The proof can be found in (Bovykin and Zolin 2005). Now consider Definition 4 of service composition $S \circ S'$ matching a service request Q. Again, Condition **(a)** is just a concept subsumption; Condition **(b)** holds iff the query

$$q(x,y) \leftarrow Z(x) \wedge Y(y) \wedge \Phi(x,y,\vec{u})$$

is subsumed by the query $q'(x,y) \leftarrow X'(y)$. That Condition **(c)** is reducible to query subsumption is not straightforward, because of the quantification over *individuals* in a knowledge base, not over arbitrary elements of models of KB. However, it can be shown that this quantification can be "internalized" and hence the Condition **(c)** is equivalent to the following one:

(c') for any ABox $\mathcal{A}$ and any individuals a, c in the knowledge base $\mathcal{KB} = \langle\mathcal{T},\mathcal{A}\rangle$, if $\mathcal{KB} \models Z(a)$, then the condition

$$\mathcal{KB} \models W(c) \wedge \exists\vec{v}\colon \Psi(a,c,\vec{v})$$

is equivalent to the condition

$$\mathcal{KB} \models \exists t,\vec{u},\vec{u}'\ \big(\Phi(a,t,\vec{u}) \wedge Y(t) \wedge \Phi'(t,c,\vec{u}') \wedge Y'(c)\big).$$

For detailed proof of the equivalence of **(c)** and **(c')** see (Bovykin and Zolin 2005). Finally, it remains to notice that Condition **(c')** states the equivalence of two conjunctive queries, so we are done.

Consequently, Definition 4 turns out to be modular in the sense that a service description for a composite service can be constructed automatically from the description of its components (provided that we have checked their compatibility), and then matching is defined as usual. To be more precise, from the above it follows that, given two services S and S' as in Definition 4 that are compatible *on inputs of* Q, the description for the composition $S \circ S'$ is the following:

$$S \circ S' = \langle x\colon X;\ y'\colon Y';\ \Phi(x,t,\vec{u}) \wedge Y(t) \wedge \Phi'(t,y',\vec{u}')\rangle,$$

provided that the lists $\vec{u}$ and $\vec{u}'$ are disjoint—and we can always rename these variables.

Next, we will briefly discuss results on the decidability and complexity of query containment w.r.t. ontologies. In general, query containment is at least as hard as concept subsumption or satisfiability. Hence, the lower bound for its complexity immediately follows from the lower complexity bound of a Description Logic itself.

As for upper bounds, there are several results in this direction. In (Calvanese, De Giacomo, & Lenzerini 1998), the query containment problem for $\mathcal{DLR}_{reg}$ is shown to be exponential in size of TBox and double exponential in size of queries. This is a logic with n-ary relations and boolean operations on them, and regular expressions for binary relations. In (Horrocks *et al.* 2000), the query containment problem for the logic $\mathcal{DLR}$ was reduced to checking ABox satisfiability for the same logic, which, in turn, was reduced to knowledge base satisfiability for the Description Logic $\mathcal{SHIQ}$, which is decided by numerous DL reasoners. In both settings, however, only so-called simple properties are allowed in query terms. Hence it does not completely suite our setting: most of our example services involve a transitive and thus non-simple property `hasPart`. In (Ortiz de la Fuente *et al.* 2005), the query containment for the logic $\mathcal{SHIQ}$ is shown to be 3coNExpTime, provided that the KB has no transitive roles. Currently, the complexity is investigated and practical algorithms are devised for query containment over the logic $\mathcal{SHOIQ}$, the DL underlying OWL-DL.

Summing up, service matching w.r.t. OWL ontologies is known to be decidable, and decision procedures for this problem are available. Yet, to the best of our knowledge, neither tight complexity bounds nor an implementation are currently available.

Related work

WSDL descriptions The Web Services Description Language (WSDL) 1.1 is used to describe around 80% of the 3 000 services currently available in bioinformatics. In bioinformatics, "WSDL in the wild" is typically underdescriptive. For example, take `xembl`,[3] which takes an input string, performs an operation `getNucSeq` on that input string, and returns a string. To the WSDL-literate domain expert, it might be clear that this service returns an XML formatted representation of a DNA sequence identifier it received as an input, yet this knowledge has to be deduced solely from the terms used in the description. As mentioned before, having inputs and outputs of type string is typical of the biomedical domain, although these strings hide complex flat-file and legacy formats. Directly annotating the WSDL file itself as proposed in WSDL-S [4] is usually not possible because the WSDL is provided as-is by a third party. Consequently, any richer semantic descriptions are best stored independently of WSDL.

RDF annotations of WSDL In order to enable semantic discovery of services, two closely related projects, BioMOBY and *my*Grid Feta (Lord *et al.* 2004; 2005) have taken existing WSDL descriptions and annotated them using RDF. These projects have created a centralised registry of many services annotated with RDF. Because of RDF's restricted expressivity, neither of these approaches, however, would allow the

[3] `http://www.ebi.ac.uk/xembl/XEMBL.wsdl`.

[4] `http://www.w3.org/Submission/WSDL-S/`

line of reasoning we used to determine that the produced-wine service matches the query for french wines: the annotation of tasks, functions, etc., does not take into account relationships between in- or outputs.

OWL-S For the problem of service matching and location, an OWL-S description includes a service *profile*. In a profile, we can specify, besides others, inputs, outputs, and their types, as well as pre- and postconditions. Now OWL-S is written in OWL, and thus the syntactic restrictions of OWL imply that the relationships between inputs and outputs cannot be described. A service profile includes pre- and postconditions whose semantics remains unclear since OWL is stateless, i.e., OWL does not provide mechanisms to distinguish different states such as "before" the execution of aservice or "after" the execution of a service.

WSMO WSMO (Roman *et al.* 2005) has a class *capability* to describe a service's functionality, and allows the relation of inputs to outputs using shared variables in a similar way as the approach described here. To the best of our knowledge, however, no decidability results are known for matching services described in WSMO.

Conclusions

We have discussed various ways of describing stateless services which are quite common, e.g., in the bioinformatics domain. We propose a framework to describe the functionality of and reason about such services. As all other web service description formalisms, we represent information about the inputs and outputs of a service. To describe a service's functionality, we allow description of how inputs and outputs are related, and we use an ontology to fix the meaning of terms used in service descriptions. Most importantly, we provide a definition of a service *matching* a request which (i) takes all this information into account and (ii) is decidable.

From a logical perspective, it is clear that our framework can be easily combined with OWL-S and WSMO, thereby allowing for automated reasoning approach for matching services with high precision and recall.

References

A.Bovykin and E.Zolin. 2005. A formal framework for describing information providing web services. Technical report, University of Manchester. Available at `http://dynamo.man.ac.uk/publ/aaai06tr.pdf`.

Baader, F.; Lutz, C.; Miličić, M.; Sattler, U.; and Wolter, F. 2005. Integrating description logics and action formalisms: First results. In Press, A. P. M., ed., *Proc. of AAAI-05*.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2005. Data complexity of query answering in description logics. In *Proc. of DL'05)*, volume 147.

Calvanese, D.; De Giacomo, G.; and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *Proc. of PODS'98*, 149–158.

Collins, F. S.; Green, E. D.; Guttmacher, A. E.; and Guyer, M. S. 2003. A vision for the future of genomics research: A blueprint for the genomic era. *Nature* 422(6934):835–847. US National Human Genome Research Institute.

Galperin, M. 2006. The molecular biology database collection: 2006 update. *Nucleic Acids Research* 34(Database issue):3–5.

Horrocks, I.; Sattler, U.; Tessaris, S.; and Tobies, S. 2000. How to decide query containment under constraints using a description logic. In *Proc. of LPAR'00*, LNAI. Springer-Verlag.

Horrocks, I.; Patel-Schneider, P. F.; and van Harmelen, F. 2003. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics* 1(1).

Hull, D.; Stevens, R.; Lord, P.; Wroe, C.; and Goble, C. 2004. Treating shimantic web syndrome with ontologies. In *Proc. of AKT-SWS04*.

Hull, D.; Wolstencroft, K.; Stevens, R.; Goble, C.; Pocock, M.R.; Li, P.; and Oinn, T. 2006. Taverna: A tool for building and running workflows of services. In *Nucleic Acids Research*. 34 (Web Server Issue)

Lord, P.; Bechhofer, S.; Wilkinson, M. D.; Schiltz, G.; Gessler, D.; Hull, D.; Goble, C.; and Stein., L. 2004. Applying Semantic Web Services to bioinformatics: Experiences gained, lessons learnt. In *Proc. of ISWC'04*. Springer-Verlag.

Lord, P.; Alper, P.; Wroe, C.; and Goble, C. 2005. Feta: A light-weight architecture for user oriented semantic service discovery. In *Proc. of ESWC'05*. Springer-Verlag.

Martin, D.; Paolucci, M.; McIlraith, S.; Burstein, M.; McDermott, D.; McGuinness, D.; Parsia, B.; Payne, T. R.; Sabou, M.; Solanki, M.; Srinvasan, N.; and Sycara, K. 2004. Bringing Semantics to Web Services: the OWL-S approach. In *SWSWPC 2004*. Springer-Verlag.

Ortiz de la Fuente, M. M.; Calvanese, D.; Eiter, T.; and Franconi, E. 2005. Data Complexity of Answering Conjunctive Queries over SHIQ Knowledge Bases. Technical report, Faculty of Computer Science, Free University of Bozen-Bolzano.

Payne, T. R.; Paolucci, M.; and Sycara, K. 2001. Advertising and Matching DAML-S Service Descriptions. *Semantic Web Working Symposium (SWWS)*.

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

Sleep, R. 2004. *Grand Challenges in Computing Research*. British Computer Society. chapter GC1: In Vivo-In Silico (iVis): The virtual worm, weed, bug. isbn:190250562X.

Stein, L. 2002. Creating a bioinformatics nation. *Nature* 417:119–120.

Stevens, R. D.; Tipney, H. J.; Wroe, C.; Oinn, T.; Senger, M.; Lord, P. W.; Goble, C. A.; Brass, A.; and Tassabehji, M. 2004. Exploring Williams-Beuren Syndrome Using myGrid. *Bioinformatics* 20.

Tessaris, S. 2001. *Questions and answers: reasoning and querying in Description Logic*. Ph.D. Dissertation, University of Manchester.