Semantic Annotations and Querying of Web Data Sources

Thomas Hornung[1] and Wolfgang May[2]

[1] Institut für Informatik, Universität Freiburg
hornungt@informatik.uni-freiburg.de
[2] Institut für Informatik, Universität Göttingen
may@informatik.uni-goettingen.de

Abstract. A large part of the Web, actually holding a significant portion of the useful information throughout the Web, consists of views on hidden databases, provided by numerous heterogeneous interfaces that are partly human-oriented via Web forms ("Deep Web"), and partly based on Web Services (only machine accessible). In this paper we present an approach for annotating these sources in a way that makes them citizens of the Semantic Web. We illustrate how queries can be stated in terms of the ontology, and how the annotations are used to selected and access appropriate sources and to answer the queries.

1 Introduction

The Web is today a major source for retrieving *data*. The term *Web Data Sources* covers Web sources that allow access to an underlying database-style repository, and that exhibit at least implicitly a table-like schema. Fortunately, this is the case for virtually all *structured, Web-accessible* data sources, ranging from *human-readable Deep Web* [10] sources – databases that are accessible via filling out Web forms – to data-centric *Web Services* that are machine-accessible via low-level technologies such as REST [9] or SOAP (Web Services). These sources could be very useful for declarative query answering and data workflows, but the task of combining information from different such sources is cumbersome, and is currently an actively researched area [4].

In this paper we propose an approach to describe Web Data Sources in a unified way by assigning semantic annotations to them. The annotations cover the technical handling of (wrapped) sources, the description of their input and output characteristics in terms of *tags*, and the relationship between these tags in terms of the underlying domain ontology. We describe how these annotations are exploited to evaluate SPARQL [28] queries that are stated in terms of the domain ontology.

Structure of the Paper. Section 2 gives a short overview of RDF as the data model underlying the Semantic Web and of SPARQL as the most common RDF query language. In Section 3 we introduce the application scenario, which deals with querying for flight connections. Section 4 develops the concepts used for

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 112–129, 2009.

semantic annotation of Web Data Sources. These concepts are formalized in an RDF ontology in Section 5. Section 6 shows how the annotations are used for evaluating queries w.r.t. the annotated sources. Section 7 discusses related work and we conclude with Section 8.

2 Preliminaries: RDF and SPARQL

The *Resource Description Framework (RDF)* [25] is the data model underlying the Semantic Web. In RDF, information is represented by a labeled directed graph where nodes represent *resources*, identified by URIs, and literals, and the labeled edges represent the relationships between them. The edge labels are also URIs, which makes up the salient feature of the Semantic Web: properties and classes can also have properties, which allows for expressing metadata information. In this paper, we represent RDF data in its Turtle [29] notation. Shortly,

res_1 a cl; prop_1 res_2; prop_2 res_3,res_4 . res_4 prop_3 [prop_4 "foo"].

sketches the syntax varieties of Turtle: the resource denoted by res_1 is of class cl ("a" is a shorthand for rdf:type), has properties prop_1, resulting in res_2, and it is related via prop_2 to resources res_3 and res_4, where res_4 in turn is related by property prop_3 to an unnamed resource that has a property prop_4 with value "foo". Instead of writing URIs completely, the usage of prefixes analogous to namespaces in XML is common. We will omit the (repeated) declarations of the common prefixes for rdf:, rdfs: (RDFS [26]), owl: (OWL [20]), and xsd: (XML Schema [32]).

SPARQL [28] is a query language for RDF data. Its design is based on SQL-like clauses using triple patterns with logical variables (join variables) of the form

select *variables* from *sources/files*
where *dot-separated-extended-triple-patterns-and-filters*

where *dot-separated-extended-triple-patterns-and-filters* are patterns similar to the above Turtle notation. Here, positions can optionally be replaced by variables (of the forms "?X" or "$X"). Additionally, filter conditions (e.g. of the form $?X \leq ?Y$) can be stated. Variables occurring twice or more act as *join variables*. A subset of the variables can be declared in the select clause to be the *answer variables*, whose matches with resources and literals are the answers. The result of a query is thus a set of tuples of variable bindings.

3 Application Scenario

We illustrate the framework with a well-known situation: querying for flight connections. Consider four sample data sources that provide information about flights:

(A) Flight portals like http://www.travenjoy.com/ can be queried with a departure airport, a destination airport (by their IATA codes, e.g., FRA and CDG), and the intended dates (roundtrip) and return (possible composite)

connections between these airports by flight code/flight codes, departure time, arrival time, and price. For connected flights, they usually succeed only if the connection is by a single airline (or alliance).

(B) Websites of a single airline like http://www.lufthansa.de can be queried by entering a departure airport, a destination airport, and the intended traveling dates (roundtrip), returning a list of connections with departure time, arrival time, and price.

(C) Sources like http://www.theairdb.com/ can be queried with a departure airport and return a list of destination airports and the respective airlines that offer flights to these destinations. In combination with sources of type (B), this can be useful to search for possible connections when the destination airport is not yet fixed (e.g., to choose amongst several airports in a certain area), or when no composite connection by a single airline or alliance exists.

(D) A third type of sources (e.g., the schedule of Frankfurt airport at http://www.airportcity-frankfurt.de/) provides three queries: ($D1$) given a destination airport, a date, and optionally desired departure time (abbreviated dDepT below), get airlines and flight numbers that provide direct connections, ($D2$) symmetrically, select an origin airport and ask for connections to Frankfurt. ($D3$) given a timepoint and a date (e.g., 20.5.2009, 9:00) yields all flights (code and destination) that depart during some hours after that time to anywhere. Note that it does not handle availability and prices at all; for this, another data source of type (B) must be used.

The formalization of the notions of the *domain ontology* is later used to correlate the Web Data Sources with the notions of the application domain. The ontology of the required fragment of the traveling ontology consists of the following classes:

- travel:Airport with property travel:code.
- travel:Airline with properties travel:name and travel:website.
- travel:FlightConnection which is the disjoint union of (i) direct flights (travel:Flight) and (ii) connected flights. Connections have the properties travel:from and travel:to, relating them to instances of travel:Airport, travel:deptTime and travel:arrTime indicating their departure time and arrival time, respectively.
- Flights have additionally the properties travel:operatedBy, relating each flight to an airline, and travel:flightCode. Flight here means the abstract notion of e.g. flight number "LH123" operating on weekdays from Frankfurt to Paris at 12:30h.
- Connections that represent connected flights have additionally the (multivalued) property travel:consistsOfFlight, relating them to flights.
- travel:BookableConn: Bookable connections are instances of connections (travel:instanceOfConn), i.e., flights or connected flights, on a given day. They are assigned an actual price for which a ticket can be booked *now*, i.e., the extension of this predicate is dynamic, and is obtained at runtime from a Web Data Source.

Some sample data triples are given in Figure 1.

```
@prefix travel: <http://www.semwebtech.org/domains/2006/travel#> .
@prefix airport: <http://www.semwebtech.org/domains/2006/travel/airports/> .
@prefix airline: <http://www.semwebtech.org/domains/2006/travel/airline/> .
@prefix flight: <http://www.semwebtech.org/domains/2006/travel/flights/> .
airport:fra a travel:Airport; travel:code "FRA" .
airport:cdg a travel:Airport; travel:code "CDG" .
airline:lh a travel:Airline; travel:name "Lufthansa";
    travel:website <http://www.lufthansa.com> .
flight:lh123 a travel:Flight; travel:operatedBy airline:lh; travel:flightCode "LH123";
    travel:from airport:fra; travel:to airport:cdg ;
    travel:deptTime "12:30"; travel:arrTime "13:50".
flight:lh123-20-05-2009 a travel:BookableConn;
    travel:instanceOfConn flight:lh123; travel:date "20-05-2009"; travel:price 300.00.
```

Fig. 1. Sample Data of the Application Scenario

4 Annotation of Web Data Sources

The annotation of Web Data sources concerns three levels. The *technical level* describes how to address the source (or its wrapper, respectively). The *signature level* corresponds to the *signature* of query services, specifying their input and output parameters. In contrast, the *semantical level* relates the query services with the underlying domain terminology. We propose WDSDL (Web Data Source Description Language) to describe Web Data Sources. WDSDL consists of a lower, signature level that covers the plain signature, and of a second, semantical level that relates it with the notions of the actual domain ontology.

4.1 Characteristics of Web Data Sources

Conceptually, every Web Data Source can be seen as an n-ary predicate $q(\overline{x}) = q(x_1, \ldots, x_n)$ (its *characteristic predicate*, which contains all input/output mappings) over variables $\{x_1, \ldots, x_n\}$.

The different interaction patterns (i.e., forms or Web Service invocations) with a Web Data Source can be regarded as predefined views over its characteristic predicate, which can also not be queried in general, but only via a restricted access pattern, i.e., certain input arguments must be given to return the corresponding output values. The modeling associates each view v with a unique identifying URI (which is not the URL of the corresponding Web form, but "simply" some RDF URI). For each view, some attributes $\overline{qin} = \{xin_1, \ldots, xin_k\} \subseteq \{x_1, \ldots, x_n\}$ act as inputs, others $\overline{qout} = \{xout_1, \ldots, xout_m\} \subseteq \overline{x} \setminus \overline{qin}$ act as outputs. In the remainder of the paper we call this the *signature* of the view, and denote it by $\overline{qout} \leftarrow v(\overline{qin})$.

4.2 Technical Level

As mentioned in the introduction, Web Data Sources can be distinguished into two types: *Deep Web Sources* where the primary interface is given by a Web form,

and *Web Services* that are machine-accessible via protocols like REST or SOAP, but do not provide a declarative interface in any query language. For both types, generic wrapper interfacing languages have been designed, called *DWQL (Deep Web Query Language)* that allows to pose queries against Deep Web Sources, and *WSQL (Web Service Query Language)* that supports the generic querying of (REST- and SOAP-based) Web Services. To the outside, DWQL and WSQL provide a uniform set-oriented interface of the generic form $\overline{qout} \leftarrow v(\overline{qin})$: given a set r of tuples of variable bindings over the input variables $\overline{qin}$ for that view, the wrapper returns the set of tuples $\pi[\overline{qout} \cup \overline{qin}](q \bowtie r)$ (which is the same as $\pi[\overline{qout} \cup \overline{qin}](q \ltimes r)$ since $\mathsf{Var}(r) = \overline{qin} \subseteq \overline{x} = \mathsf{Var}(q)$) where q is the characteristic predicate of the source as introduced above[1].

The actual wrappers that map the Web Data Sources to DWQL or WSQL have to be programmed manually; as has been done for the above Sources (A)-(D). The annotations start above this level and annotate the wrapped source.

4.3 Signature Level

The first step of assigning a *signature* consists of *naming* the variables of the characteristic predicate by so-called *tags*, similar to the way they are used in social bookmarking sites.

Example 1. *Consider the sample data sources from above that provide information about flight schedules. The sources (A) and (B) provide nearly the same schema (and similar views over it), while source (B) is implicitly restricted to flights of Lufthansa. Thus, both sources can uniformly be tagged as follows (the chosen tag names are not necessarily the same as the ontology notions – they will be correlated explicitly later):*

(A) connection(from, to, date, airline, fcode, deptTime, arrTime, price).
view: (airline, fcode, deptTime, arrTime, price) ← travenjoyConns(from, to, date).
(B) connection(from, to, date, fcode, deptTime, arrTime, price).
view: (fcode, deptTime, arrTime, price) ← LHConnections(from, to, date).

Source (C) provides only a restricted characteristic predicate and view (which is actually a projection of the above, where all concrete information about the flights is missing):

(C) flight(airline, from, to).
view: (airline, to) ← flies(from).

For source (D), the reference airport, Frankfurt (FRA), is fixed. The formal characterization is less simple than above, where the characteristic predicate could be seen as the contents of an actual database: due to the possibility to enter a desired departure/arrival time in the views which is compared *as an additional condition to the data, the characteristic predicate is an infinite view over the possibly materialized database. We start with the view definitions:*

[1] π denotes relational projection, $\bowtie$ and $\ltimes$ denote natural join and left semijoin, respectively.

(D1) (airline, fcode, depT, arrT) ← fromFRAtoX(to, date, dDepT).
(D2) (airline, fcode, depT, arrT) ← fromXtoFRA(from, date, dArrT).
(D3) (to, airline, fcode, depT, arrT) ← fromFRA(date, dDepT).

The underlying characteristic predicate in this case is the universal relation, i.e., the join of all views, after extending (D1) and (D3) with from: "FRA", (D2) with to: "FRA", and has thus the format

(D) flight_D(from, to, date, airline, fcode, dDepT, depT, dArrT, arrT).

As motivated above, it is defined as an intensional predicate over the database that represents all valid answers:

$\mathsf{flight}_D(f, t, d, a, c, dDT, dT, dAT, aT)$ *holds whenever there is an actual flight* (f, t, d, a, c, dT, aT) *such that* $dT > dDT$ *and* $aT > dAT$.

The above description is human-readable since intuitive names are used, but it is not machine-usable since the tag names are not (yet) formally associated to the domain notions. Even more, it does not specify the actual correlation between the values of the result tuple. Formally, it can be considered as a *schema*, similar to an SQL schema, that assigns names to columns of the relation defined by the characteristic predicate, e.g., for source (D): flight_D(from, to, date, airline, fcode, dDepT, depT, dArrT, arrT). The semantics what existence of a tuple $(a, b, c, d, e, f, g, h, i) \in \mathsf{flight}_D$ means in terms of the domain ontology is not yet specified.

4.4 Semantical Level

Assuming a relational modeling of the domain, assertions about the characteristic predicate can be expressed in terms of the ontology. As usual for ontologies formulated in RDF, the class names are unary predicates, and the property names are binary predicates. For instance, flight_D can be axiomatized as

$$
\begin{aligned}
(*)\quad & \mathsf{flight}_D(a,b,c,d,e,f,g,h,i) \Leftrightarrow \exists v,w,x,y,z : \\
& \mathsf{Flight}(v) \wedge \mathsf{flightCode}(v,e) \wedge \mathsf{Airline}(w) \wedge \mathsf{operatedBy}(v,w) \wedge \mathsf{name}(w,d) \wedge \\
& \mathsf{Airport}(x) \wedge \mathsf{from}(v,x) \wedge \mathsf{code}(x,a) \wedge \\
& \mathsf{Airport}(y) \wedge \mathsf{to}(v,y) \wedge \mathsf{code}(y,b) \wedge \\
& \mathsf{deptTime}(v,g) \wedge f < g \wedge \mathsf{arrTime}(v,i) \wedge h < i \wedge \\
& \mathsf{BookableConn}(z) \wedge \mathsf{instanceOfConn}(z,v) \wedge \mathsf{date}(z,c).
\end{aligned}
$$

which is a *local-as-view* [14] mapping. The mappings of the individual views of a source are obtained as projections.

For applications in the *Semantic Web*, underlying features of the Semantic Web and its data model, RDF, can be exploited:

- Source annotations in RDF format wrt. the agreed target ontology of the application domain can be provided in a homogeneous, web-wide format,
- Description Logic reasoners are available, which implement –depending on the chosen DL– expressive, but decidable fragments of first-order logic.

The following section introduces the WDSDL ontology for expressing Web Data Source Descriptions themselves in RDF.

5 WDSDL: The Ontology of Web Data Source Descriptions

5.1 Signature Level in RDF

The central notion of Web Data Sources is, as introduced above, the characteristic predicate, where the Web Data Source provides one or more views upon. The WDSDL ontology first provides notions for describing the characteristic predicate by enumerating the tags used by the source. In RDF terminology, the tags are just resources that have a name, and that can be annotated by datatypes, and optionally by dimensions (i.e., "time", "price", etc.), formats (e.g, "HH:mm" for times), and units (meters, miles, $, € etc.) (cf. [12]). The WDSDL vocabulary continues with describing the views provided by the source, and for each view which tags are used in it as input or output. On this level, there is not yet a relationship between the tags and the domain notions.

The WDSDL ontology on the signature level is given in Figure 2, a sample WDSDL Source Description for source (D) is given in Figure 3.

```
@prefix : <http://www.semwebtech.org/languages/2008/wdsdl#>.
:WebDataSource a owl:Class.
:baseURL rdfs:domain :WebDataSource; rdfs:range xsd:anyURI.
:DeepWebSource rdfs:subClassOf :WebDataSource.
:WebServiceSource rdfs:subClassOf :WebDataSource.
:View a owl:Class.
:providesView rdfs:domain :WebDataSource; rdfs:range :View.
:Tag a owl:Class.
:hasTag rdfs:domain :WebDataSource; rdfs:range :Tag.
  # property: tags (called variables in the logic-oriented setting) have a name
:hasInputVariable rdfs:domain :View; rdfs:range :Tag.
:hasOutputVariable rdfs:domain :View; rdfs:range :Tag.
```

Fig. 2. The WDSDL Ontology Part I: Technical and Signature Level

Example 2 (Web Data Source Description). *Consider the WDSDL Source Description for source (D) given in Figure 3. The* RDF blank nodes *of the form "_:xxx" act only internally as tag identifiers. To the outside, only the tag names are known as illustrated by the SPARQL query*

```
prefix :  <http://www.semwebtech.org/languages/2008/wdsdl#>
select  ?N
from    RDF data given in Figure 3
where { <bla://views/travel/fra/fromFRAtoX> :hasOutputVariable [ :name ?N ] }
```

that can be used to query the names of the tags that are output variables of the view. It yields the variable bindings N/"to", N/"airline", N/"flightCode", N/"deptTime", N/"arrTime".

```
@prefix : <http://www.semwebtech.org/languages/2008/wdsdl#> .
@prefix travel: <http://www.semwebtech.org/domains/2006/travel#> .

<bla://views/travel/fra> a :WebDataSource;
  :baseURL <http://www.airportcity-frankfurt.de/>;
  :providesView <bla://views/travel/fra/fromFRAtoX>,
                <bla://views/travel/fra/toFRAfromX>,
                <bla://views/travel/fra/fromFRA>;
  :hasTag _:from,_:to, _:date, _:airline, _:fcode, _:depT, _:arrT, _:dDepT, _:dArrT.

_:from a :Tag; :name "from"; :datatype xsd:string.
_:to a :Tag; :name "to"; :datatype xsd:string.
_:date a :Tag; :name "date"; :datatype xsd:date; :format "dd.MM.yyyy".
_:airline a :Tag; :name "airline"; :datatype xsd:string.
_:fcode a :Tag; :name "flightCode"; :datatype xsd:string.
_:depT a :Tag; :name "deptTime"; :datatype xsd:time; :format "HH:mm".
_:arrT a :Tag; :name "arrTime"; :datatype xsd:time; :format "HH:mm".
_:dDepT a :Tag; :name "desiredDeptTime"; :datatype xsd:time; :format "HH:mm".
_:dArrT a :Tag; :name "desiredArrTime"; :datatype xsd:time; :format "HH:mm".
<bla://views/travel/fra/fromFRAtoX> a :View;
  :hasInputVariable _:to, _:date, _:dDepT;
  :hasOutputVariable _:airline, _:fcode, _:depT, _:arrT.
<bla://views/travel/fra/toFRAfromX> a :View;
  :hasInputVariable _:from, _:date, _:dArrT;
  :hasOutputVariable _:airline, _:fcode, _:depT, _:arrT.
<bla://views/travel/fra/fromFRA> a :View;
  :hasInputVariable _:date, _:dDepT;
  :hasOutputVariable _:to, _:airline, _:fcode, _:depT, _:arrT.
```

Fig. 3. WDSDL Source Description for Source (D): Frankfurt Airport

5.2 Semantical Level: Correlating to the Domain Ontology

The description on the semantical level refers to the description of the sources on the signature level, i.e., the tags (corresponding to the variables of the *characteristic predicate*), and relates its components to the actual domain notions.

The correlation has to cover what has been expressed by the predicate logic formula (*) above. The basic idea is to express the relationships between objects of the application domain in terms of a relevant prototype fragment of the RDF graph, and to associate the tags with the corresponding nodes.

Example 3. *For Source (D), the relationship between the values of* from, to, date, airline, fcode, dDepT, depT, dArrT, *and* arrT *is induced by instances of the class* travel:Flight *as follows:* fcode, depT, arrT *are the respective attributes of the flight itself, while* from *and* to *are actually the airport codes of the origin and destination airport,* airline *is the name of the airline, there must be an instance of the flight on the given* date, *and* depT/arrT *must be later than* dDepT/dArrT. *Fig. 4 depicts this situation for sample data. Note that Source (D) does not cover all aspects of the ontology, e.g., the price of the connection is not available in it.*

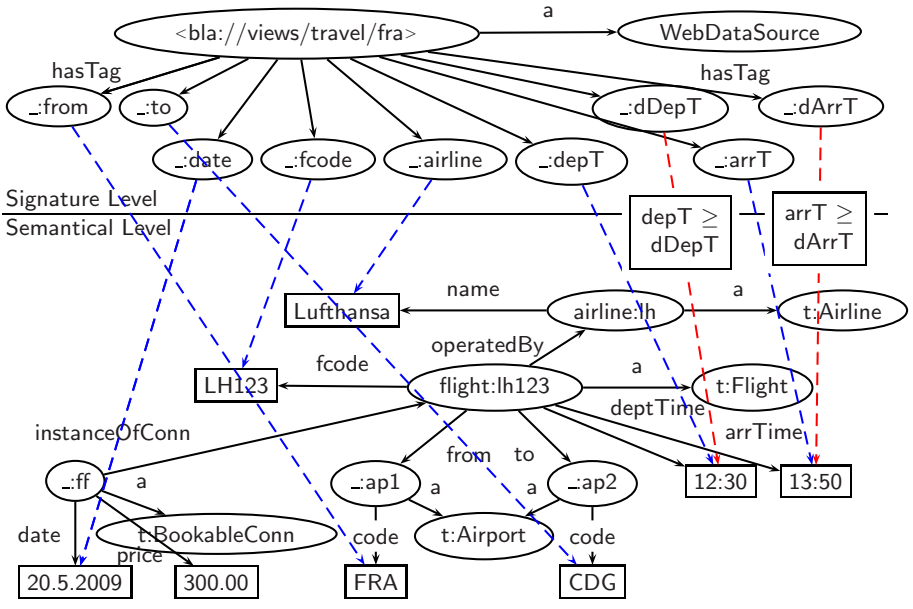


Fig. 4. Semantic Mapping from Source (D) to the Ontology

Temptation: Canonical Instance as Prototype Triples. At a first glance, this correlation could be represented by RDF triples where tag names and existential variables are replaced by literal constants and resource constants, respectively (similar to the idea of the definition of a canonical instance of a conjunctive query in relational database theory [7]). The canonical instance as triples for Source (D) is given in Figure 5 where the replacements are highlighted by boxes. Since the canonical instance is only concerned with the structural aspects, the additional conditions on desired departure/arrival time are not part of it.

```
prefix t : <http://www.semwebtech.org/domains/2006/travel#>.
_:flightR a t:Flight; t:operatedBy [ a t:Airline; t:name "airlineTag" ];
  t:from [a t:Airport; t:code "fromTag" ];
  t:to [a t:Airport; t:code "toTag" ] ;
  t:flightCode "fcodeTag" ; t:deptTime "depTTag" ; t:arrTime "arrTTag" .
_:bconnR a t:BookableConn; t:instanceOfConn _:flightR ; t:date "dateTag" .
```

Fig. 5. Canonical instance of Source (D) as RDF triples

Problems. The first problem is, how this prototype description of the pattern is related in RDF to the WDSDL source description. It can be put in a file at a url u, and a triple (<bla://views/travel/fra> wdsdl:hasPrototypePattern u) can

be added. This file is then intended for separated use as pattern, and does not make the description itself part of the Semantic Web.

Nevertheless, since the file at URI u is accessible in the Web, it implicitly *contributes* its triples to the "world-wide RDF database" in which also the triples from the ontology in Section 3 can be found.

If some RDF application (e.g., a Semantic Web search engine) encounters these triples, it cannot know that they are a prototype that is intended for separate use only, and will add them to its knowledge base. This leads to wrong facts (e.g., the resource with URI local0815:flightR is considered to be an instance of the class travel:Flight and is operated by a blank node local0815#id4711 that represents an airline that has a travel:name property with value "airlineTag"). Obviously, having the canonical instance in the Web together with the actual instances is a problem in the RDF world. Instead, an RDF-based language for *describing* patterns has to be defined and used.

Source Annotation Statements. The annotation on the semantic level consists of a set of statements that *refer* to the tags and optionally also refer to additional placeholders acting as existential variables (the RDF *blank nodes* in Figure 5). The statements *describe* the graph pattern that is asserted to hold for all answers returned by any query against the data source. For this, WDSDL makes use of the mechanism of *reification*: it talks *about* statements by wdsdl:AnnotationStatements. These statements relate

- tags (which have already been introduced as resources of the class wdsdl:Tag),
- local variables,
- constants if required,
- the notions of the application domain.

Additionally, constraints (using the operators from the XPath Functions and Operators namespace) are expressed by wdsdl:AnnotationConstraints[2]. Figure 6 shows the second part of the WDSDL ontology that covers the annotations on the semantical level.

Example 4. *The WDSDL source annotation of Source (D) is given in Figure 7. It has to be added to the source description on the signature level that has been given in Figure 3 since it refers to the same local tag resource URIs (the tag URIs are indicated by frames). Note that it additionally contains local identifiers (of the form _:xxxV) for the blank nodes. Note also that each of the views has an additional constraint that specifies _:from and _:to, respectively, to be "FRA".*

If the AnnotationStatements were materialized, they would exactly result in the canonical instance given in Section 5.2. Note that the same reification strategy

[2] The comparison operators used in these constraints are the same as for XPath/XQuery. As the W3C documents do explicitly list these operators like op:time-less-than, but do not assign a namespace URI to the op namespace prefix. For that, we temporarily assign it within the wdsdl namespace.

```
@prefix : <http://www.semwebtech.org/languages/2008/wdsdl#> .
:Annotation a owl:Class.
:hasAnnotation rdfs:domain [ owl:unionOf (:WebDataSource :View)] ;
  rdfs:range :Annotation.
:AnnotationStatement a owl:Class.    :AnnotationConstraint a owl:Class.
:hasAnnotationStatement rdfs:domain :Annotation; rdfs:range :AnnotationStatement.
:hasAnnotationConstraint rdfs:domain :Annotation; rdfs:range :AnnotationConstraint.
```

Fig. 6. The WDSDL Ontology Part II: Semantical Level

```
@prefix wdsdl: <http://www.semwebtech.org/languages/2008/wdsdl#> .
@prefix travel: <http://www.semwebtech.org/domains/2006/travel#> .
<bla://views/travel/fra/> wdsdl:hasAnnotation
[ wdsdl:localVar _:flightV, _:airlineV, _:airp1V, _:airp2V, _:bconnV ;
  wdsdl:hasAnnotationStatement
   [ rdf:subject _:flightV; rdf:predicate rdf:type; rdf:object travel:Flight],
   [ rdf:subject _:flightV; rdf:predicate travel:operatedBy; rdf:object _:airlineV ],
   [ rdf:subject _:airlineV; rdf:predicate rdf:type; rdf:object travel:Airline],
   [ rdf:subject _:airlineV; rdf:predicate travel:name; rdf:object _:airline ],
   [ rdf:subject _:flightV; rdf:predicate travel:from; rdf:object _:airp1V],
   [ rdf:subject _:flightV; rdf:predicate travel:to; rdf:object _:airp2V],
   [ rdf:subject _:flightV; rdf:predicate travel:flightCode; rdf:object _:code ],
   [ rdf:subject _:airp1V; rdf:predicate travel:code; rdf:object _:from ],
   [ rdf:subject _:airp2V; rdf:predicate travel:code; rdf:object _:to ],
   [ rdf:subject _:flightV; rdf:predicate travel:deptTime; rdf:object _:depT ],
   [ rdf:subject _:flightV; rdf:predicate travel:arrTime; rdf:object _:arrT ],
   [ rdf:subject _:bconnV; rdf:predicate rdf:type; rdf:object travel:BookableConn],
   [ rdf:subject _:bconnV; rdf:predicate travel:instanceOfConn; rdf:object _:flightV ],
   [ rdf:subject _:bconnV; rdf:predicate travel:date; rdf:object _:date ];
  wdsdl:hasAnnotationConstraint
   [ rdf:subject _:depT ; wdsdl:comparator wdsdl:time-less-than ; rdf:object _:dDepT ],
   [ rdf:subject _:arrT ; wdsdl:comparator wdsdl:time-less-than ; rdf:object _:dArrT ] ].
<bla://views/travel/fra/toFRAfromX/> wdsdl:hasAnnotation
  [ wdsdl:hasAnnotationConstraint [ wdsdl:onVariable _:to ; wdsdl:hasValue "FRA"] ].
<bla://views/travel/fra/fromFRAtoX/> wdsdl:hasAnnotation
  [ wdsdl:hasAnnotationConstraint [ wdsdl:onVariable _:from ; wdsdl:hasValue "FRA"] ].
<bla://views/travel/fra/fromFRA/> wdsdl:hasAnnotation
  [ wdsdl:hasAnnotationConstraint [ wdsdl:onVariable _:from ; wdsdl:hasValue "FRA"] ].
```

Fig. 7. WDSDL Source Annotation for Source D: Frankfurt Airport

is followed e.g. in OWL-2 [21] when asserting that a statement does *not* hold – describe the statement and annotate it.

On the other hand, the triples described in Figure 7 also represent the formula (*) given in Section 4.4. For instance, all statements about _:flightV (that stands for the variable v in (*)) are equivalent to

$\mathsf{Flight}(v) \wedge \mathsf{operatedBy}(v, w) \wedge \mathsf{from}(v, x) \wedge \mathsf{to}(v, y) \wedge \mathsf{flightCode}(v, e) \wedge \mathsf{deptTime}(v, g) \wedge \mathsf{arrTime}(v, i)$.

6 Source Selection and Query Answering

As already mentioned above, a SPARQL query basically specifies a graph pattern that yields answer bindings by matching the pattern against RDF data. SPARQL queries to be evaluated are submitted to a Query Broker together with the input variable bindings. The task of the query broker is to identify appropriate sources to query them in a suitable order, and to combine the answers. For this, the Query Broker is aware of the WDSDL descriptions of the sources that it can use to answer the query. The identification of Web Data Sources and their views that can contribute to the answer to a SPARQL query can then be reduced to finding suitable structural overlappings between the query and the views provided by Web Data Sources.

We will first illustrate the problem a bit more. Then we analyze the relationship with the areas of query answering using views [30,11,14] and querying data under access limitations [16,15,19,33,6].

6.1 Motivational Examples

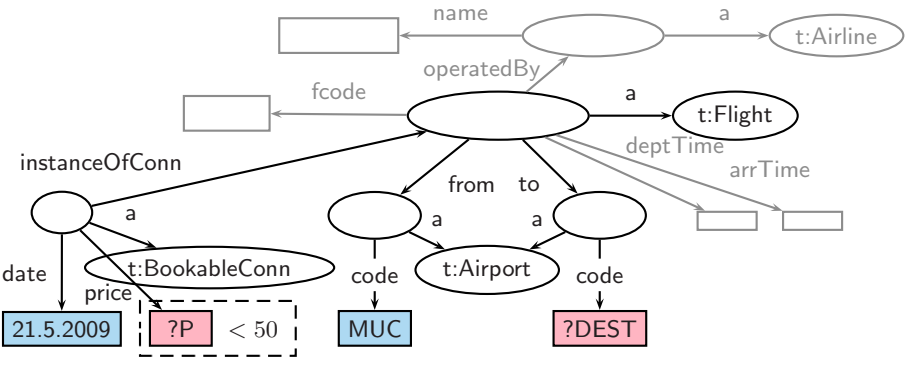
Example 5 (Coverage between Sources and a Query). *Consider the query "where can we go from MUC on 21.5.2009 for less than 50€?".*
The SPARQL query is (without redundant class constraints)

```
prefix  t: <http://www.semwebtech.org/domains/2006/travel#>
select  ?DEST ?P
where {?C a t:Flight; t:from [t:code "MUC"]; t:to [ t:code ?DEST] .
       ?BC t:instanceOfConn ?C; t:date "21-05-2009"; t:price ?P .
       filter (?P < 50) }
```

The prototype pattern of the query is shown in the upper part of Figure 8. The black portion is the prototype, while the gray portion is the part of the ontology that is not relevant to the query.

Consider first source (A) whose prototype graph is shown in the lower part of Figure 8: obviously, (A)'s prototype covers the query prototype (and additionally departure, arrival, and code of the flight connection). Analogously, also source (B) covers the query. Sources (C) and (D) do not cover the query since they do not contain the price (for (D) compare the upper part of Figure 8 with the mapping shown in Figure 4).

The above example illustrates the basic idea how to map queries onto the *annotated* sources, by using their canonical RDF instances. Nevertheless, there is not necessarily any source that completely covers the query. Additionally, in presence of limited access patterns, complete coverage does not guarantee that the source can be used for actually answering the query, but the input-output-characteristics must also be checked for compatibility.



Above: Query Prototype as a Fragment of the Ontology given in Figure 4

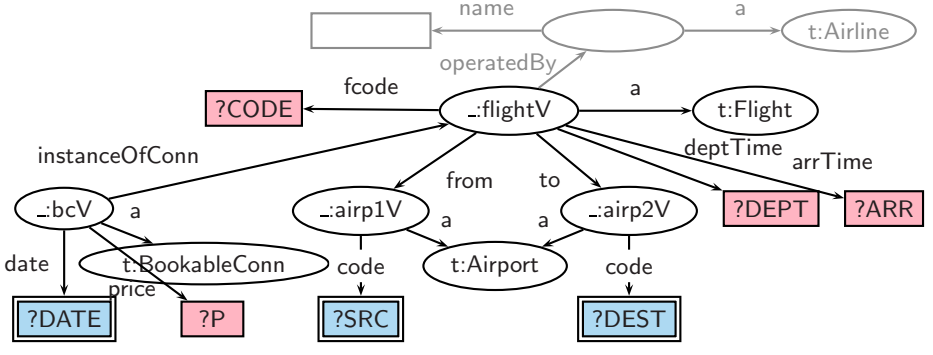


Fig. 8. Query Prototype as a Fragment of the Ontology (upper part) and coverage of the view provided by source (A) of the ontology (lower part). Input variables are double-framed.

Example 6 (Access Limitations). *Reconsider Figure 8 that illustrates that Source (A) covers the notions required in the query. Nevertheless, Source (A) provides the only view*

(airline, fcode, deptTime, arrTime, price) ← travenjoyConns(from, to, date)

*that requires the destination (*to*) to be an input parameter. Similarly, the only view on Source (B) also declares* to *to be an input parameter. Both cannot be used immediately, since no destination is known for answering the query.*

Furthermore, the views provided by Source (D) require from *or* to *to be "FRA" (so only view <*bla://views/travel/fra/toFRAfromX/*> would only have a small overlap with the flights relevant for the query, namely those from "MUC" to "FRA").*

Source (C) is applicable, which is able to return values for to*, and also for* airline*. The WDSDL Source Annotation is given in Figure 9; its relationship with the query prototype is depicted in Figure 10. Thus, it is able to return the*

destination airports reachable from Munich, and additionally returns knowledge (the name of the airline) which is not required by the query, but can also be used for the next step.

After that, the input requirements of Sources (A) and (B) are satisfied (i.e., date, start and now also the destinations are known), but (B) is applicable only for connections by Lufthansa. Both will be queried, and the resulting connections that are below 50€ will be returned. Note that this implicitly makes use of currency conversion described in [12] between $ and € if necessary.

```
@prefix travel: <http://www.semwebtech.org/domains/2006/travel#> .

<bla://views/travel/airdb/> :hasAnnotation
[    :localVar _:flightV, _:airlineV, _:airp1V, _:airp2V ;
  :hasAnnotationStatement
    [ rdf:subject _:flightV; rdf:predicate rdf:type; rdf:object travel:Flight],
    [ rdf:subject _:flightV; rdf:predicate travel:operatedBy; rdf:object _:airlineV ],
    [ rdf:subject _:airlineV; rdf:predicate rdf:type; rdf:object travel:Airline],
    [ rdf:subject _:airlineV; rdf:predicate travel:name; rdf:object _:airline ],
    [ rdf:subject _:flightV; rdf:predicate travel:from; rdf:object _:airp1V],
    [ rdf:subject _:flightV; rdf:predicate travel:to; rdf:object _:airp2V],
    [ rdf:subject _:airp1V; rdf:predicate travel:code; rdf:object _:from ],
    [ rdf:subject _:airp2V; rdf:predicate travel:code; rdf:object _:to ] ].
```

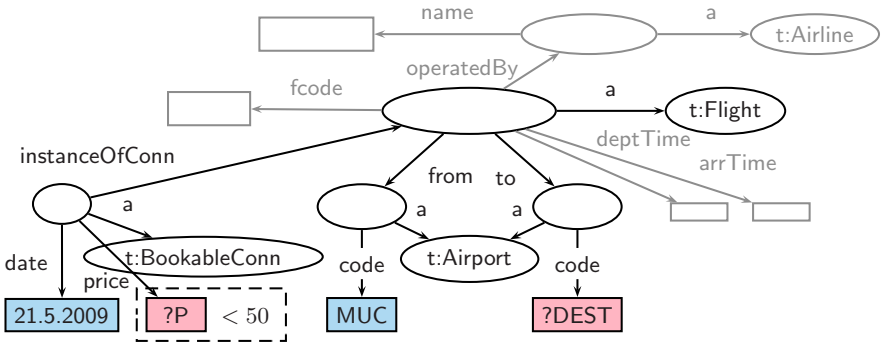
Fig. 9. WDSDL Source Annotation for Source (C): airdb

6.2 Query Evaluation

The actual query evaluation can then make use of solutions of two well-investigated issues: *query answering using views* for obtaining possible rewritings, and *querying data under access limitations* for checking which rewritings are compatible with the access limitations, and obtaining executable query plans.

Query Answering Using Views is e.g. investigated in [30,11,14]. Here, *GAV (global-as-view)* and *LAV (local-as-view)* approaches can be distinguished. Intuitively, LAV is the more obvious way: given a global schema, the local schemas are expressed as views over it. In contrast, GAV expresses the global schema in terms of the local ones.

In our setting, the global schema is given as an RDF schema, i.e., unary and binary predicates, and the sources are given as predicates, i.e., a relational schema. As already mentioned in Section 4.4, our approach is based on a *LAV* mapping. The LAV mappings like (*) in Section 4.4 can be derived automatically from the WDSDL Source Annotations. The usual algorithms, like MiniCon [24] construct a union of conjunctive queries over the available views that yields all obtainable answers.



Above: Query Prototype as a Fragment of the Ontology given in Figure 4

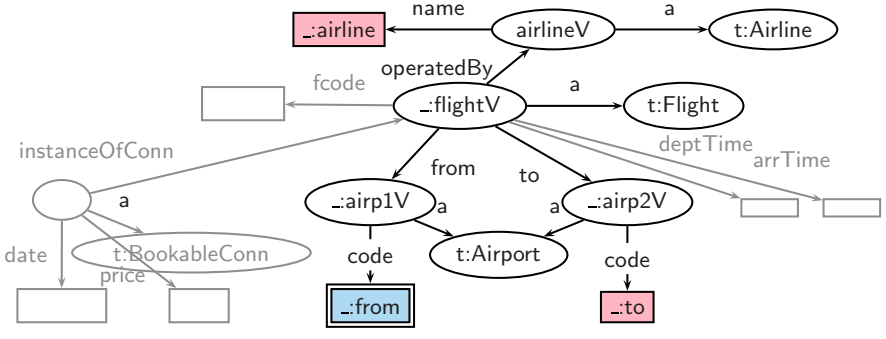


Fig. 10. Query Prototype as a Fragment of the Ontology (upper part) and coverage of the view provided by source (C) of the ontology (lower part)

Example 7. *In the running example, the original SPARQL query is seen as a predicate q(To,Price) where the following conjunctive queries contribute results:*

1.) q(To, Price) :-
connection$_A$("MUC", To, "21.05.2009", _Airl, _FCode, _DT, _AT, Price).
2.) q(To, Price) :- flight$_C$(_Airl, "MUC", To),
connection$_A$("MUC", To, "21.05.2009", _Airl, _FCode, _DT, _AT, Price).
3.) q(To, Price) :- flight$_C$("Lufthansa", "MUC", To),
connection$_B$("MUC", To, "21.05.2009", _FCode, _DT, _AT, Price).

Note that the second one is –on this stage– redundant as it is a refinement of the first one. Also the third one is a subset of the second one (the Lufthansa flights).

In the given setting with limited access capabilities, one must not eliminate the redundant rewritings yet, since some of them may turn out not to be executable.

Querying Data under Access Limitations is concerned with the case where the reformulation and mapping to the sources is already done, and a concrete plan is to be found how to do the evaluation. Concretely, two algorithms have been presented in [33,6].

They have to be applied to each of the found rewritings, resulting in an executable plan for those rewritings where such a plan exists. Only after that, redundant rewritings/plans can be removed.

Example 8. *In the running example, from the above rewritings, (1.) is not executable since Source (A) expects* To *as an input. (2.) and (3.) are executable since the subquery containing* flight_C *is supported by (C)'s only view, yielding the possible solutions for* To*. With these, Sources (A) (for all airlines) and (B) (for Lufthansa) can be queried. After that, (3.) can be dropped since it is redundant.*

The query broker executes the obtained query plans and returns the resulting bindings as answers of the original SPARQL query.

7 Related Work

The core related aspects of *Query Answering Using Views* and *Querying Data under Access Limitations* have been discussed in Section 6.2.

Our work is also related to the field of *Semantic Web Services*. There, several different formalisms for enriching Web Services with semantic annotations have been proposed. In [8] a layered language architecture (similar to the standard Semantic Web layer cake [13]) is introduced to enable automatic Web Service discovery, composition and execution. The framework considers different aspects of Web Services, such as the heterogeneity in data representations and the pre- and postconditions of Web Service invocations. DAML-S [5] enhances the WSDL descriptions of input and output messages to cover abstract types for each message part and also comprises a process model ontology, which can be used to support automated Web Service invocation, composition, and interoperation. [17] describes how the sucessor of DAML-S, OWL-S [18], fits into the picture of the "Semantic Annotations for WSDL and XML Schema" (SAWSDL), a standard set by the W3C. SAWSDL defines a fixed set of WSDL extension attributes, which allow to use elements of an external Semantic Web framework, e.g. for mapping message parts to abstract types. Finally, [1] proposes the "Semantic Web Process Description Language" (SWPDL), which is based on an OWL ontology and allows to describe (composite) Web Services, as well as (collections of) Web pages.

Our approach is also based on several layers, which have a clear cut purpose and each level builds neatly on top of the lower level. All semantic annotations are given in standard RDF and OWL constructs and are thus Semantic Web documents. Additionally, we support a more generic and abstract notion of Web Data Sources, which allows us to represent and use heterogeneous data sources, ranging from Deep Web Sources to Web Services. This is possible because we do not rely on a specific low level service specification, such as WSDL.

One of the main incentives for the semantic annotations of Web Services is the ability to do *automatic selection or matchmaking* of these services. Current approaches to this problem usually employ some kind of *degree of match* metric to find relevant services [31,23,3,22]. For the composition of the thus found services

sometimes additional user interaction is required [27,2]. In our approach we are only concerned with querying, and with *exact* matches for the query description.

8 Conclusion

The WDSDL ontology serves for annotating Web Data Sources to relate them to the terminology of their domain ontology. These annotations enable a WDSDL-based query broker to handle SPARQL queries that are stated in terms of the domain ontology by selecting appropriate sources to stepwise process the complete query or portions of it, and to combine their answers to answer the original query. The approach is currently under implementation.

References

1. Agarwal, S.: Specification of Invocable Semantic Web Resources. In: ICWS, pp. 124–131. IEEE Computer Society Press, Los Alamitos (2004)
2. Agarwal, S., Studer, R.: Automatic Matchmaking of Web Services. In: ICWS, pp. 45–54. IEEE Computer Society Press, Los Alamitos (2006)
3. Benatallah, B., Hacid, M.-S., Rey, C., Toumani, F.: Request Rewriting-based Web Service Discovery. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 242–257. Springer, Heidelberg (2003)
4. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of Multi-domain Queries on the Web. PVLDB 1(1), 562–573 (2008)
5. Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D., McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K.: DAML-S: Web service description for the semantic web. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 348–363. Springer, Heidelberg (2002)
6. Calì, A., Martinenghi, D.: Conjunctive query containment under access limitations. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 326–340. Springer, Heidelberg (2008)
7. Chandra, A.K., Merlin, P.M.: Optimal Implementation of Conjunctive Queries in Relational Data Bases. In: STOC, pp. 77–90. ACM Press, New York (1977)
8. de Bruijn, J., Lausen, H., Polleres, A., Fensel, D.: The Web Service Modeling Language WSML: An Overview. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 590–604. Springer, Heidelberg (2006)
9. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, California (2000)
10. Florescu, D., Levy, A.Y., Mendelzon, A.O.: Database Techniques for the World-Wide Web: A Survey. SIGMOD Record 27(3), 59–74 (1998)
11. Halevy, A.Y.: Answering queries using views: A survey. VLDB J. 10(4), 270–294 (2001)
12. Hornung, T., May, W.: Deep web queries in a semantic web environment. In: Workshop on Advances in Accessing Deep Web (ADW). LNBIP, vol. 37, pp. 39–50. Springer, Heidelberg (2009)
13. Horrocks, I., Parsia, B., Patel-Schneider, P.F., Hendler, J.A.: Semantic Web Architecture: Stack or Two Towers. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 37–41. Springer, Heidelberg (2005)

14. Lenzerini, M.: Data integration: A theoretical perspective. In: PODS, pp. 233–246. ACM Press, New York (2002)
15. Li, C.: Computing complete answers to queries in the presence of limited access patterns. VLDB J. 12(3), 211–227 (2003)
16. Li, C., Chang, E.Y.: Answering queries with useful bindings. ACM Trans. Database Syst. 26(3), 313–343 (2001)
17. Martin, D., Paolucci, M., Wagner, M.: Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC/ISWC 2007. LNCS, vol. 4825, pp. 340–352. Springer, Heidelberg (2007)
18. Martin, D.L., Burstein, M.H., McDermott, D.V., McIlraith, S.A., Paolucci, M., Sycara, K.P., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing Semantics to Web Services with OWL-S. In: World Wide Web, pp. 243–277 (2007)
19. Nash, A., Ludäscher, B.: Processing Unions of Conjunctive Queries with Negation under Limited Access Patterns. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 422–440. Springer, Heidelberg (2004)
20. OWL Web Ontology Language (2004), http://www.w3.org/TR/owl-features/
21. OWL 2 Web Ontology Language (2009), http://www.w3.org/2007/OWL/wiki/OWL_Working_Group (work in progess)
22. Pantazoglou, M., Tsalgatidou, A., Athanasopoulos, G.: Quantified Matchmaking of Heterogeneous Services. In: Aberer, K., Peng, Z., Rundensteiner, E.A., Zhang, Y., Li, X. (eds.) WISE 2006. LNCS, vol. 4255, pp. 144–155. Springer, Heidelberg (2006)
23. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic Matching of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
24. Pottinger, R., Halevy, A.Y.: Minicon: A scalable algorithm for answering queries using views. VLDB J. 10(2-3), 182–198 (2001)
25. Resource Description Framework, RDF (2000), http://www.w3.org/RDF
26. Resource Description Framework (RDF) Schema specification (2000), http://www.w3.org/TR/rdf-schema/
27. Sirin, E., Parsia, B., Hendler, J.A.: Filtering and Selecting Semantic Web Services with Interactive Composition Techniques. IEEE Intelligent Systems 19(4), 42–49 (2004)
28. SPARQL Query Language for RDF (2006), http://www.w3.org/TR/rdf-sparql-query/
29. Turtle - Terse RDF Triple Language, http://www.dajobe.org/2004/01/turtle/
30. Ullman, J.D.: Information integration using logical views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1996)
31. Vaculín, R., Chen, H., Neruda, R., Sycara, K.P.: Modeling and Discovery of Data Providing Services. In: ICWS, pp. 54–61. IEEE Computer Society Press, Los Alamitos (2008)
32. XML Schema (1999), http://www.w3.org/XML/Schema
33. Yang, G., Kifer, M., Chaudhri, V.K.: Efficiently ordering subgoals with access constraints. In: PODS, pp. 183–192. ACM, New York (2006)