Unify the Description of Web Services and RDF Data Sources towards Uniform Data Integration

Wenfeng Zhao, Xiangwu Meng, Junliang Chen, Chuanchang Liu

State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications (BUPT), Beijing, China
zhaowenfeng@gmail.com, {mengxw, chjl, lcc3265}@bupt.edu.cn

Keywords: Web Service, Data Integration, Semantic Web, SPARQL

Abstract

It is valuable to integrate together the information provided by both Web Services and kinds of data repositories, especially RDF datasets. However, current Semantic Web Service technologies don't meet this requirement well for there coarse annotation granularity. In this paper, we propose a semantic description technology of fine granularity for the information-providing Web Service, which treats it as a special data view in terms of the RDFS/OWL ontology and thus makes it natural to be incorporated in the data integration system. Then we discuss the related match criteria of a data source against a query in this form. We also present our proof-of-concept system which integrates both RDF data sources and information-providing Web Services uniformly. At runtime the queries posed by the user against the domain ontology results in dynamic invocation of the matched Web Services as well as evaluation of SPARQL queries on RDF datasets.

1 Introduction

With the progress of Semantic Web\(^1\) research in recent years, there emerge more and more RDF datasets on the web ([10]) and in some data-dense discipline such as health care and life science. Benefiting from the highly structured nature and the shared schema, i.e. ontology, of such data, it’s promising to integrate data from various sources together to provide a one-stop data query system for the user. Furthermore, data queries in such system can comprise complicated SQL-like structure rather than merely a keyword list.

Meanwhile, SOAP&WSDL-based Web Services\(^2\) are also emerging in recent years. As pointed out previously (e.g. [7]), Web Services fall into two categories: the information-providing one and the world-altering one. In our opinion, The former in fact is also a type of data sources as the RDF dataset, since they both share the same usage pattern that the user specifies some data to get some other ones which having a particular relation to the specified ones. And the Web Service is more suitable than RDF datasets or relational databases to deliver dynamic data such as weather information, price quotation and data gotten from sensors. So it’s important to incorporating Web Services into the data integration system too.

As the first step, it is necessary to exactly express the capability of a Web Service formally. However, current Semantic Web Service technologies - OWL-S [9], WSMO[13], WSDL-S[12], and SA-WSDL[11] - don’t meet this requirement very well. They in general treat the service input/output as a list of Description Logic concepts. But in the data query case, the data specified and required by users are all of primitive data type. So they correspond to data-type properties of concepts rather than the concepts themselves. Although information in the concepts could be mapped to the ground WSDL I/O message elements with a complex multi-to-multi XSLT mapping, this coarse-granularity style of I/O annotation makes the Web Service unlike a data source (i.e. a view on database schema).

So we put forward a fine-granularity technology to describe the capability of information-providing Web Services – Uniform Query, which itself resembles a SPARQL\(^3\) query and can be used to annotate the WSDL I/O messages clearly at leaf level. Thus the Web Service can be seen as a database view with a limited access pattern, which makes it naturally to be treated as a data source. With proper dynamic invocation mechanism, a Web Service can be registered and invoked at runtime to answer the matched data query. Thus a data integration system integrating RDF datasets and Web Services uniformly can be realized.

2 Motivating Scenario

In order to fit the dynamic nature of data provided by Web Services, our uniform data integration system should take the virtual method[5,6], rather than data warehouse-like materialized one, from the data integration research in database discipline. That means the user formulates the query in terms of the domain ontology and the system describes data sources in the same ontology (or those having mapping defined to it), so that a single query can be evaluated by the system on all data sources at runtime. The data sources are manually registered into the system by the administrator in background. The architecture of the system, called USDIS

---

\(^1\) http://www.w3.org/2001/sw/

\(^2\) http://www.w3.org/2002/ws/

\(^3\) An RDF query language from w3c. See http://www.w3.org/TR/rdf-sparql-query/
(Uniform Semantic Data Integration System), is shown in Figure 1.

As to the query language, we need a variant of SPARQL. On one hand, SPARQL is expressive enough and the standard language for querying RDF data. If the data query is expressed in SPARQL or a less expressive form, it can be directly evaluated on SPARQL-enabled RDF datasets. On the other hand, the Web Service must undergo a matchmaking procedure against the query before taking its outputs as the query result. But SPARQL can’t describe the capability of a Web Service exactly because there is no explicit concept of “input” in it.

In fact, SPARQL is able to express a data query or a data view, but a Web Service is not a simple data view, but a data view with access pattern limitation which means some certain fields must be bound when to commit a query to the view [4]. So a little enhancement need be done on SPARQL. In addition, in order to design an easy-to-use query formulation interface, some complicated part of SPARQL may be abandoned in USDIS.

3 The Uniform Query

Regarding to the above issues, we purpose the new description mechanism, Uniform Query, to uniformly describe data queries and the capability of Web Services. The intuition behind it is: given a data model composed by some RDFS/OWL individuals and limiting values of some attributes when specified those of some others.

3.1 Definitions

Definition 1. The Query Pattern Skeleton QPS is a weakly connected directed graph of RDF blank nodes, in which for every tuple <s, p> in QPS, there is at most one o that the triple <s, p, o> appears in QPS. The data-type property of a node in QPS is called a Skeleton Property of QPS.

Definition 2. A Uniform Query Q is a tuple <Qps, Sel, I, O, C> where:

1) Qps is a Query Pattern Skeleton;

2) Sel is a set of tuple <IndvProp, ValueScope>, where IndvProp is a Skeleton Property of Qps, and ValueScope is a value span of the data-type of IndvProp.

3) I, and O, each is a vector of Skeleton Properties of Qps;

4) C is a set of OWL Classes, instances of which appear in Qps.

Here Qps corresponds to the join operations in relation algebra, and Sel corresponds to the select operations. O is the result fields of the query. C is completely derived from Qps and is explicitly listed only for the convenience of matchmaking. The main feature of the definition lies in I.

When Q is used to express a data query, I is derived from Sel so that it contains all the skeleton properties that are restricted in Sel into a single value. When Q is used to express a Web Service, it corresponds to the input vector. When describing a common data source, i.e. a data view, it points out the access pattern limitation [4] if it exists.

3.2 Grounding to Web Services

A Uniform Query (Q) is bound to an operation of the ground Web Service (WS) as:

1) For the leaf I/O elements of WS - either the primitive type message parts or the leaf nodes of complex type ones – that are relative to current data integration purpose, annotate each of them with an element from I/O of Q.

2) For the compulsory inputs of WS that never appears in the data query, such as the developer account key in some public Web Services, specify a fixed value to each of them when WS is registered in.

Here clause 1) reduces the complexity of I/O grounding from m-to-n in OWL-Ss to 1-to-n. The reason that only some selected elements rather than all of them are annotated is that some practical Web Services have a great amount of optional input parameters which might be irrespective with a particular data integration system.

In terms of OWL-Ss, Qps and Sel as a whole play the role of Effect. The individual in them corresponds to Local of OWL-S. Although Precondition isn’t differentiatied explicitly from Effect as in OWL-S, it can be derived out. All the conditions in Qps and Sel in which skeleton properties all
appear in I belong to Precondition, and others are of Effect. In practice, when invoked with input values not satisfying Precondition a Web Service always returns an empty result rather than some unpredictable ones. Thus that Precondition and Effect are not differentiated while matching a Web Service against a data query won’t incur false results.

Example 2. Annotate the ItemSearch operation of Amazon E-Commerce Service (ECS) $^4$ in purpose of book search. First we establish a Uniform Query:

\[ Q_\text{ps} : \{ \begin{align*}
& \text{Inputs: } \\
& \text{Sel: } \langle \_b, \_b, \text{book:Year} \rangle, \langle \text{[1900, +\infty)} \rangle \\
& \text{Output: } \langle \_b, \_b, \text{book:Title} \rangle, \langle \_b, \_b, \text{book:Author} \rangle, \langle \_b, \_b, \text{book:Price} \rangle, \langle \_b, \_b, \text{book:PublishYear} \rangle, \langle \_b, \_b, \text{book:AmazonRank} \rangle
\end{align*} \]

Then we ground it to WSDL elements as Table 1 and Table 2.

(As to the easy-to-use GUI, see Figure 4).

---

### Input element and Specified value

<table>
<thead>
<tr>
<th>Input element</th>
<th>Specified value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemSearch/SubscriptionId</td>
<td>1CXXYQ</td>
</tr>
<tr>
<td>ItemSearch/Request[0]/SearchIndex</td>
<td>Books</td>
</tr>
<tr>
<td>ItemSearch/Request[0]/ResponseGroup[0]</td>
<td>Medium</td>
</tr>
</tbody>
</table>

Table 1. The bound of compulsory inputs in Example 2

---

### Syntax side and Semantic side

<table>
<thead>
<tr>
<th>Syntax side</th>
<th>Semantic side</th>
</tr>
</thead>
<tbody>
<tr>
<td>ItemSearch/Request[0]/Keywords</td>
<td>$\langle _b, _b, \text{hasTopic} \rangle$</td>
</tr>
<tr>
<td>ItemSearchResponse/Items[0]/Name</td>
<td>$\langle _b, _b, \text{title} \rangle$</td>
</tr>
<tr>
<td>ItemSearchResponse/Items[0]/Title</td>
<td>$\langle _b, _b, \text{Author} \rangle$</td>
</tr>
<tr>
<td>ItemSearchResponse/Items[0]/Listing</td>
<td>$\langle _b, _b, \text{price} \rangle$</td>
</tr>
<tr>
<td>Summary/LowestNewPrice</td>
<td>...</td>
</tr>
<tr>
<td>FormatPrice</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 2. IO annotation in Example 2

Note the array index in the element path expression has 2 special features:

1. “*” is used to describe multiple records in the service output. Certainly there are at most one “*” appeared in an element. Its position – called "fork point" - must be same in all elements which have a fork point.
2. “..” is used in an array that is at the right-most node of an element path to indicate the concatenation of its items (separated by "..")
3. In all other cases, only ‘0’ is permitted for simplicity.

### 4 The Matching Algorithm

Upon receiving a query $Q$, the system fetches all the candidate data sources by filtering those registered in it through the related classes or properties. Then for each candidate source, if it’s a SPARQL-enabled one, a SPARQL query derived from $Q$ is submitted to it; otherwise, if it’s a Web Service, the system matches it against $Q$ and invokes it if matched, as depicted below.

#### 4.1 Matching Web Services Data Sources

The matching criteria of a Web Service type data source $WS$ against a data query $Q$ is that there exists an injective mapping from individuals of $Q.Qps$ to those of $WS.Qps$, under which:

1. $Q.Qps$ is entailed by $WS.Qps$ (which means as directed graphs $Q.Qps$ is subgraph of $WS.Qps$); and
2. the resulting value space of the effect of $Q Sel$ and that of $WS Sel$ have a non-empty intersection.
3. $Q.O$ should be (partially) entailed by $WS.O$; and
4. $WS.I$ should be completely entailed by $Q.I$.

Additionally, to enable dynamic service invocation, the system should during the matchmaking draw out an execution plan which:

1. binds each skeleton property of $WS.I$ with a definite value from $Q$;
2. binds as many as possible skeleton properties of $Q.O$ each with a skeleton property list of $WS.O$; and
3. for the skeleton properties of $WS.O$ that might take a value outside $Q Sel$, relates a ValueScope to it, so that the system can filter the returned values from $WS$ before presents the result to the user.

The consequent matchmaking algorithm is detailed in Algorithm 1.

**Algorithm 1.** Match a Web Service against a data query.

**Input:** a data query $Q$, a Web Service $DS$, both described in Uniform Query.

**Output:** whether they match, and an execution plan $Plan$ if yes.

1. Find out all possible injections from individuals of $Q.Qps$ to $DS.Qps$, notated as maps, in each of which for each individual $n$ in $Q.Qps$ there exists a different image individual $n'$ in $DS.Qps$ so that $n'$ entails $n$. If maps is empty, return FALSE.
2. for each map : maps) given the effect of map:
   1. set Plan as empty;
   2. if (scope(map) is a single value span, AND map appears in $DS.I$) **goto** 2;
   3. for each ($i : DS.I$)
      1. if (map doesn’t appear in $Q.I$) **goto** 2;
      2. else add a binding of $i$ with the single value of map
   3. for each (skeleton property $t$ appeared in $Q Sel$)
      1. if (scope($t$) is a single value span, AND map($t$) appears in $DS.I$) **goto** 2.4; 
      2.4.1 if (scope($t$) doesn’t intersect) **goto** 2;
      2.4.3 if (scope($t$) - scope(map($t$)) is empty) **goto** 2.4;
      2.4.4 if (map($t$) doesn’t appear in $DS.O$) **goto** 2;
      2.4.5 Add a filtering ValueScope vs on map($t$) into Plan;
2.5 stuffedFlag := false;
2.6 for each (o : Q.O)
2.6.1 if (map(o) appears in DS.O)
2.6.1.1 stuffedFlag := true;
2.6.1.2 Add a binding of o with map(o) into Plan;
2.6.2 else if (o appears also in Q.I) add a binding of o
2.6.3 else add a binding of o with empty value into Plan.
2.7 if(stuffedFlag == true) return TRUE with Plan.
3. return FALSE;

5 Implementation and Experiments

We have implemented a proof-of-concept version of USDIS.
Its source codes have been published to
http://usdis.googlecode.com. USDIS is built in Java 1.5 with
assistant of Eclipse. Axis 1.4 is used to invoke Web Services
and Jena 2.5.4 is used to perform SPARQL query. As to
OWL reasoner we employ the KAON2. In addition, we use
Graphl to display the Query Pattern Skeleton graph. At
present the user interface is based on user-friendly local GUI.
The query formulating interface is shown in Figure 2.

5.1 The dynamic Invocation Mechanism

As a featured part, our dynamic Web Service invocation
mechanism in Java is realized on Axis 1.4 and a tool
DyncClient which is developed by us by virtue of Java
reflection mechanism. DyncClient is able to call methods of a
class with default constructor in condition that only the
followings are provided: the class name, the method name,
<parameter element path, value string> pairs to specify input
values at leaf level, and the <parameter element path, String
object> pairs to accept outputs and the return. At runtime,
these data are derived from the execution plan and an XML-
based annotation file(See below).

When a Web Service is to be registered into USDIS, the
administrator processes it with 5 steps as shown in Figure 3:

5.2 Experiments

We have performed a data query that is answered by both a
demonstrative local RDF file and Amazon ECS(annotated as
Example 2). The query is as follows and the snapshot of the results is shown in Figure 5:

```
Qps: {
  Individuals: {_:b – bk:Book }
  Connections: {}  }
Sel: {<<_:b, bk:hasTopic >, ["Semantic Web"], <<_:b, bk:publishYear >, [2006, +∞)}}
O: {[_:b, bk:title], [_:b, bk:author], [_:b, bk:publishYear], [_:b, bk:amazonRank], [_:b, bk:descriptionURL]}
I: {[_:b, bk:hasTopic]}
C: {[bk:Book, bk:Goods]}
```

Although this query seems trivial, it definitely validates the capability of USDIS that integrating RDF datasets and real world Web Services uniformly.

6 Related Work

The problem of integrating Web Services with other types of data sources hasn’t been researched a lot yet. Information Manifold [5] and TSIMMIS [6] are data integration systems built in environment of relational databases and semi-constructed data. The technique of answering queries using views they used provides an important foundation for the virtual style of data integration. The ontology-based data integration systems [2] [1] are also emerging in recent Semantic Web environment. But they seldom consider the Web Service type of data sources. Among them we adopt the flavour of query formulating interface from [1].

Halpin etc. [3] puts forward a lightweight vocabulary “Semantic fXML” to unifying the diverse strands of XML-based Web technologies including Semantic Web and Web Service. As to how to represent the semantic of Web Services, it simply refers the I/O annotating style of previous Semantic Web Service technologies. Sbodio etc. [8] connects SPARQL to Web Services by using it to express the Precondition/Effect in OWL-S. It remains in the framework of OWL-S yet. So neither of them helps to treat Web Services as data sources.

7 Conclusions

We put forward a fine-granularity semantic description technology - Uniform Query - that describes information-providing Web Services as data views with access pattern limitation. Based on it we build a data integration system that uniformly integrating both RDF datasets and information-providing Web Services. Next, we plan to extend the system with the capability of answering a query by composing data from multiple data sources into a single record in results, as described in [4] for databases queries.

Acknowledgements

This work is supported by National Natural Science Foundation of China (60432010), National Basic Research Program of China (2007CB307103) and National Key Technology R&D Program of China (2006BAH02A11).

References

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