LDaaSWS: Toward Linked Data as a Semantic Web Service

Leandro José S. Andrade and Cássio V. S. Prazeres
Computer Science Department
Federal University of Bahia
Salvador, Bahia, Brazil
Email: {leandrojsa, prazeres}@dcc.ufba.br

***Abstract*—The Web was originally created to link HTML documents. Nowadays, the Web has improved its potential, and heterogeneous applications, resources, data and users can interact with each other. Two proposals for improvement of the current Web, Semantic Web and Web Services, have established standards that make the interoperability between heterogeneous Web applications possible. Another way to improve the current Web is the Web of Data, which provides guidelines (Linked Data) about how to use Semantic Web standards for publication and definition of semantic links on diverse data sources. However, there is a gap in the integration between Web Service based applications and Web of Data applications. Such a gap occurs because Web Services are "executed" and Web of Data applications are "queried". Therefore, this paper introduces the LDaaSWS (Linked Data as a Semantic Web Service), in order to provide Web Services for data sources from the Web of Data. The LDaaSWS can fulfill the current gap between Web Services and Web of Data applications by making the Web of Data "executed" through Web Services. In order to compare this work with current approaches for Web Services, this paper also presents an evaluation of LDaaSWS by comparing with SOAP Web Services.**

Keywords*–*Web of Data; Semantic Web Services; OWL-S; Linked Data.

I. Introduction

The initial purpose of the Web was to create hyperlinks between Hypertext Markup Language (HTML) documents [1]. From this initial purpose, the capability of the Web has been improved extensively, for example, now making user collaboration (Web 2.0) [2] and applications interoperability (Web API and Web Services) possible [3]. According to Martin et al. [4], standards should be developed for the Web and, furthermore, Web Services have to produce and consume data through a common protocol to make data interchange between heterogeneous applications possible.

In this case, standards are means to describe Web Services with languages such as Web Services Description Language (WSDL) and Web Application Description Language (WADL), which present service syntactical description of Simple Object Access Protocol (SOAP) services [5] and RESTful services [6], respectively. After Berners-Lee's [7] first article about the Semantic Web, several works have introduced different approaches for the semantic description of Web Services, in order to automate tasks, such as discovery, composition and invocation of Web Services [8] [9]. As a result, several approaches were proposed as standards for semantic description of Web Services, among which the Semantic Markup for Web Services (OWL-S) used in this work.

On the other hand, Berners-Lee [10] introduced a set of guidelines (Linked Data) to publish data on the Web. These rules indicate to use URI for identify resources, the HTTP protocol to access resources, Resource Description Framework (RDF) and SPARQL Query Language for RDF (SPARQL) for description, query, and hyperlinks to other resources. Such guidelines were inspired by a project to publish open data on the current Web: Linking Open Data [11]. Furthermore, the Web of Linked Data is being called the Web of Data [1].

Developers of applications from the current Web want to make their applications functionalities and/or data available to be accessed by other applications [12]. According to O'Reilly [2], there are several different data sources that demand applications through combining such data sources to offer composite services. These kinds of applications are known as mashups, which integrate Web resources to create new applications [13]. However, developing such mashups demands programming efforts for program developers, because they have to discover and compose the available Web resources [14] .

Therefore, there is a gap in the integration between these two Web evolution trends: Web Service based applications and Web of Data applications. Such a gap occurs because Web Services are "executed" through Hypertext Transfer Protocol (HTTP) requests, and Web of Data applications are "queried" through SPARQL queries. This issue is current and relevant, as several authors presented works [15] [16] [17] with approaches to address it. Some of these works introduce approaches to describe Web Services with Linked Data and others introduce approaches to produce Linked Data through Web Services or Web APIs. In order to overcome this gap, this paper introduces the Linked Data as a Semantic Web Service (LDaaSWS), in order to provide Semantic Web Services of data sources from the Web of Data. In summary, this paper proposes to make access on Linked Data sources through Web Services possible.

In order to establish the Linked Data cloud as a Web Service provider, LDaaSWS implements Semantic Web Services described with OWL-S. This approach enables the automatic integration of data from the Web of Data with others types of OWL-S based services (for instance, SOAP and RESTful). Furthermore, LDaaSWS also enables the automatic generation of Linked Data queries from service requests described on OWL-S.

Figure 1 presents the overall view of our proposal by explaining a possible scenario of a tourism application, which needs a Web Service that receives as input a city and gives as outputs some information about this city (latitude and longitude, hotels, description, population size and phone code). In this scenario, there is not a Web Service that fulfill the request, however some parts of the resquest can be attend. In Figure 1 shows two OWL-S/WSDL Web Services (Services 2 and 5 in Figure 1) that fulfill output about hotels and phone code of city. It has other Web Service (Service 1 in Figure 1), now a LDaaSWS one, that suplement information about latitude and longitude of the city. However, in this scenario there are not Web Services for outputs about description of city and population size (Services 3 and 4 in Figure 1). Then, we present an extension (Section III) to language OWL-S in order to allow the description of LDaaSWS proposed in this article, used in Service 1 in Figure 1.

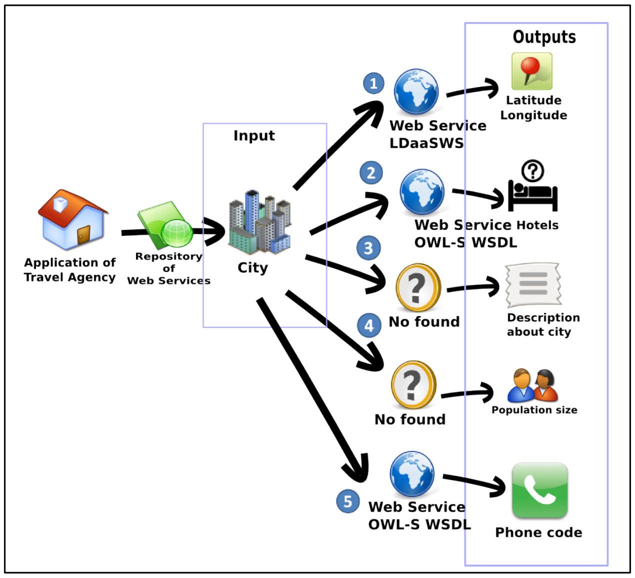

Figure 1. LDaaSWS overall view.

We developed a module for discovery of new LDaaSWS services to fulfill parts of services requests without a match (showed in Section IV). In Figure 1 the outputs 3 and 4 do not have services to attend, so we can use the module for discovery LDaaSWS to do it.

In this sense, this paper introduces the following results: i) OWL-S ontology extension to provide support for services derived from the Web of Data; ii) the usage of the OWL-S API [18], in order to enable the execution of the LDaaSWS in the same way as traditional (SOAP or RESTful) Web Services are executed; iii) automatic generation of Web of Data queries from OWL-S service requests, in order to enable the automatic requests and execution of LDaaSWS.

In this paper, Section II presents related works. Linked Data as a Semantic Web Service (proposal of this work) is described in Section III. Section IV introduces the automatic request and execution of LDaaSWS. Section V describes the results of the evaluation performed in our approach. Finally, Section VI presents the final remarks and recommendations for future works.

II. Related Work

The literature reports works that use the Web of Data as Web Services, or even use Linked Data for describing Semantic Web Services (SWS). According to Pedrinaci et al. [19], SWS and Web of Data together can solve some problems that limit the use of both. The combination of these two features, in addition, can increase use of SWS, as this will add to your field a growing multidisciplinary information base, serving as a complete and complementary method in the discovery and composition of Web Services.

Following a different line of research, but within the same context, Taheriyan et al. [16] identified the need for the composition of services from different sources to improve application development. Thus, the authors proposed an approach to integrate Web API's to a Linked Data cloud with the use of semantic description, using RDF and SPARQL. Norton and Stadtmller [20], [21] underscore the need for composing RESTful services by reducing the effort of the manual programming developer; it proposes a description of the services using Linked Data principles and semantically describing its inputs and outputs with SPARQL and RDF.

Paolucci et al. [22] propose the integration of SAWSDL (Semantic Annotations for WSDL) with Semantic Web Services described in OWL-S. The authors point out advantages in describing Semantic Web Services in OWL-S language and propose an extension of the ontology description of executions of OWL-S Web Services (Grounding) to support services with SAWSDL descriptions in OWL-S.

III. Linked Data as a Semantic Web Service

This paper proposes Linked Data as a Semantic Web Service (LDaaSWS), in order to provide Web Services from Linked Data sources. The motivation for this proposal mainly focuses on three points: i) the Web of Data is a database where access is restricted to SPARQL, which limits its potential queries, because it hinders integration with other data sources that have no Linked Data; ii) LDaaSWS makes the usage of the Linked Data cloud as a service provider possible; iii) LDaaSWS enables Linked Data to be integrated automatically with other Web Services supported by the language OWL-S (for instance, SOAP and RESTful Web Services), enabling interoperability of data and reducing the programming effort for service discovery and composition.

Thus, this paper presents SPARQLGrounding, which is an extension to the language OWL-S in order to allow the description of LDaaSWS. The OWL-S ontology is composed of three sub-ontologies: `Profile`, `Process` and `Grounding`. The first two are abstract, generic and can include any implementation of service (SOAP-WSDL, RESTful-WADL, etc.). The `Grounding` ontology had been defined to be the concrete part of the OWL-S. Whereas it does not define type of service implementation, the `Grounding` is responsible for describing how the service will be performed. OWL-S can and should be extended to any type of service through the extension of the `Grounding` ontology.

Thus, the `SparqlGrounding` ontology proposed in this work is an extension of the OWL-S `Grounding` used to allow the Web of Data be executed as a service, i.e., to actually implement execution of LDaaSWS proposed in this paper.

In this context, the extension proposed in this paper followed the following requirements: i) allow execution of SPARQL queries based services that enable the mapping of input and output of services to SPARQL triples; ii) be in line with other OWL-S ontologies, that is, its inputs and outputs are correctly associated with the elements of `Process` and `Profile` ontologies; iii) not to be dependent on anything other than document OWL-S and its sub-ontologies for a full description; iv) offer a semantic description, unambiguously, for automated processes of discovery, selection, composition and execution of services, performed by software agents.

The mapping of OWL-S `Grounding` for SPARQL extends the abstract layer composed by `Grounding` and `AtomicProcessGrounding` classes, both defined by OWL-S. Figure 2 shows a UML class diagram that displays such extension.

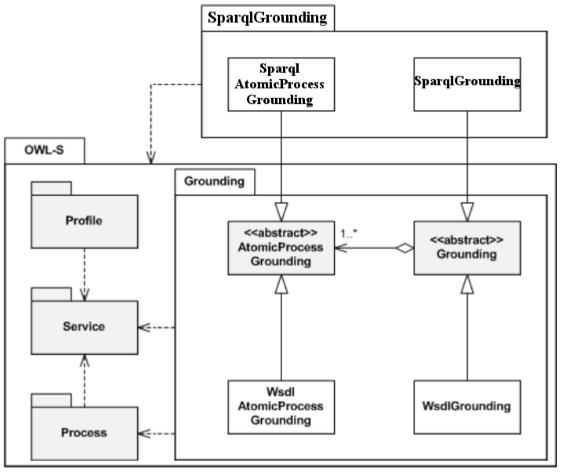

Figure 2. SparqlGrounding model.

Figure 2 illustrates the `Service`, `Profile` and `Process` ontologies which details were omitted to highlight the specialization of `Grounding` ontology. Classes `SparqlGrounding` and `SparqlAtomicProcessGrounding` in Figure 2 are not part of the OWL-S `Grounding` ontology. These classes are proposed in this paper with the aim of describing the execution of LDaaSWS.

The model presented in Figure 2 proposes grouping the new classes in a dedicated ontology, called `SparqlGrounding`. This approach avoids any change in the specification of OWL-S, which will facilitate the adoption of our new ontology, without interfering with existing services described in OWL-S prior to incorporation of `SparqlGrounding`.

In order to formalize `SparqlGrounding`, it is necessary to provide a description of all its elements following the syntax of the OWL language. Figure 3 describes the formalization of the `SparqlGrounding` class as a subclass of the `Grounding` class previously defined by OWL-S. Note also that the definition has restrictions related to the existence of `AtomicProcessGrounding` (line 5 of Figure 3) and that all such should be of type `SparqlAtomicProcessGrounding` (line 6 of Figure 3).

The `SparqlAtomicProcessGrounding` class, shown in Figure 4, is a subclass of `AtomicProcessGrounding` and has the restriction of a data property (`Datatype-Property`) called `sparqlEndPoint`, which stores the URI endpoint that a SPARQL query must be submitted on completion of the service. Other elements are associated with LDaaSWS belonging to this class.

```
1 <owl:Class rdf:ID=”SparqlGrounding”>
2   <rdfs:subClassOf rdf:resource=”&grounding;Grounding”/>
3   <rdfs:subClassOf>
4     <owl:Restriction>
5       <owl:onProperty rdf:resource=”&grounding;hasAtomicProcessGrounding”/>
6       <owl:allValuesFrom rdf:resource=”#SparqlAtomicProcessGrounding”/>
7     </owl:Restriction>
8   </rdfs:subClassOf>
9 </owl:Class>
```

Figure 3. SparqlGrounding Class.

```
1 <owl:Class rdf:ID=”SparqlAtomicProcessGrounding”>
2   <rdfs:subClassOf rdf:resource=”&grounding;AtomicProcessGrounding”/>
3   <rdfs:subClassOf>
4     <owl:Restriction>
5       <owl:onProperty rdf:resource=”#sparqlEndPoint”/>
6       <owl:cardinality rdf:datatype=”&xsd;nonNegativeInteger”>1</owl:cardinality>
7     </owl:Restriction>
8   </rdfs:subClassOf>
9 </owl:Class>
```

Figure 4. SparqlAtomicProcessGrounding Class.

A SPARQL query can have prefixes that are eventually used in triplets, so in `SparqlGrounding` a property of type *ObjectProperty* (lines 1 to 4 of Figure 5) is defined, followed by the OWL class `SparqlPrefixMap` (lines 6 to 19 of Figure 5), which defines the prefix name (`PrefixName`) and the associated URI (`PrefixUri`). Figure 5 presents an excerpt of this definition; note that `SparqlPrefixMap` defines the existence of only one elements `PrefixName` and `PrefixUri`, ensuring integrity for the formation of prefixes.

```
 1 <owl:ObjectProperty rdf:ID=”SparqlPrefixes”>
 2   <rdfs:domain rdf:resource=”#SparqlAtomicProcessGrounding”/>
 3   <rdfs:range rdf:resource=”#SparqlPrefixMap”/>
 4 </owl:ObjectProperty>
 5
 6 <owl:Class rdf:ID=”SparqlPrefixMap”>
 7   <rdfs:subClassOf>
 8     <owl:Restriction>
 9       <owl:onProperty rdf:resource=”#PrefixName”/>
10       <owl:cardinality rdf:datatype=”&xsd;nonNegativeInteger”>1</owl:cardinality>
11     </owl:Restriction>
12   </rdfs:subClassOf>
13   <rdfs:subClassOf>
14     <owl:Restriction>
15       <owl:onProperty rdf:resource=”#PrefixUri”/>
16       <owl:cardinality rdf:datatype=”&xsd;nonNegativeInteger”>1</owl:cardinality>
17     </owl:Restriction>
18   </rdfs:subClassOf>
19 </owl:Class>
```

Figure 5. SparqlPrefixes Property and SparqlPrefixMap Class.

Figure 6 (lines 1 to 4) shows the definition of input parameter of `SparqlGrounding`. The `SparqlIntputParamMap` class (lines 6 to 21 of Figure 6) is responsible for the mapping between elements of `Process` (`Input`) through of property `owlsParameter` (line 17 of Figure 6), and data associated with the input of the service. In WADL and WSDL groundings, the `owlsParameter` is used to connect the input ID of syntactic service document. However, in the context of LDaaSWS, it is used to indicate which variable in the query will be mapped with the input data.

Finally, Figure 7 describes the definition of triple belonging to the clause `WHERE` of SPARQL queries, mapping the subject, predicate and object of the triple, respectively represented by the properties `TripleSubject` (line 9 of Figure 7), `TriplePredicate` (line 15 of Figure 7) and `TripleObject` (line 21 of Figure 7). In

```
<owl:ObjectProperty rdf:ID="SparqlInputParam">
  <rdfs:domain rdf:resource="#SparqlAtomicProcessGrounding"/>
  <rdfs:range rdf:resource="#SparqlInputParamMap"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SparqlIntputParamMap">
  <rdfs:subClassOf rdf:resource="#SparqlDataParamMap"/>
  <rdfs:subClassOf rdf:resource="#InputMessageMap"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#SparqlDataParam"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&grounding;owlsParameter"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 6. OWL elements to describe service input.

`SparqlTripleMap` there is a restriction on the amount of elements that make up the triple to ensure its syntactic integrity.

```
<owl:ObjectProperty rdf:ID="SparqlTriples">
  <rdfs:domain rdf:resource="#SparqlAtomicProcessGrounding"/>
  <rdfs:range rdf:resource="#SparqlTripleMap"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SparqlTripleMap">
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#TripleSubject"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#TriplePredicate"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#TripleObject"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Figure 7. OWL elements to describe SPARQL triples.

IV. Automatic Request and Execution of LDaaSWS

LDaaSWS makes automatic service requests (Section IV-A) possible through automatic generation of SPARQL queries from OWL-S requests. Moreover, it is also possible to perform automatic execution (Section IV-B) of LDaaSWS from the queries generated. This section presents such two features (automatic request and execution) of LDaaSWS, which enable software agents to access services based on Linked Data automatically.

A. Automatic Request

In order to explore the Web of Data, it is necessary to use SPARQL, which makes the need to map the semantics described by the OWL-S language to a SPARQL query indispensable. Figure 8 presents an overview of the generation of the SPARQL queries module from service requests described in OWL-S: OWL-S to SPARQL. From the service request (part 1 of Figure 8), one SPARQL query of type *ASK* (returns *true* or *false*, respectively, exist or not one or more data according to the request). (part 2 of Figure 8) is developed, which is generated from the ontology of the inputs and outputs. We choose this type of query, because it has a lower cost of implementation, given that, at this stage, no one wants to return data but rather validate the query.

After the generation of an ASK query, the next step is the application of this query in an *endpoint* (part 3 of Figure 8) to check for data that matches the request. In the case of ASK query return positive, an OWL-S service can be created and can use this database for the answers (part 4 of Figure 8), which one can follow to implement the `SparqlGrounding` (described in Section III).

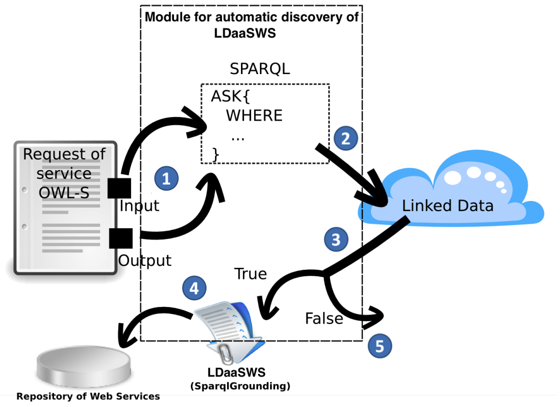
Module for automatic discovery of LDaaSWS
SPARQL
ASK{
WHERE
...
}
Request of service OWL-S
Input
Output
1
2
3
4
5
Linked Data
True
False
LDaaSWS (SparqlGrounding)
Repository of Web Services

Figure 8. OWL-S to SPARQL: Automatic Request.

There are some challenges associated with this mapping OWL-S to SPARQL shown in Figure 8. Initially, the semantic expressiveness of OWL-S language is greater than the expressiveness of SPARQL. This implies the need to identify what types of requests are enabled to perform the query expression. Therefore, for an experimental evaluation, we selected a profile of simple OWL-S request, which allows the generation of queries. Thus, requests for services where the input element is owned (is a property) by the output element (or otherwise) were selected. For example, a service request where the input is the latitude and longitude of a city and the output is an ontology class that represents city.

Figure 9 presents the description of a service request (part "a" of Figure 9), which has the ISBN of a book as input and a Book (a class) as output. The ISBN is part of the domain Book (part "b" of Figure 9), thus this type of service allows to generate a SPARQL query similar to the query displayed in part "c" of Figure 9.

We can map other types of queries through using techniques for similarity of ontologies, or even through inferences, which allow the resources of inputs to be related with the resources of outputs. We can also generate queries that partially meet the requests, where the generated service can later be combined with other services to meet the initial request. As a result, `SparqlGrounding` allows the execution (Section IV-B) of any service described with SPARQL query, in other words, it does not restrict the execution of services with queries automatically generated. Thus, if a developer designs a SPARQL query manually and wants to set it as an OWL-S service, the `SparqlGrounding` can be implemented for such a query. However, the mapping of queries is out of the scope of this paper.

B. Automatic Execution

Figure 10 illustrates an overview of the LDaaSWS execution. The starting point is the OWL-S request (step 1 in

```
A) Description of ProccessDescrição of OWL-S service
<process:Input rdf:ID="ISBN">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/
XMLSchema#anyURI">http://dbpedia.org/ontology/isbn</
process:parameterType>
<rdfs:label/>
</process:Input>

<process:Output rdf:ID="BOOK">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/
XMLSchema#anyURI">http://dbpedia.org/ontology/Book</
process:parameterType><rdfs:label/>
</process:Output>
B) RDF of ontology http://dbpedia.org/ontology/Book
<rdf:RDF>
 ...
  <rdf:Description rdf:about="http://dbpedia.org/ontology/isbn">
<rdfs:domain rdf:resource="http://dbpedia.org/ontology/Book" />
  </rdf:Description>
  <rdf:Description rdf:about="http://dbpedia.org/ontology/
numberOfPages">    <rdfs:domain rdf:resource="http://dbpedia.org/
ontology/Book" />  </rdf:Description>
...
</rdf:RDF>

C) SPARQL query resulted
ASK
WHERE{
?varbook rdf:type <http://dbpedia.org/ontology/Book>
?varbook <nhttp://dbpedia.org/ontology/isb> ?varisbn
}
```

Figure 9. Service request and SPARQL query.

Figure 10) and from analysis of the ontologies that describe the inputs and outputs, a SPARQL query equivalent to the request is developed (OWL-S to SPARQL module in Figure 10 corresponds to Figure 8). At this stage, not all service requests produce SPARQL queries, since not all of them refer to information available on the Web of Data (this issue is treated in Section IV-A). After that, the resulting query is validated in the Linked Data cloud (step 2 in Figure 10), at which time whether or not there is data available to fulfill this request is reviewed.

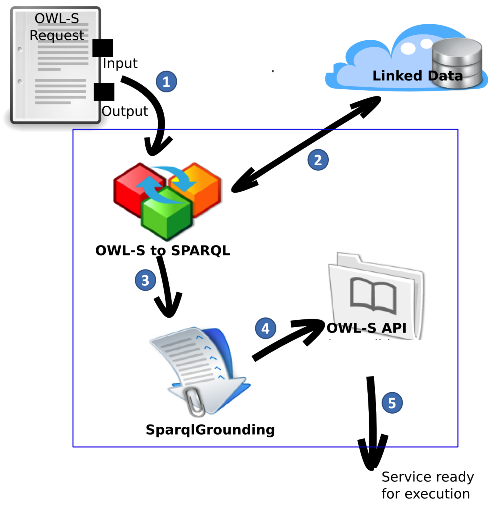

Figure 10. LDaaSWS Automatic Execution.

In step 3 in Figure 10, if there is information in the Web of Data for the OWL-S request, it has been a Grounding OWL-S (described in Section III), which describes how to run a service. Finally, the resulting `Grounding` is sent to the OWL-S API (step 4 in Figure 10) to be executed (step 5 in Figure 10).

Importantly, once a whole process to request execution of LDaaSWS is held (steps 1-5 of Figure 10), this procedure should not be repeated for subsequent requests, because the new discovered service may be stored in a database of services that can be accessed in the future, reducing the time for discovery and generation of service.

Figure 11 shows a snippet of a code to perform a service that uses the `SparqlGrounding`. Initially, the service is loaded into knowledge base (line 3 of Figure 11) to allow access to the `Process`, which, consequently, indicates the data input (lines 5 to 7 Figure 11). Finally, in line 12 of Figure 11, the service is performed, which, from `Process`, is called the class `SparqlGroundingProvider`, which starts the whole foundation class `SparqlGrounding` implemented by the service.

```
1  // loading of ontologies
2  URI uri = new URI("http://localhost/services/isbn_book_sparql_grounding.owls");
3  Service service = kb.readService(uri);
4
5  Process process = service.getProcess();
6  // Creating input parameters
7  ValueMap<Input, OWLValue> inputs = new ValueMap<Input, OWLValue>();
8  inputs.setValue(process.getInput("ISBN"), kb.createDataValue("0-375-50137-1"));
9
10 // Creating output parameters and executing the service
11 ValueMap<Output, OWLValue> outputs = new ValueMap<Output, OWLValue>();
12 outputs = exec.execute(process, inputs, kb);
```

Figure 11. Code snippet to perform a service.

V. LDaaSWS Evaluation

In the LDaaSWS evaluation, we used an Intel Core i5 computer with four cores of 1.80GHz, 6GB of RAM memory, with operation system Debian/Linux 8.0, Java environment with J2SE 1.7 and Eclipse 3.8.1. Additionally, we used the Apache Web Service for access and storage of Web Services and the DBPedia [23] for access to a Linked Data base. The services used to execute the tests were extracted from the *OWL-S Test Collection* [24] package, which have been adapted to use DBPedia ontology and the updated version 1.2 of OWL-S.

Experiments have been performed to evaluate the performance (execution time) of our proposal. Toward greater consistency of results, for each evaluation 30 tests were executed and the execution time in each test was observed. As described in Sections III and IV, the development of LDaaSWS has three important contributions, which were separately analyzed for better evaluation. Therefore, Section V-A shows the evaluation of the `SparqlGrounding` ontology; Section V-B presents the evaluation of automatic generation of SPARQL queries – request and execution.

A. SparqlGrounding ontology

In the ontology evaluation, the OWL-S API was used. Indicating the correct functioning of `SparqlGrounding` ontology is necessary in order to run a Semantic Web Service using `SparqlGrounding`; in other words, it must use the OWL-S API with support for LDaaSWS for execution of a Web Service.

Therefore, in order to evaluate the performance of `SparqlGrounding`, the execution time was fully measured in the following points: i) reading of ontologies before starting the execution Process; ii) mapping and reading of `Grounding` classes; and iii) preparation time for Semantic Web Service execution. This paper did not deal with the Web Service execution, because it was beyond the scope of contributions and its execution time is generally associated with the performance of the service itself.

Because the `WSDLGrouding` is a built-in grounding of the OWL-S API, we related measurements taken with `SparqlGrounding` in comparison to equivalent measurements of `WSDLGrouding`. Figures 12, 13 and 14 show the results of measurements of `SparqlGrounding` in comparison with `WSDLGrouding`. We can see that there are no substantial differences in performance; however, `SparqlGrounding` has better performance because in the query preparation it does not need access to a syntactic document, which reduces the execution time of `SparqlGrounding`. As a result, through an analysis of the graphs, it appears that the `SparqlGrounding` gives satisfactory performance with the OWL-S API.

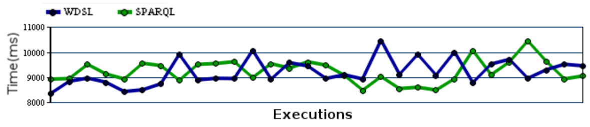

Figure 12. Mapping and reading classes WSDLGrounding and SPARQLGrounding

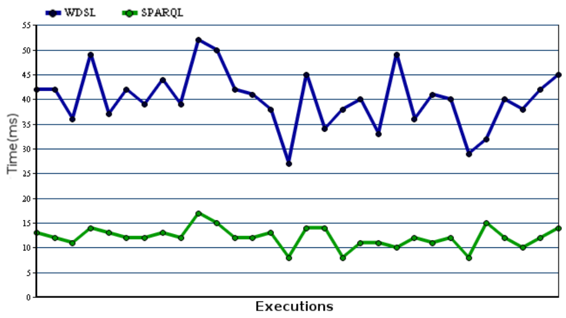

Figure 13. Reading ontologies related to WSDLGrounding and SPARQLGrounding

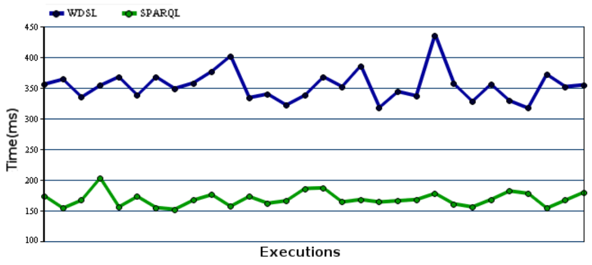

Figure 14. Preparation time for execution of the service with WSDLGrounding and SPARQLGrounding

B. Request and Execution

In order to evaluate the automatic generation of SPARQL queries, we applied a scenario where the main module for SPARQL queries generation was subjected to executions using OWL-S service requests. Figures 15 and 16 show a piece of the code of service requests highlighting elements of input (lines 1 to 6 of Documents 15 and 16) and output (lines 8 to 13 of Figure 15 and lines 8 to 20 of Figure 16) of the `Process`, which are the main features for the generation of SPARQL queries.

```
 1 <process:Input rdf:ID="_ISBN">
 2   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
 3      http://dbpedia.org/ontology/isbn
 4   </process:parameterType>
 5   <rdfs:label>isbn</rdfs:label>
 6 </process:Input>
 7
 8 <process:Output  rdf:ID="_BOOK">
 9   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
10      http://dbpedia.org/ontology/Book
11   </process:parameterType>
12   <rdfs:label>book</rdfs:label>
13 </process:Output>
```

Figure 15. Part of the service request ISBN-BOOK

```
 1 <process:Input rdf:ID="_CITY">
 2   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
 3      http://dbpedia.org/ontology/City
 4   </process:parameterType>
 5   <rdfs:label></rdfs:label>
 6 </process:Input>
 7
 8 <process:Output  rdf:ID="_LAT">
 9   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
10      http://www.w3.org/2003/01/geo/wgs84_pos#lat
11   </process:parameterType>
12   <rdfs:label></rdfs:label>
13 </process:Output>
14
15 <process:Output  rdf:ID="_LON">
16   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
17      http://www.w3.org/2003/01/geo/wgs84_pos#long
18   </process:parameterType>
19   <rdfs:label></rdfs:label>
20 </process:Output>
```

Figure 16. Part of the service request City-Latitude/Longitude

We split the evaluation into three parts, for better measurement of the performance of the SPARQL query generator module: i) time of reading of ontologies associated with the request OWL-S service; ii) the building time of the SPARQL query; and iii) query execution time (DBPedia endpoint[25]).

Figures 17, 18 and 19 show graphs with execution time measurements for OWL-S service requests of Figures 15 and 16, noting the time spent in three situations: reading of the OWL-S service request (Figure 17), creation of the SPARQL query (Figure 18) and execution of the SPARQL query created (Figure 19). It is important to note that over 50% of total execution cost is associated with the reading of ontologies, a point that is not connected with the implemented solution, but rather the access of resources of the OWL-S service request.

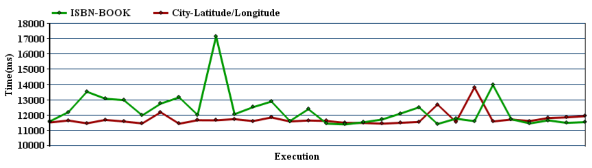

Figure 17. Time of reading of ontologies

Thus, the results reported for measuring execution time of the creation and execution of SPARQL queries show that total time for automatic generating SPARQL queries was about 5 seconds. This time is quite acceptable considering that the developer of a Web Service would not need to develop a query manually and this process will not be repeated in cases of OWL-S services requests already converted into SPARQL queries.

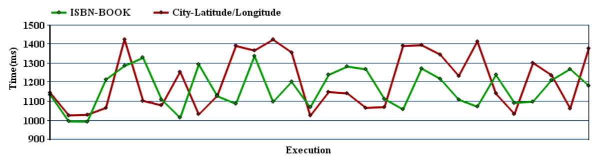

Figure 18. Time of creation of SPARQL query

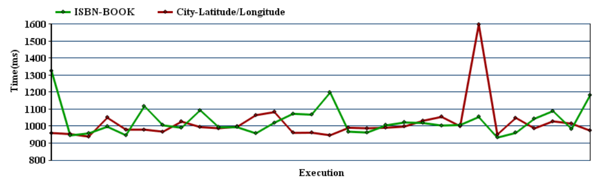

Figure 19. Time of execution SPARQL query

VI. Final Remarks

The evolution of the Web presents a scenario with data coming from various sources and applications that can provide its functionality by merging information from different sources. This trend motivates researches efforts for the development of techniques that create environments and techniques for automatic discovery of data and services in the Web.

Therefore, this paper aims at providing Semantic Web Services using semantic descriptions with OWL-S language from Linked Data: the LDaaSWS. This proposal presents important contributions to the area of Semantic Web Services, especially regarding the discovery, because LDaaSWS allow automatic generation of Web Services from the Linked Data cloud. Furthermore, it enables the development of more elaborate applications, which require less expertise of developers and enable more integration and reuse of data from the Web of Data.

Improvements that can be made from the reported work include: i) implementation of other mappings of complex OWL-S requests to automatically generated SPARQL queries; ii) conducting experiments to evaluate the composition of LDaaSWS with other already established Web Services types (such as SOAP and RESTful), based on approaches to Semantic Web Service composition reported in literature [26].

References

[1] C. Bizer, “The emerging web of linked data,” IEEE Intelligent Systems, vol. 24, no. 5, Sep. 2009, pp. 87–92.

[2] T. OŔeilly, “What is web 2.0. design patterns and business models for the next generation of software,” Communications and Strategies, September 2005.

[3] C.-C. Tsai, C.-J. Lee, and S.-M. Tang, “The web 2.0 movement: mashups driven and web services,” W. Trans. on Comp., vol. 8, no. 8, aug 2009, pp. 1235–1244.

[4] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan, “Bringing semantics to web services with owl-s,” World Wide Web, vol. 10, no. 3, Sep. 2007, pp. 243–277.

[5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, “Simple Object Access Protocol (SOAP) 1.1,” World Wide Web Consortium, W3C Note, 2000.

[6] L. Richardson and S. Ruby, Restful web services, 1st ed. OŔeilly, 2007.

[7] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” Scientific American, vol. 284, no. 5, May 2001, pp. 34–43.

[8] S. McIlraith, T. C. Son, and H. Zeng, “Semantic web services,” Intelligent Systems, IEEE, vol. 16, no. 2, 2001, pp. 46–53.

[9] P. Larvet, B. Christophe, and A. Pastor, “Semantization of legacy web services: From wsdl to sawsdl,” in Internet and Web Applications and Services, 2008. ICIW ’08. Third International Conference on, June 2008, pp. 130–135.

[10] T. Berners-Lee, “Linked data - design issues,” W3C, no. 09/20, 2006. [Online]. Available: http://www.w3.org/DesignIssues/LinkedData.html

[11] C. Bizer, T. Heath, D. Ayers, and Y. Raimond, “Interlinking open data on the web,” www4.wiwiss.fu-berlin.de/bizer/pub/LinkingOpenData.pdf, 2007, stand 12.5.2009. [Online]. Available: www4.wiwiss.fu-berlin.de/bizer/pub/LinkingOpenData.pdf

[12] D. Benslimane, S. Dustdar, and A. Sheth, “Services mashups: The new generation of web applications,” Internet Computing, IEEE, vol. 12, no. 5, 2008, pp. 13–15.

[13] S. Makki and J. Sangtani, “Data mashups & their applications in enterprises,” in Internet and Web Applications and Services, 2008. ICIW ’08. Third International Conference on, June 2008, pp. 445–450.

[14] G. Di Lorenzo, H. Hacid, H.-y. Paik, and B. Benatallah, “Data integration in mashups,” SIGMOD Rec., vol. 38, no. 1, jun 2009, pp. 59–66.

[15] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, “Baya: assisted mashup development as a service,” in Proceedings of the 21st international conference companion on World Wide Web, ser. WWW ’12 Companion. New York, NY, USA: ACM, 2012, pp. 409–412.

[16] M. Taheriyan, C. A. Knoblock, P. Szekely, and J. L. Ambite, “Rapidly integrating services into the linked data cloud,” in Proceedings of the 11th international conference on The Semantic Web - Volume Part I, ser. ISWC’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 559–574.

[17] S. Stadtmüller and B. Norton, “Scalable discovery of linked apis,” Int. J. Metadata Semant. Ontologies, vol. 8, no. 2, Sep. 2013, pp. 95–105.

[18] M. D. Evren Sirin and T. Mller. Owl-s api. Available on the internet at http://on.cs.unibas.ch/. Last access in 21 October 2014. [Online]. Available: http://on.cs.unibas.ch/ (2012)

[19] C. Pedrinaci, J. Domingue, and R. Krummenacher, “Services and the web of data: An unexploited symbiosis.” in AAAI Spring Symposium: Linked Data Meets Artificial Intelligence. AAAI, 2010.

[20] B. Norton and S. Stadtmüller, “Scalable discovery of linked services,” in Proceedings of the Fourth International Workshop on Resource Discovery, vol. 737, RED Workshop. Heraklion, Greece: CEUR-WS, Mai 2011.

[21] S. Stadtmuller, “Composition of linked data-based restful services,” in Proceedings of the 11th international conference on The Semantic Web, ser. ISWC’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 461–464.

[22] M. Paolucci, M. Wagner, and D. Martin, “Grounding owl-s in sawsdl,” in Service-Oriented Computing - ICSOC 2007, ser. LNCS, B. Kramer, K.-J. Lin, and P. Narasimhan, Eds. Springer Berlin Heidelberg, 2007, vol. 4749, pp. 416–421.

[23] Dbpedia. Available on the internet at http://www.dbpedia.org. Last access in 24 December 2014. [Online]. Available: http://www.dbpedia.org (2014)

[24] M. Klusch and P. Kapahnke. Owls-tc is a owl-s service retrieval test collection to support the evaluation of the performance of owl-s semantic web service matchmaking algorithms. [Online]. Available: http://projects.semwebcentral.org/projects/owls-tc/ (2010)

[25] Sparql endpoint dbpedia. Available on the internet at http://www.dbpedia.org/sparql. Last access in 27 October 2014. [Online]. Available: http://www.dbpedia.org/sparql (2014)

[26] T. Weise, S. Bleul, D. Comes, and K. Geihs, “Different approaches to semantic web service composition,” in Internet and Web Applications and Services, 2008. ICIW ’08. Third International Conference on, June 2008, pp. 90–96.