MAPPING XML TO EXISTING OWL ONTOLOGIES

Toni Rodrigues, Pedro Rosa, Jorge Cardoso
Department of Mathematics and Engineering
University of Madeira
9050-390 Funchal, Portugal
toni@apus.uma.pt, pcosta@apus.uma.pt, jcardoso@uma.pt

ABSTRACT

Now-a-days, XML has reached a wide recognition and brought interoperability at a syntactic level. Unfortunately, even when using XML to represent data, problems arise when it is necessary to integrate different data sources because XML lacks support for efficient sharing of conceptualization. Emerging Semantic Web technologies, such as ontologies, can enable semantic interoperability. With ontologies, it is possible to formally represent shared domain knowledge models defined with concepts, attributes, relationships and instances. In this paper, we present a notation to map XML Schema to existing OWL ontologies and the qualities an algorithm should have to transform XML documents (instances of the mapped schema) into instances of the mapped ontology.

KEYWORDS

Semantic Web, Ontology Mapping, Ontology Generation, XML, OWL.

1. INTRODUCTION

XML brought syntactic interoperability and became the de facto standard as a B2B data exchange format [1]. However, XML "lacks semantics" [2, 3]. Thus problems arise when it is necessary to manipulate and integrate different XML data sources. Consequently, today's organizations are again shifting (or it is expected them to do so) from a syntactic interoperability level to a semantic one [4]. Emerging Semantic Web technologies, such as ontologies, can play an important role in this scenario [3, 5]. Ontologies are a formal and explicit specification of a shared conceptualization [6] can enable semantic interoperability. They are supported by the W3C through their Web Ontology Language (OWL) [7] Recommendation. To assist the migration from XML to OWL, it is necessary to develop tools supporting mappings and transformation between these two standards. The goal presented in this paper is to narrow the gap between XML and OWL proposing a strategy to map XML Schema to existing OWL ontologies and transform XML data (instances of the mapped XML Schema) into instances of the ontology according to the performed mapping.

Several works have been done in this area. Most of them intend to automatically transform XML Schema into newly created ontologies capturing the implicit semantics existing in the structure of XML documents. Such approach is used in [8]. They describe automatic mappings from XML to RDF as well as from XML Schema to OWL. However, since the mappings are independents, the generated instances may not respect the OWL model created from the XML Schema. In [9] the authors cope with this independency between the mappings. Their implemented framework creates a new ontology from an XML Schema and transforms instances of the XML Schema into instances of the created ontology.

Another interesting and more complete approach is the XML2OWL framework [10]. It is developed in XSLT and transforms XML Schema into an OWL ontology. Additionally, it also supports instances transformation like the previous described work. This framework goes even further since it is able to generate an ontology from an XML instances document if no schema is available. This framework resembles the one we have developed but there are several differences. In fact, this tool creates a new ontology from an XML schema during which the user has no control on the process. That is, the user has no control on the newly created ontology which, similarly to XSD2OWL framework discussed earlier, captures the implicit semantics existent in the XML Schema structure. Therefore the created ontologies are quite primitive, that is, they are not really semantically richer than the mapped XML Schemas. Our main objective is different. Our approach allows the mapping of XML schema to an existing ontology and appropriately generates rules that automatically transform instances of the XML schema to instances of the mapped ontology. Usually, the mapped ontology is semantically richer than the mapped XML Schema since the ontology can be created independently of the mapped XML data sources.

Tools supporting mapping from XML Schemas to an existing OWL ontology are very scarce. COMA++ is a schema and ontology matching tool [11] providing several automatic matching algorithm. COMA++ can establish mappings from XML Schema to OWL but it does not intend to map them with the purpose of facilitating the transformation of schema's instances into individuals as we do.

Most of the related works (actually all the one found during our research) does not support mappings and instances transformation to an existing OWL ontology. These facts make our research and implemented framework a unique contribution to the pool of semantic Web applications. This paper presents a complete approach to manually map XML Schemas to existing OWL ontologies with the purpose of automatically transform instances of the mapped schema into instances of the ontology. We backup our work presenting a scenario solved with the implemented and ready-to-use JXML2OWL framework.

2. MAPPING XML TO OWL

This section presents a notation to specify mappings between XML and OWL. This notation support several kinds of mappings: one-to-one, one-to-many, many-to-one and many-to-many. Therefore, the presented notation allows mappings from one XML node to several OWL concepts and mappings from several XML nodes to one OWL concept. This section also presents several aspects that must be taken into account when writing an algorithm to perform the instances transformation according to the created mappings.

2.1. XML to OWL Mapping: The Notation

This section defines a notation, represented in Table 1, to specify mappings between elements of an XML Schema and resources defined by an OWL ontology. Just like OWL ontologies, which are mainly defined by classes, datatype and object properties, we classify mappings in three distinct types:

- Class mapping: Maps an XML node to an OWL concept
- Datatype property mapping: Maps an XML node to an OWL datatype property
- Object property mapping: Relates two class mappings to an OWL object property

Table 1. Mapping Notation

Mappings	Notation

Class	`(OWL Class URI, XPath expression)`
	`(OWL Class URI, XPath expression, ID XPath expression)`

Datatype Property	`(OWL Datatype Property URI, Domain Class Mapping, XPath Expression)`
Object Property	`(OWL Object Property URI, Domain Class Mapping, Range Class Mapping)`

OWL resources (classes, object and datatype properties) are addressed using their URI references [12] while XPath [13] expressions are used to address the mapped XML nodes. The use of XPath expressions allows distinguishing several XML nodes with the same name but with different ancestors and permits to map them to their corresponding OWL concepts. It is also possible to use XPath predicates to enables conditional mappings.

An examination of Table 1 reveals that class mappings are defined by pairs containing the URI reference of the mapped OWL class as well as an XPath expression identifying the mapped XML nodes. Such pair means that an instance of the mapped OWL class is created for each XML nodes matching the XPath expression. As an alternative, it is also possible to create class mappings with triplets where the XML node used the compute the IDs of the generated instances are directly specified. This second possibility is discussed in more details in the next section 2.2 of this paper.

Additionally, Table 1 reveals that not only property mappings are specified with triplets but also that class mappings are used to define property mappings. Such solution enables a complete support of property mappings in the context of many-to-many mappings. An example of many-to-many mappings is explored as a scenario in section 4.

The following example shows how class, datatype and object property mappings are created. Let us consider an ontology with two OWL classes, `tourism:Country` and `tourism:City`, which are respectively the domain and range of the object property `tourism:hasCity`. Let us also consider its inverse property `tourism:belongsToCountry`, the `xsd:string` datatype properties `tourism:city_name` and `tourism:country_name`, as well as the following XML document:

```
<locations>
  <location>
    <country name="Portugal"/>
    <city name="Funchal"/>
  </location>
  <location>
    <country name="France"/>
    <city name="Paris"/>
  </location>
</locations>
```

Using the notation from Table 1, the following mappings are valid:

- `cm1 = (tourism:Country, /locations/location/country)`
- `cm2 = (tourism:City, /locations/location/city)`
- `op1 = (tourism:hasCity, cm1, cm2)`
- `op2 = (tourism:belongsToCountry, cm2, cm1)`
- `dp3 = (tourism:country_name, cm1, /locations/location/country/@name)`
- `dp4 = (tourism:city_name, cm1, /locations/location/city/@name)`

cm1 mapping means that an instance of `Country` class is created for each XML node matching the specified XPath expression. Therefore, two `Country` individuals are created, one for Portugal and one for France. Similarly, two `City` individuals are created. `op1` mapping means that each OWL instance created from the class mapping `cm1` is the domain of an object property `tourism:hasCity` whose range is an individual generated from the class mapping `cm2`. To obtain the exact individual used as range for a given individual, it is necessary to compute the relative path from the XPath expression used in the class mapping to the one used in the object property mapping, which is in the `op1` example `../city`. `dp3` means that for each instance created from the `cm1` class mapping, a datatype property tourism:`country_name` is also created. Again, it is also necessary to compute the relative path to find the value used to fill the property, which is for `dp3` mapping `@name`. Four relationships between individuals (corresponding to the two object property mappings) are also created: two that relates Portugal and Funchal, and other two that relates France and Paris. Finally, four datatype properties are also created with the names of the cities and countries.

2.2. XML to OWL Mapping: Instances Transformation

Instances of OWL classes are characterized by having unique identifiers. When creating the OWL instances document, it must be ensured that unique identifiers are generated for each individual. Another important task is to detect duplicate instances on the XML document. In fact, several XML nodes identified by different or even by the same XPath expressions may refer to the same individual. Based on their unique identifier, duplicate instances (so instances with the same ID) must be detected and filtered so that only one instance is created. By default, the ID of the instances are generated by sequentially concatenating the underscore symbol '_' with the prefix of the mapped class, with its local name and with the string-value [13] of the mapped XML node. In section 2.1 we referred to an alternative notation to specify class mappings where the XML node used to compute the IDs of the generated instances is directly specified. This notation with triplets allows the detection of duplicated instances even if the default string-value is different because the comparison is performed according to the XPath expression specified in the class mapping.

The OWL recommendation also places several restrictions on properties. The most important one when generating the properties of individuals is that OWL does not allow the assignment of duplicates to property values. As such, it is necessary to filter and eliminate duplicates when creating both the OWL instances and the properties. But this is not enough since it is also necessary to perform the union of all the distinct property values mapped as can see in the following example. Putting all this together, let us consider the following XML document as well as an ontology with the `univ:student` OWL class which is the domain of the `univ:email` datatype property.

```
<univ>
  <students>
    <student>
      <name age="25">Toni Rodrigues</name>
      <email>rodrig.toni@gmail.com</email>
    </student>
    <student>
      <name age="24">Pedro Rosa</name>
      <email>pedro1215@gmail.com</email>
    </student>
  </students>
  <professors>
    <professor>
      <name>Jorge Cardoso</name>
      <students>
        <student name="Toni Rodrigues">
          <email>rodrig.toni@gmail.com</email>
          <email>a2008899@max.uma.pt</email>
        </student>
        <student name="Marco Sousa"/>
      </students>
     </professor>
  </professors>
</univ>
```

The following mappings are valid:

- `cm1 = (univ:student, /univ/students/student, /univ/students/student/name)`
- `cm2 = (univ:student, /univ/professors/professor/students/student,`
  `/univ/professors/professor/students/student/@name)`
- `pm1 = (univ:email, cm1, /univ/students/student/email)`
- `pm2 = (univ:email, cm2, /univ/professors/professor/students/student/email)`

Three `student` instances are created with the followings IDs: `_univToniRodrigues`, `_univPedroRosa` and `_univMarcoSousa`. One should note that it may be necessary to encode the generated IDs into NCNames [14] to ensure their validity. For the student Toni Rodrigues, the duplicate instance is discarded, that is, only one instance is created since the generated IDs are the same. However, two `univ:email` properties must be created, one for rodrig.toni@gmail.com and one for a2008899@max.uma.pt because they are distinct. One of the rodrig.toni@gmail.com emails was discarded because it is a duplicate one. We can state that for each generated individual, it is necessary to perform the union of all the properties related to this individual, to remove the duplicates and finally to create the remaining properties. This process must be done to both datatype and object properties.

OWL recommendation also supports the definition of several restrictions such as maximal and minimal cardinality restrictions of properties over classes. Maximal cardinality restrictions can easily be supported since it is just a matter of ensuring that the maximum number of allowed properties is not exceeded, discarding the remaining ones. A complete support of minimal cardinality restriction is impossible, that is, it is impossible to guarantee that this kind of restrictions is always satisfied. However, there are two distinct cases that must be supported. For each case, appropriate warnings and comments need to be generated. The first case happens when properties on which minimal cardinality restrictions exist are not mapped. Warnings and comments are generated both on the transformation rules and on the OWL instances document. The second one happens when XML instances document does not contain enough instances to satisfy the minimal cardinality restriction. Since this case can only be evaluated at run-time, comments can only be generated in the OWL instances document.

3. IMPLEMENTED FRAMEWORK: JXML2OWL

JXML2OWL is a framework divided in two sub projects: JXML2OWL API and JXML2OWL Mapper. The API is a generic and reusable open source library for mapping XML schemas to OWL ontologies for the Java platform while the Mapper is an application with a graphical user interface (GUI) developed in Java Swing that uses the API and eases the mapping process. JXML2OWL supports manual mappings from XML, XSD or DTD documents to an OWL ontology adhering to the notation presented in Table 1, thus supporting all the kinds of mappings such as many-to-many. Currently, conditional mappings through XPath predicates are not implemented within the framework. According to the mapping performed, JXML2OWL generates mapping rules wrapped in an XSL document that allows the automatic transformation of any XML data, that is, any XML document validating against the mapped schema, into instances of the mapped ontology. Figure 1 represents such process. The algorithm defined by the generated XSL document satisfies all the aspect discussed in section 2.2.

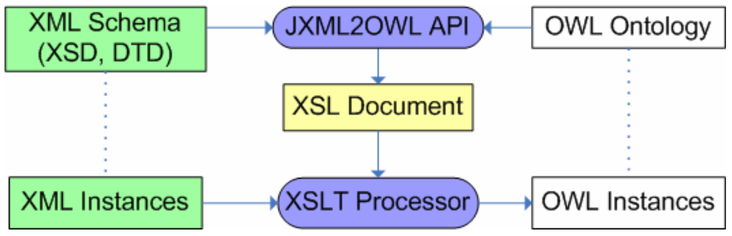


Figure 1. JXML2OWL supports mappings and instances transformation

With JXML2OWL, the mapping process requires several steps. The first step consists in creating a new mapping project and loading both the XML Schema related file (XSD or DTD) and the OWL ontology. If an XML schema in not available, it is possible to load an XML document. In this case, JXML2OWL extracts a possible schema. In the second step, the user creates class mapping between elements of the loaded XML schema and classes of the ontology. Once these mappings are created, it is possible to relate them to each other to create object property mappings, or to relate them with elements of the XML schema to create datatype property mappings. Finally, in the last step, it is possible to export the transformation rules, generated according to the mapping performed, as an XSL document. With this XSL document it is possible to transform any XML document which validates against the mapped XML schema into individuals of the mapped OWL ontology. Obviously, both the API and the Mapper support all these steps.

4. MAPPING SCENARIO

JXML2OWL was developed during the year 2005/2006. It has been successfully employed in the context of a major project called SEED (SEmantic E-tourism Dynamic packaging - http://seed.expedita.com.pt) whose purpose is to integrate disparate and heterogeneous e-tourism data sources into a unique knowledge base. Let's consider a use case where an XML Schema is mapped to an existent ontology. One should note that in this scenario there is not a direct correspondence between the mapped XML elements and the ontology. This short but rather complex scenario was purposely chosen to enact how it is possible to play with all the possible combination offered with many-to-many mappings. All the mappings are performed using the notation defined in Table 1. According to the mappings, the XSL Transformation is performed to generate to individuals of the ontology. Figure 2 graphically represents the mapped ontology with several classes. Each class has several object properties represented by links in Figure 2 and datatype properties represented in plain text within the class container.

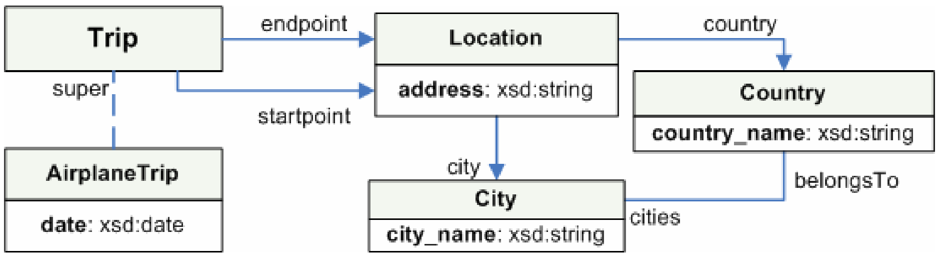


Figure 2. Mapped Ontology

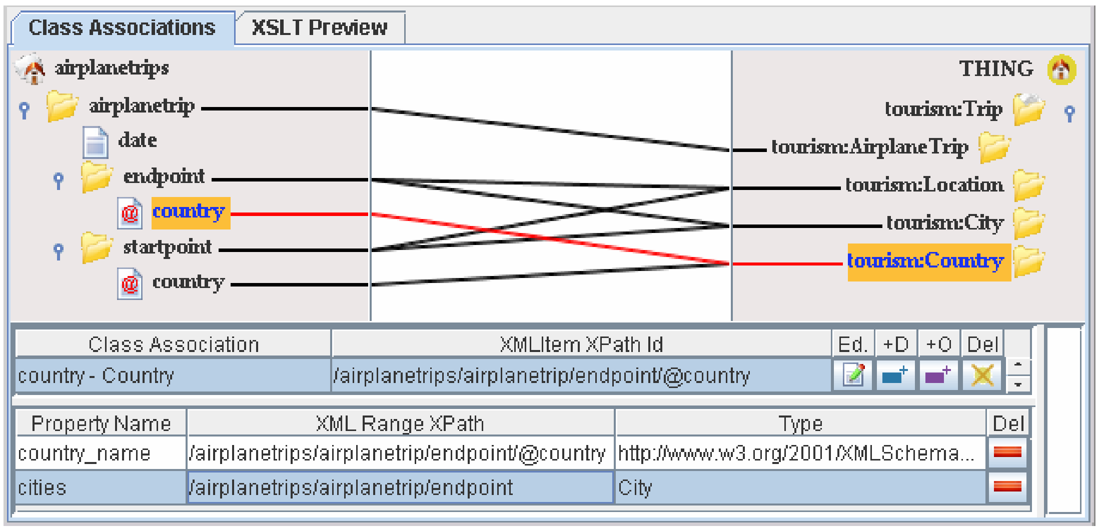
This is an XML instances document of the mapped schema:

```
<airplaneTrips>
  <airplaneTrip>
    <date>12-06-2006</date>
    <startpoint country="Portugal">Lisbon</startpoint>
    <endpoint country="Portugal">Funchal</endpoint>
  </airplaneTrip>
  <airplaneTrip>
    <date>25-06-2006</date>
    <startpoint country="Portugal">Funchal</startpoint>
    <endpoint country="Portugal">Lisbon</endpoint>
  </airplaneTrip>
</airplaneTrips>
```

And these are the created mappings according to the notation presented in Table 1:

- `cm1 = (tourism:AirplaneTrip, /airplaneTrips/airplaneTrip)`
- `cm2 = (tourism:City, /airplaneTrips/airplaneTrip/startpoint)`
- `cm3 = (tourism:City, /airplaneTrips/airplaneTrip/endpoint)`
- `cm4 = (tourism:Location, /airplaneTrips/airplaneTrip/startpoint)`
- `cm5 = (tourism:Location, /airplaneTrips/airplaneTrip/endpoint)`
- `cm6 = (tourism:Country, /airplaneTrips/airplaneTrip/startpoint/@country)`
- `cm7 = (tourism:Country, /airplaneTrips/airplaneTrip/endpoint/@country)`
- `pm1 = (tourism:date, cm1, /airplaneTrips/airplaneTrip/date)`
- `pm2 = (tourism:startpoint, cm1, cm4)`
- `pm3 = (tourism:endpoint, cm1, cm5)`
- `pm4 = (tourism:city, cm4, cm2)`
- `pm5 = (tourism:city, cm5, cm3)`
- `pm6 = (tourism:cities, cm6, cm2)`
- `pm7 = (tourism:cities, cm7, cm3)`
- `pm8 = (tourism:belongsTo, cm2, cm6)`
- `pm9 = (tourism:belongsTo, cm3, cm7)`
- `pm10= (tourism:country, cm4, cm6)`
- `pm11= (tourism:country, cm5, cm7)`
- `pm12= (tourism:city_name, cm2, /airplaneTrips/airplaneTrip/startpoint)`
- `pm13= (tourism:city_name, cm2, /airplaneTrips/airplaneTrip/endpoint)`
- `pm14= (tourism:country_name, cm6, /airplaneTrips/airplaneTrip/ /startpoint/@country)`
- `pm15= (tourism:country_name, cm7, /airplaneTrips/airplaneTrip/ /endpoint/@country)`

Figure 3 shows the state of the JXML2OWL Mapper for the given scenario and mappings. On the left side, the XML schema is represented, while on the right side the OWL classes defined by the ontology are shown. In-between we can see the mapping zone. It is possible to drag-and-drop elements from the left to the right (and vice-versa) to create mappings. By selecting a created mapping, it is possible to create datatype and object property mappings. Under the mapping zone, the XML node used as ID for the select class mapping is displayed as well as all the datatype and object property mappings created. and related to the selected class mapping.



Class Association	XMLItem XPath Id	Ed.	+D	+O	Del
country - Country	/airplanetrips/airplanetrip/endpoint/@country				

Property Name	XML Range XPath	Type	Del
country_name	/airplanetrips/airplanetrip/endpoint/@country	http://www.w3.org/2001/XMLSchema...	🟥
cities	/airplanetrips/airplanetrip/endpoint	City	🟥

Figure 3. Mapper state with some mappings

Running the XSLT Transformation over the XML instances document, produces an OWL output document containing all the generated individuals and their properties. The OWL instances document imports the mapped ontology to which the prefix `tourism` is bound. The individuals are generated satisfying all the aspects discussed in section 2.2. For instance, duplicate instances and properties were discarded.

```
<owl:Ontology rdf:about="">
  <owl:imports  rdf:resource="  http://jxml2owl.projects.semwebcentral.org/tou-
          rism.owl"/>
</owl:Ontology>
<tourism:AirplaneTrip rdf:ID="_tourismAirplaneTrip2006-06-12LisbonFunchal">
  <tourism:startpoint rdf:resource="#_tourismLocationLisbon"/>
  <tourism:endpoint rdf:resource="#_tourismLocationFunchal"/>
  <tourism:date rdf:datatype="xsd:date">2006-06-12</tourism:date>
</tourism:AirplaneTrip>
<tourism:AirplaneTrip rdf:ID="_tourismAirplaneTrip2006-06-25FunchalLisbon">
  <tourism:startpoint rdf:resource="#_tourismLocationFunchal"/>
  <tourism:endpoint rdf:resource="#_tourismLocationLisbon"/>
  <tourism:date rdf:datatype="xsd:date">2006-06-25</tourism:date>
</tourism:AirplaneTrip>
<tourism:Location rdf:ID="_tourismLocationLisbon">
  <tourism:city rdf:resource="#_tourismCityLisbon"/>
  <tourism:country rdf:resource="#_tourismCountryPortugal"/>
</tourism:Location>
<tourism:Location rdf:ID="_tourismLocationFunchal">
  <tourism:city rdf:resource="#_tourismCityFunchal"/>
  <tourism:country rdf:resource="#_tourismCountryPortugal"/>
</tourism:Location>
<tourism:City rdf:ID="_tourismCityLisbon">
  <tourism:belongsTo rdf:resource="#_tourismCountryPortugal"/>
  <tourism:city_name rdf:datatype="xsd:string">Lisbon</tourism:city_name>
</tourism:City>
<tourism:City rdf:ID="_tourismCityFunchal">
  <tourism:belongsTo rdf:resource="#_tourismCountryPortugal"/>
  <tourism:city_name rdf:datatype="xsd:string">Funchal</tourism:city_name>
</tourism:City>
<tourism:Country rdf:ID="_tourismCountryPortugal">
  <tourism:cities rdf:resource="#_tourismCityLisbon"/>
  <tourism:cities rdf:resource="#_tourismCityFunchal"/>
  <tourism:country_name rdf:datatype="xsd:string">Portugal
          </tourism:country_name>
</tourism:Country>
```

The produced OWL instances document can obviously be loaded in any OWL editor such as Protégé-OWL.

5. CONCLUSION

We presented an approach and implemented the JXML2OWL framework to manually map XML Schema documents to existing OWL ontologies and automatically transform XML instances documents into individuals of the mapped ontology. Such framework is crucial for organizations that plan to move from a syntactic representation of data using XML to a semantic one using OWL.

The conducted scenario was deliberately chosen to enact the power of the defined notation and transformation algorithm, which are particularly suited for the expressiveness of OWL language. Examining the created individuals, one can see how the syntactic XML document gets a lot of semantics when transformed into individuals. This is mainly due to the fact that XML is mapped to an existent ontology which is much richer than the ontologies generated by other tools [8, 9, 10] which capture the implicit semantics available in the structure of XML documents.

We believe the presented framework is appropriate to integrate any XML data into semantic information systems based on OWL ontologies when conditional mappings, which will be the focus of our subsequent work, are not required. JXML2OWL has been successfully employed to integrate disparate e-tourism data sources in XML format as individuals of an e-tourism OWL ontology and is available for download at http://jxml2owl.projects.semwebcentral.org. We hope the research done to bridge the gap between XML and OWL as well as the implemented prototype demonstrated the need for semantic mapping tools and will stimulate software companies, mainly the ones developing mapping applications, to develop professionals mapping tools supporting mappings and instances transformation to existing OWL ontologies.

ACKNOWLEDGEMENT

This work was funded by grants from the FCT (Fundação para a Ciência e a Tecnologia) and carried out in cooperation with Expedita.

REFERENCES

[1] Bussler , C., 2003. *B2B Integration, Concepts and Architecture*. Springer, Germany.

[2] Hawke, S., 2001. *XML with Relational Semantics: Bridging the Gap to RDF and the Semantic Web*. W3C, http://www.w3.org/2001/05/xmlrs/.

[3] Shabo, A. et al, 2006. Revolutionary impact of XML on biomedical information interoperability. *IBM Systems Journal,* Vol. 45, No. 2, pp. 361-372.

[4] EBizQ, 2005. *Semantic Integration: A New Approach to an Old Problem*. EBizQ.

[5] Alexiev, V. et al, 2005. *Information Integration with Ontologies*. John Wiley & Sons, New Jersey, USA.

[6] Gruber, T., 1993. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, Vol. 5, No. 2, pp. 199-220.

[7] Bechhofer, S. et al, 2004. *Web Ontology Language (OWL) Reference version 1.0*. W3C, http://www.w3.org/TR/owl-ref/.

[8] Ferdinand, M. et al, 2004. Lifting XML Schema to OWL. *Web Engineering - 4th International Conference*, ICWE, Munich, Germany, pp. 354-358.

[9] Garcia, R. et al, 2006. Ontological Infrastructure for a Semantic Newspaper. *Semantic Web Annotations for Multimedia Workshop*, SWAMM'06, Edinburghm, UK.

[10] Bohring, H., Auer, S., 2005. Mapping XML to OWL Ontologies. *Marktplatz Internet: Von e-Learning bis e-Payment.* Leipziger Informatik-Tage (LIT2005), Leipzig, Germany, pp.147-156.

[11] Aumueller, D et al. Schema and Ontology Matching with COMA++. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. Baltimore, USA, pp. 906-908.

[12] Burners-Lee et al, 1993. *Naming and Addressing: URIs, URLs, ...*. W3C, http://www.w3.org/Addressing/.

[13] Berglund, A. et al, 1999. *XML Path Language (XPath) Version 1.0*. W3C, http://www.w3.org/TR/xpath.

[14] Bray, T. et al, 2006. *Namespaces in XML 1.0 (Second Edition).* W3C, http://www.w3.org/TR/REC-xml-names/#NT-NCName.