Moving from Syntactic to Semantic Organizations using JXML2OWL

Toni Rodrigues [1], Pedro Rosa [2], Jorge Cardoso [3] (Contact Author)

[1] *SQLI*
Immeuble le Pressensé
268 Avenue du Président Wilson
93210 La Plaine Saint-Denis
France
trodrigues@sqli.com

[2] *Department of Mathematics and Engineering*
University of Madeira
9050-390 Funchal
Portugal
pcosta@apus.uma.pt

[3] *SAP Research CEC Dresden – SAP AG*
Chemnitzer Strasse 48
01187 Dresden, Germany
T +49 351 4811-6145
F +49 6227 78 50340
jorge.cardoso@sap.com

Abstract. Today's enterprises face critical needs in integrating disparate information spread over several data sources inside and even outside the organization. Most organizations already rely on XML standard to define their data models. Unfortunately, even when using XML to represent data, problems arise when it is necessary to integrate different data sources. Emerging Semantic Web technologies, such as ontologies, RDF, RDFS, and OWL, can play an important role in the semantic definition and integration of data. The purpose of our study is to present a framework to assist organizations to move from a syntactic data infrastructure defined in XML to a semantic data infrastructure using OWL. The framework supports mappings and fully automated instance transformation from syntactic data sources in XML format to a common shared global model defined by an ontology using Semantic Web technologies. The presented framework, JXML2OWL, allows organizations to automatically convert their XML data sources to a semantic model defined in OWL.

Keywords: semantic web, ontologies, information systems integration, mapping, transformation

1 Introduction

The Semantic Web is a project and a vision of the World Wide Web Consortium (W3C). It is an extension of the current Web in which "information is given a well-defined meaning, better enabling computers and people to work in cooperation" [1]. While the current Web is only human-understandable, the Semantic Web vision intends to represent Web content in such a form that it becomes machine-processable [2]. Toward this objective, several standards have emerged under the initiative of the W3C, such as RDF, RDFS and OWL. Ontologies play an important role to realize this vision allowing data to be defined and linked in a way that it enables its use for more effective discovery, integration, re-use across various applications and machine processing [1].

According to TopQuadrant, a consulting firm that specializes in Semantic Web technologies, the market for semantic technologies will grow at an annual growth rate of between 60% and 70% until 2010. It will grow from its current size of US$2 billion to US$63 billion [3]. Semantic Web technologies find one of their first commercial users in organizations facing data integration needs [4] and always seeking for better data integration solutions. Company mergers, integration of new software together with legacy systems which need to share data, the necessity of a unique global view of all the internal enterprise and external partner data sources, the need to be compliant with emerging standards to enable and maintain B2B cooperation, are all forces driving the need for data integration [5]. According to the InfoWorld's 2002 Application Integration Survey of IT leaders, integration costs consumed at that time an average of 24 percent of the yearly IT budget [6]. For midsize to large companies this represents millions of dollars. The year 2005 also was a busy year for data integration and nowadays companies are increasing their budget to better address data integration needs and related difficulties [7].

Integrating data from various data sources is not an easy task. In fact, several obstacles, mainly related with semantic heterogeneity, have been identified by researchers, such as [8]:

- Syntactic obstacles: Different terminology can be used to refer to semantically identical concepts. For example, a data source may use a table named *client* while another data source uses a table named *customer* with the same meaning.
- Semantic obstacles: The semantics may differ for similar terms. For example, the term *customer* can have different meanings. In one data source, it can include only end-customers while in another data source it can combine end-customers with dealers.
- Structural obstacles: The information may not only be structured differently but may also use distinct data formats. For instance, one data source may be available as a database using the relational model while another data source is provided using the XML format.

Current data integration approaches heavily rely on knowing what data is where and on the meaning of the data. Data description, or metadata, is essential to ease data integration and discovery [9]. Enterprise metadata repositories based on standards can be used as platforms for storing, accessing and managing metadata, as well as to locate information across an organization [10]. Meta-data also allows efficient re-use of integration efforts [10] and, in combination with semantic, it is possible to describe

contextually relevant or domain-specific information about content based on a domain metadata model [11]. This is what semantic metadata is.

With semantic metadata, a rich semantic domain model with concepts, attributes and relationships can be built. Ontologies constitute a good candidate to represent this kind of domain model. An Ontology is a formal, explicit specification of a shared conceptualization [12]. Thus, ontologies are particularly suitable to play the role of a central model. In fact, the conceptualization would be an abstract model of all the enterprise domain concepts. These domain concepts are explicitly defined and related to other concepts independently of the underlying applications. An ontology is not only human understandable. Indeed, because an ontology is a formal specification, it makes it also machine-processable. This formal specification also enables inference, that is, logical reasoning about concepts. Consequently, it is possible to derive (potentially new) knowledge from previously known facts. Ontologies, and therefore information, can be shared between applications or business partners because they are essentially domain specific. Products, such as Oracle 10g and Cerebra Server, which use ontologies for metadata description, global domain model specification, or more generally for data integration, are already available on the market.

The S2S (Syntactic-to-Semantic) middleware follows such a paradigm [13]. S2S uses ontologies to provide a semantic layer and transparently integrates disparate data assets of an organization hiding several details of the integrated data sources [14] . Such hidden details include the distributed nature of enterprise data sources and their structure, and semantic as well as syntactical heterogeneity. Figure 1 represents at a very high level the architecture of the S2S middleware. One can notice the use of an ontology, which provides a shared common understanding of a domain and enables semantic data integration, and other important modules such as the Extractor, Mapper and Instance Generator.

A typical scenario with this middleware essentially takes place as follows. When applications perform queries, the extractor module is responsible to transparently extract from the disparate assets the data needed to answer the query. This extracted data is then automatically transformed at run-time into instances of the ontology (usually called individuals) by the Instances Generator component according to the mapping previously performed by the Mapper module between the heterogeneous data sources schema and the ontology. Finally, inference can then be carried out over the knowledge base (ontology and its populated individuals). According to the business rules and query performed, appropriate results are returned to the applications. As enacted in this scenario, besides the extractors, the Mapper and the Instance Generator are key elements of such middleware, and more generally, of any system supporting semantic data integration. This approach allows organizations to view their heterogeneous data sources as one global ontology and brings all the discussed advantages such as: the ability to discover new knowledge from known facts; the capacity to share the global model between partners; and the capability to annotate data with metadata which can be used to ease data integration and discovery.

Within the scope of this paper, we are particularly interested in the Mapper and Instance Generator modules. We propose a semantic approach to cope with the data integration problems defining the JXML2OWL framework which can be used to map syntactic data in XML format to an ontology defined in OWL (Web Ontology Language). This paper is organized as follows. Section 2 presents the JXML2OWL project and then compares it to other related works. In Section 3, we propose a notation to specify mappings between XML schema and OWL ontology and discuss important

aspects regarding the instances generation. We call instances generation to the transformation process of XML data (validating against the mapped XML schema) into instances of the mapped ontology. Section 4 introduces the prototype we successfully implemented. Finally, Section 5 presents our conclusions.

2 A Brief Overview of JXML2OWL

Java XML to OWL project is divided into 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.

The final objective of JXML2OWL project is to develop a user-friendly, interactive and manual mapping tool that allows a user to map syntactic data in XML format to any existing ontology defined in OWL language with the purpose of easing and automating the semantic data integration process. More precisely, the developed mapping tool supports mappings between any XML schema (XSD and DTD) to concepts (classes and properties) of any OWL ontology. According to the mapping performed, the tool generates mapping rules as 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. XML documents were chosen as input because today's most commercial and scientific applications (such MS Excel, Apple iTunes or Matlab) as well as databases (such as MS SQL Server) provide services for automatically exporting their data or results into XML format. Additionally, XML has become the de facto standard for B2B data exchange and cooperation [15]. Thus we argue that XML can be considered as a standard representation of a wide variety of data sources. As such, the third obstacle, the structure problem, mentioned in Section 1 of this paper is partially addressed. Mapping to an ontology, which represents a shared common understanding of a specific domain, solves the terminology incompatibility problem. OWL was chosen as the ontology language because it is the W3C recommendation for building ontologies. Generated mapping rules are wrapped in an XSL document to easily support instances transformation. XSLT is the used standard to transform XML documents. The XSL document generated by JXML2OWL can be used by any XSLT processor to automatically transform instances of the mapped schema into instances of the ontology. The XSLT choice gets even more obvious considering that OWL is specified with XML syntax. This option towards XSLT standard allows us to simultaneously address the Mapper and Instance Generator modules specified in the S2S middleware architecture. As illustrated in Figure 2, the JXML2OWL API represents the Mapper component while the XSLT processor represents the Instance Generator module.

In the data integration context, the JXML2OWL API can be used ahead of query time to manually map each XML data source schema to the ontology and to generate the mapping rules wrapped in an XSL document. At run-time, that is, at query-time, data is fetched from the XML data sources and is automatically transformed into individuals using an XSLT processor and the generated rules over the fetched data.

2.1 Related work

Since Semantic Web technologies, and more precisely the OWL recommendation, are emerging concepts, not many applications using these technologies have been developed in the industry. Besides OWL, researchers are also developing specification to represent rules semantically [16, 17]. While specifications already exist for a few years, there is a lack of prototypes that can demonstrate without a doubt that the semantic Web is a solution for many of the problems that the industry faces. Moreover, many vendors seem to be taking a "wait-and-see" approach while the emerging standards converge. This position has already been discussed in [18] with respect to the Semantic Web and Web services.

Nevertheless, due to the development pace of Semantic Web technologies, a fair amount of applications that use OWL are available and result from academic research, projects and theses. These OWL applications can be grouped into three main categories:

- Editors/Browsers – Applications that allow to create, edit, and browse OWL ontologies
- Annotation tools – Applications that enable end users to annotate existent data with semantic context
- Mapping tools – Tools that enable the creation of correspondences/ mappings between two schemas. According to this mapping, instances of source schema can be transformed into instances of target schema.

A large part of the applications related to the OWL W3C recommendation are included in either Editor/Browser or Annotation categories. Mapping tools that involve XML and OWL data models are scarce. In fact, during our research, we did not find any tool supporting mappings from XML schemas to existing OWL ontologies. This fact makes our project JXML2OWL a unique contribution to the pool of semantic Web applications.

A considerable amount of mapping tools has the purpose of creating mappings between two distinct ontologies such as FOAM (Framework for Ontology Alignment and Mapping) [19]. Other works, which in some way are related with JXML2OWL, include COMA++, XML2OWL and Lifting XML Schema to OWL. These projects are discussed and compared to JXML2OWL in the next paragraphs.

COMA++ is a schema and ontology matching tool [20] developed at the University of Leipzig. This tool mainly supports XML schema and OWL ontology documents as data sources and enables a user to identify semantic correspondences between XML schemas, OWL ontologies or even between an XML schema and an OWL ontology. When mapping between XML schema elements and OWL concepts, it creates correspondences between them. These correspondences attach meaning to syntactic data and are expressed with simple pairs (XML schema element, OWL concept). The main objective of COMA++ is to provide several automatic matching algorithms. For example, it is possible to compare source elements and target schema taxonomies running an algorithm which suggests mappings that the user can validate, edit or discard. This ability of COMA++ is really interesting since it attenuates the problems related with manual mapping processes, which have the advantage of being accurate (because the mapping is performed by a human) but also have the inconvenience of being time-consuming, tedious and error-prone (for the same reason). Although COMA++ is part of the mapping group, it does not really intend to map XML schemas to ontologies with

the purpose of facilitating the transformation of schema's instances into individuals as we do. Also, the resulting mapping is quite primitive, with lot of semantic loss (such as relations between mapped concepts).

The major part of the other work done in this field intends to transform an XML Schema to a new ontology capturing the implicit semantics available in the structure of XML documents. Such an approach is used in [21]. The authors propose an approach to narrow the gap between XML, RDF and OWL. Their work is split into two independent parts: they describe automatic mappings from XML to RDF as well as from XML schema to OWL. However, since the mappings are independent, the generated instances may not respect the OWL model created from the XML schema. In [22] the authors cope with this independency between the mappings. Their framework automatically creates a partial mapping from an XML schema to an ontology using an XSLT transformation. It basically converts an XML Schema into a newly created ontology that captures the implicit semantics existent in the XML schema structure. It also transforms XML instances documents into instances of the newly created ontology. These two approaches are distinct from ours since both create a new ontology while JXML2OWL maps XML schemas to an already existent ontology.

Another interesting, very similar and complete approach is the XML2OWL framework. It is developed in XSLT and also transforms XML schema (XSD) into a newly created ontology in OWL [23]. Additionally, an XSLT that transforms instances of the XML schema into instances of the created ontology is also generated. This framework is similar with, but more complete than the works discussed in the previous paragraph since the generated instances respect the created OWL model and also support the creation of the OWL model directly from XML instances even if no XML schema is available. This project 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 over the process. That is, the user has no control over the newly created ontology which captures the implicit semantics existent in the XML schema structure. Our main objective is different. Our project, JXML2OWL, allows a user to map XML schema to an existing ontology, which is usually richer than the one created by XML2OWL framework, and appropriately generate an XSLT that automatically transforms instances of the schema to instances of the mapped ontology. In JXML2OWL, during the mapping, the user has an active role and controls the process. As such, we argue that XML2OWL is not a mapping tool. Instead, it is more appropriate to call it a converter or a transformer. In addition, the instances generation is quite primitive because duplicate instances are created with distinct IDs while JXML2OWL detects and filters duplicate instances, merging all the associated properties as explained in Section 3.

In this Section, all the projects that we have described are in some way related to our JXML2OWL project. However, none of them supports mappings and instances transformation to an existing OWL ontology. This means that, in the context of data integration, they do not support the incremental addition and mapping of new data sources. This was the main reason that led us to the specification and development of the JXML2OWL application.

3 XML to OWL Mapping: Specification and Instances Transformation

This section defines a notation to specify mappings between an XML schema to an OWL schema ontology. Important aspects of instances transformation are also discussed. When appropriate, several approaches are examined and then we indicate the chosen one. One should note that a good knowledge of several W3C recommendations, such as XML [24], OWL [25], and XPath [26], is necessary to better understand the concepts discussed.

3.1 The structures of XML and OWL

In order to fully understand the transformation process of instances between XML and OWL schema we have to understand the differences of these two data models. XML's data model [24] describes a node labeled tree (independently of using XML Schema or DTD to define the model), while OWL's data model is based upon the subject-predicate-object triples from RDF [27]. RDF schema defines a vocabulary for creating class hierarchies, attaching properties to classes and adding instance data.

Since the main characteristic of XML Schema and DTDs is to define a tree structure for the data, the transformation of instances from one data model to the other consists to simply map the XML tree structure to a class hierarchy. It should be noticed that transforming XML to OWL is a simpler task than the inverse mapping, in other words, mapping OWL to XML. This is because the elements and the expressiveness of XML are a subset of the elements and the expressiveness of OWL. Therefore, when creating mappings between elements of an XML schema and an OWL schema, we need to consider, in the one hand, the tree structure of XML, and, in the other hand, the class structure of an OWL ontology.

An XML DTD only provides basic cardinality constraints such as the Kleene operators ? (0 or 1), * (0+), and + (1+). A DTD also allows defining enumerations. Besides these basic cardinality constraints, the XML Schema also allows the specification of data types. Since OWL allows specifying cardinality constraints, enumeration, data types (OWL uses the same data types as XML Schema), it becomes straightforward to map cardinality constraints and enumeration from a DTD/XML Schema to OWL. It should be noticed that since the XML Schema and OWL use the same data types, when establishing mappings there is not need to perform any conversion or transformation. OWL only needs to reference the data types that were referenced by the XML Schema.

The nodes of the tree structure can be easily identified and referenced using an XPath expression. Since all the nodes have the same syntactic representation, no more considerations need to be drawn with respect to XML. Dealing with OWL is more involved, since depending on the semantics of an XML node it can be mapped to different OWL elements. Having a particular XML node, we need to consider three possible mapping scenarios to OWL:

- Map a XML node to an OWL concept;
- Map a XML node to an OWL datatype property
- Relate a XML node to an OWL object property

XML element	OWL element
Node	Class
Node	Datatype Property
Node	Object Property

Table 1. Elements that need to be considered when mapping XML to OWL

The data model mapping decisions that need to be taken into account when transforming XML to OWL are illustrated in Table [1]. This table shows that the mappings are established between XML elements (i.e., nodes) and OWL elements. The challenge is to formally specify under which conditions a XML node need to be mapped to an OWL class, datatype or object property. Our solution uses XPath expressions to distinguish XML nodes with the same name but with different ancestors and permits to map them to their corresponding OWL elements. Our approach references XML nodes with XPath expressions while OWL classes are referenced with their URIs. The pair (OWL class URI, XPath expression) identifies a mapping and means that an instance of the OWL class identified by the URI reference is created for each XML node matching the specified XPath expression. The following subsections discuss how the issue of establishing mappings between XML nodes and OWL elements has been addressed.

3.2 Referencing XML nodes

To map an XML node (i.e., an XML element or an attribute) to an OWL class, it is necessary to reference the XML node to be mapped. The first possible approach would be to reference the node by its name. However we cannot forget that XML lacks semantics. This means that XML nodes with the same name but with different parents may have different semantics. The following example, an XML document describing electronic products and technology, represents such a case.

```
<products>
   <electronics>
       <product>SONY LCD 28TV </product>
       <product>Philips Flat 32AB</product>
       <product>...</product>
   </electronics>
   <computers>
       <product>HP 720.us</product>
       <product>Dell P4-DC2 </product>
       <product>...</product>
   </computers>
</products>
```

The product elements have different semantics. For instance, the product elements with the names SONY LCD 28TV and Philips Flat 32AB are part of the electronics section while the product HP and Dell refer to computers. It is possible that those two kinds of products are represented by different concepts on the ontology. Referencing product nodes by their names simultaneously identifies electronics and computers and thus they would be mapped to the same ontological concept. Therefore, referencing XML nodes by their names is not a suitable solution.

A second approach, much more appropriate, is to identify the XML nodes with an XPath expression. For instance, the XML nodes representing electronics are referenced

with `/products/electronics/product` and can be mapped to the appropriate concept defined by the ontology. Similarly, computers are addressed by `/products/computers/product` and mapped to the corresponding ontological concept.

XPath expressions have other advantages: XML attributes can be easily addressed prefixing the attribute name with the '@' symbol. For instance, `/products/computers/product/@price` could address the price of a computer. Also, XPath predicates could be used to support conditional mappings.

Since this second approach using XPath expressions is more suitable, it was the selected to overcome the lack of semantics of XML documents.

3.3 Referencing OWL resources

OWL resources are the classes and properties defined by an ontology. The W3C OWL recommendation requires OWL resources to have unique identifiers. URI references are used as unique identifiers in the Semantic Web context to reference resources. URI references can be broken up in a namespace and a local name (or in a URI and a fragment). The namespace is usually the URI of the whole ontology. The local name uniquely identifies a resource within a namespace, that is, within an ontology. A prefix can be associated with a namespace and can then be used to reference a resource without writing the complete URI.

For example, `http://jxml2owl.sourceforge.net/ontologies/tourism.owl#` is a namespace (and a URI) that identifies the tourism ontology [28]. The concept `Accommodation` is the local name (or fragment) of a resource defined by this ontology. `http://jxml2owl.sourceforge.net/ontologies/tourism.owl#Accommodation` is the complete URI reference of this resource. The term `tourism:Accommodation` is equivalent to the previous URI reference if the prefix `tourism` is associated with the namespace identifying the ontology.

Since our objective is to map an XML schema to an ontology and knowing that a local name is unique within an ontology, one could think that the local name is appropriate to uniquely reference a resource defined by the mapped ontology. However, the local name is not enough because the OWL recommendation permits an ontology to import other ontologies. Therefore, to be able to address resources defined by the imported ontology, resources must be referenced by their URI. However, because URIs can be quite long, it is also possible to reference resources using a prefix and a local name.

3.4 Mapping XML nodes to OWL classes

XML nodes are referenced with XPath expressions while OWL classes are referenced with their URIs. The pair `(OWL class URI, XPath expression)` identifies a mapping and means that an instance of the OWL class identified by the URI reference is created for each XML node matching the specified XPath expression. Let us consider an ontology which defines two classes: `computerProduct` and `electronicProduct`. The term `product` is a prefix associated with the

namespace of this ontology. Let us also consider the XML document introduced in Section 3.1.

The following pair (`product:eletronicsProduct`, `/products/computers/product`) indicates that an instance of `electronicsProduct` is created for each XML nodes matching the XPath expression `/products/computers/product`. Therefore, applying the mapping rules to the considered XML document will generate two instances of the class `electronicProduct`, one instance for product SONY LCD 28TV and the other one for the product Philips Flat 32AB.

3.5 Mapping XML nodes to OWL properties

The W3C OWL recommendation defines two kinds of OWL properties: datatype and object properties. Both properties have a domain and a range. The domain of a property is not always a single class. For instance, it is possible to define the domain of a property as the union of several classes. Consider the property `displaySize` and the classes `LCD-TV` and `Plasma-TV`. Both a LCD television and a Plasma television have a display size. Therefore, the domain of the property `displaySize` can be specified as the union of the classes `LCD-TV` and `Plasma-TV`.

The range of a property varies according to the type of the property. Datatype properties are properties for which the value is a data literal, such as `xs:integer` (where `xs` is a prefix associated to the namespace `http://www.w3.org/2001/XMLSchema`), while object properties take individuals of a particular class as range.

3.5.1 Mapping XML nodes to OWL datatype properties

To create a datatype property mapping, the property as well as both its domain and its range must be specified. The OWL datatype property, which is an OWL resource, is addressed as we saw in Section 3.2 by its URI reference or by its prefix and local name. The value of a datatype property range is a data literal such as `xs:integer` or `xs:string`. Such a value can be specified with an XPath expression to indicate the XML element, attribute or node containing the value used to fill the property value.

A discussion can arise to determine the best way to specify the domain of a mapped property. How should it be referenced? Is it necessary to specify the domain to map a property? In order to answer these questions, let us consider the following case. Let us consider an ontology, identified by the prefix `product`, defining the concept `computerProduct` as an OWL class which is the domain of two datatype properties: `name`, whose range is a data literal `xs:string`, and `price`, whose range is `xs:integer`. Let us consider as well the following XML document.

```
<products>
   <price value="2500">
      <computer>
         <product>
             <name>Philips Flat 32AB</name>
         </product>
      </computer>
   </price>
   <price value="2400">
      <computer>
         <product>
```

```
                    <name>SONY LCD 28TV</name>
                </product>
            </computer>
        </price>
    </products>
```

Using the notation introduced in Section 3.3, the following mapping to an OWL class is valid:

```
(product:computerProduct, /products/price/computer/product)
```

Given this scenario, the challenge is to discover a generic way to specify that we intend to map the XML element `name` and the attribute `@value` to the OWL datatype properties `product:name` and `product:price` and specify the domain of these properties, which is `product:computerProduct`.

A pair such as `(product:name, /products/price/computer/product/name)` would mean that the value under `/products/price/computer/product/name` would be used as the range of the OWL datatype property `product:name`. This approach is not suitable because the domain is not specified. One could argue that the domain of the created property could be easily found checking the OWL class to which the parent node of `/products/price/computer/product/name` is mapped, namely `product:computerProduct`. However, this is not a valid solution. In fact, consider that we also want to map `/products/price/@value` to the datatype property `product:price`. Looking at the parent node is not appropriate. We could also check the child node of `price`, but it is not mapped. Even worse, it could be mapped to another OWL class that could also be part of the domain of the mapped OWL property! Clearly, this approach is not suitable. The domain must be specified.

The first approach to specify the domain is to use an XPath expression identifying the XML nodes mapped to the OWL class which is the domain of the property. Once again, we need to find another solution because an XML node can be simultaneously mapped to several OWL classes and the domain of the property can be the union of those several classes.

The best solution is to associate the datatype property mapping to a class mapping. This can be achieved using a triplet like

```
(OWL datatype property URI, domain class mapping, range XPath expression)
```

to specify a datatype property mapping. Considering the previous XML document and the following class mapping, `cm`:

```
cm = (product:computerProduct, /products/price/computer/product)
```

the following triplet `(product:price, cm, /product/price/@value)` is a valid datatype property mapping. It means that for each instance created from the `cm` class mapping, a datatype property `product:price` is also created and its value is filled using the one under `/products/price/@value`. Considering the previous XML document, the class mapping `cm` and the previous datatype property mapping, two instances of the OWL class `product:computerrProduct` are created: one for the product Philips Flat 32AB and the other one for SONY LCD 28TV (in fact one for each XML node matching the XPath expression used in the class mapping `cm`). For each of these instances, a datatype property `product:price` is created, whose value is found with `/products/price/@value`. In reality the value is not exactly found under the XPath expression used in the triplet identifying the mapping of the property. In order to

get the value used as the range of a property, it is necessary to compute the relative path from the XPath expression used in the class mapping to the XPath expression used in the property mapping. For instance, for each XML node mapped to `product:computerProduct`, the value of the property `product:price` is found under the relative path `../../@value` and the value of the property `product:name` is found under the relative path `name`.

3.5.2 Mapping XML nodes to OWL object properties

Mapping OWL object properties is very similar to the mapping of datatype properties. The difference occurs in the range of the property. While the range of datatype properties takes literal values, the range of object properties takes instances of OWL classes. The OWL object property is addressed like any other OWL resource (see Section 3.2). The domain is specified like the domain of datatype properties. For the same reasons of the domain of properties, the range of object properties is also referenced with a class mapping. As such, object property mappings are also specified with triplets:

```
(OWL object property URI, domain class mapping, range class mapping)
```

Let us consider an ontology with two OWL classes, `tourism:Country` and `tourism:City` and an object property `tourism:hasCity` whose domain and range are, respectively, `tourism:Country` and `tourism:City`. Let us also consider its inverse property `tourism:belongsToCountry` 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>
```

And the following mappings:

```
cm1 = (tourism:Country, /locations/location/country)
cm2 = (tourism:City, /locations/location/city)
```

The following triplets are valid object property mappings: `(tourism:hasCity, cm1, cm2)` and `(tourism:belongsToCountry, cm2, cm1)`. The first object property 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`. Again, in an identical way to the datatype properties, it is necessary to compute the relative path, which is in the example `../city`, to obtain the exact individual used as range. Running the instances' transformation over the considered XML document, four individuals are created: two instances of the OWL class `tourism:Country`, one for Portugal and one for France, as well as two instances for the OWL class `tourism:City`, one for Funchal and one for Paris. Four relationships between individuals (corresponding to the two object property mappings) are also created: two that relate Portugal and Funchal, and other two that relate France and Paris.

3.6 A Solution to the semantic heterogeneity problem: conditional mapping

In Section 2 we explained how the structural and syntactic problems of data integration are solved by JXML2OWL. We are now going to address the semantic obstacle in this subsection. The semantic heterogeneity problem was not completely solved by JXML2OWL application. Semantic incompatibility can arise in two different ways. The most simple semantic incompatibility situation happens when the same terminology is used with different but unique meanings by distinct data sources. For instance, a data source can use *customer* terminology with the meaning of *end-customers* while another source can use the same terminology but with a different meaning such as *dealers*. Since each data source uses the terminology with a unique and clearly defined meaning, it is just a matter of mapping the elements from each data source to the appropriate concept of the ontology. This situation is currently fully supported by the JXML2OWL mapping tool.

The second case, more complex than the first situation, happens when a terminology is used for more than one meaning in a data source. For example, this situation happens when a data source uses *customer* terminology to include both *end-customers* and *dealers*. To overcome this semantic problem, we propose two distinct solutions: pre-processing the data source and enabling conditional mappings within the JXML2OWL project. Considering the current state of JXML2OWL, it is necessary to pre-process the XML source using one of the available mapping tools supporting conditional mappings (such as Stylus Studio XML-to-XML Mapper) between two XML schemas to normalize the semantics. Then JXML2OWL can be used to map from the semantically normalized schema to an OWL ontology. Since the majority of XML mapping tools supports conditional mappings, we decided to concentrate our resources to define and implement a solution to map from XML to OWL, discarding the not so important conditional mapping feature.

Conditional mappings are an elegant solution to solve this problem. JXML2OWL was designed in such a way that it can easily be extended to support this kind of mapping. The following example explains how conditional mappings can be used to solve the semantic heterogeneity. Let us consider a `product` ontology with two concepts: `product:electronicProduct` and `product:computerProduct`. Let us also consider the following XML document.

```
<products>
  <product>
    <name>Philips Flat 32AB</name>
     <electronics>true</electronics>
   </product>
  <product>
    <name>SONY LCD 28TV</name>
     <computer>true</computer>
   </product>
 </products>
```

This XML document uses the same terminology, `product`, with different meanings: electronic and computer product. The following class mappings solve this semantic problem:

```
cm1 = (product:eletronicsProduct, /produtcs/product[electronics='true'])
```

```
cm2 = (product:computerProduct, /produtcs/product [computer='true'])
```

Predicates are used in the XPath expressions to elegantly solve the semantic heterogeneity problem.

3.7 Transforming XML instances to OWL instances

Two main problems need to be solved in order to successfully complete the JXML2OWL tool. The first problem deals with schema manipulations and it is related to the strategy that needs to be implemented to reference XML nodes and OWL resources, and the strategy to create mappings between classes and properties. The second problem that needs to be addressed concerns OWL instances generation. OWL instances are generated from the mappings created between the XML schema and the OWL ontology. This section addresses important aspects of the individual generation and creation of properties.

3.7.1 Generating class instances

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. With the support of many-to-one mappings, several XML nodes identified by the different or even by the same XPath expressions may refer to the same individual. Based on their unique identifier, duplicate instances (instances with the same ID) must be detected and filtered so that only one instance is created.

Bearing in mind what was said in the previous paragraph, an important decision must be taken with respect to how unique identifiers are created. JXML2OWL supports two approaches. By default, the ID is generated by sequentially concatenating the underscore symbol '_' with the prefix of the mapped class, with its local name and with the string-value [25] of the mapped XML node. Considering the following XML document, an ontology with the class `org:product` and these two class mappings: `(org:product, /org/products/product) and (org:product, /org/warehouses/warehouse/products/product)`, only two individuals are created, one for the product Philips Flat 32AB and the other for SONY LCD 28TV. The IDs of the two generated instances are `_orgproductPF32AB2` and `_orgproductSONY28TV3`. This happened because the string-values of the XML nodes representing each product are the same.

```
<org>
  <products>
    <product>
      <name> Philips Flat 32AB</name>
      <price>2500</price>
    </product>
    <product>
      <name>SONY LCD 28TV</name>
      <age>2400</age>
    </product>
  </products>
  <warehouses>
    <warehouse>
      <name>SEED Fx</name>
      <products>
```

```
            <product>
              <name>Philips Flat 32AB</name>
              <age>2500</age>
            </product>
            <product>
              <name>SONY LCD 28TV</name>
              <age>2400</age>
            </product>
          </products>
        </warehouse>
      </warehouses>
    </org>
```

One can argue, and we agree, that XML documents with the previous structure are rare since they are much too verbose and thus it could be quite impossible to detect duplicate instances using this approach in a real scenario (such as a dump of a database extracted in XML format). As such, we also propose an alternative solution where it is possible to specify the XML node whose string-value is used to generate the ID. With this alternative, the class mappings can also be specified with triplets:

(OWL class URI, XPath expression, ID XPath expression).

Considering now the following XML document, the same ontology with the class org:product and these two class mappings:

cm1 = (org: product, /org/products/product, /org/products/product/name) and cm2 = (org:product, /org/warehouses/warehouse/products/product, /org/warehouses/warehouse/products/product/@name), only two individual are created, one for the product Philips Flat 32AB and the other for SONY LCD 28TV because the XML nodes selected as ID have the same string-value. Now the IDs of the created instances are: _orgproductPF32AB and _orgproductSONY28TV.

```
    <org>
      <products>
        <product>
          <name> Philips Flat 32AB</name>
          <serial>333444555</serial>
          <price>25</price>
        </product>
        <product>
          <name> SONY LCD 28TV</name>
          <serial>666777888<</serial>
          <price>24</price>
        </product>
      </products>
      <warehouses>
        <warehouse>
          <name>SEED Lx </name>
          <products>
            <product name="Philips Flat 32AB">
              <serial>333444555</serial>
              <serial>333444555FX</serial>
            </product >
            <product name="SONY LCD 28TV"/>
          </products>
        </warehouse>
      </warehouses>
    </org>
```

One should note that using the string-value of an XML node to generate the IDs of OWL instances is not a perfect solution because the string-value can contain several symbols, such as '%' and ';', which are not valid within a unique identifier. However, since it is possible to specify the XML node used to generate the ID, one that does not contain such symbols can be chosen. Possible solutions to solve this problem consist of encoding such symbols into valid ones to generate a valid ID (such a valid string is called NCName [29] or use some kind of hash function to generate a valid ID from the string resulting of the several concatenations.

3.7.2 Generating properties

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. Special care must be taken because of the support of many-to-one mappings. In fact, with this kind of mapping, not only several XML nodes can be mapped to the same OWL class but also several XML nodes or class mappings can respectively be used as the range of OWL datatype or object properties. In such a case, 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.

With the intention of better understanding this situation, let us consider again the previous XML document, as well as the previous ontology with the same class mappings where the nodes used as IDs are directly specified, mapping different XML nodes to the same OWL class. Let us also consider that the ontology defines the datatype property `org:serial` as well as the followings triplets representing property mappings:

- `(org:serial, cm1, /org/products/product/serial)`
- `(org:serial, cm2, /org/warehouses/warehouse/products/products/serial)`

For the product Philips Flat 32AB, only one instance is created since the generated IDs are the same. However, two `org:serial` properties must be created, one for 333444555 and one for 333444555FX because they are distinct. One should note that one of the 333444555 serials 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. The generation of OWL instances must support those kinds of restrictions. 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 restriction 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 two distinct cases are:

- 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 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.

3.8 Summary

This section presents a notation, displayed in Table [2], to map an XML schema to an existing ontology defined in OWL and discussed important aspects regarding the transformation of instances of the XML schema into instances of the ontology. It does not constitute an exhaustive solution. Our purpose is to provide a solution that considers the most important aspects narrowing the huge gap existing between XML and OWL recommendations.

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)

Table 2. Mapping notation

4 How Organizations Can Use JXML2OWL?

Nowadays, an increasing number of organizations are operating in a global business environment. This global environment requires an adequate B2B integration for them to remain competitive. B2B integration is concerned essentially with the coordination of data, information, and processes among businesses and their information systems. Organizations need to avidly interact with suppliers, partners, and customers. However, if this integration is done on a point-to-point basis, these companies end up spending up to 35% to 40% of their software maintenance budgets simply on maintaining these connections [30].

4.1 B2B integration

Integration in a B2B context is hard to achieve since organizations use different vocabularies to describe their products, part numbers, invoices, and numbering purchase orders. As depicted by Figure 3, in B2B settings, it is possible to find two types of vocabularies: internal and external. Internal vocabularies are only visible inside organizations. Typically, organizations use data dictionaries and taxonomies to make their vocabulary explicit. Modern approaches would involve OWL, RDF, and RDFS in addition to XML in order to describe internal vocabularies and taxonomies.

On the other hand, external vocabularies are defined to be used with partners to exchange data and information. The most common way to describe an external vocabulary is to use a standard, such as RosettaNet and ebXML. Various industries have their own standards such as HL7 in the health care industry. In general, every

industry develops a standard or set of standards in order for companies in these industries to communicate with each other.

Organizations are starting to look for architectural solutions that allows their participation in B2B transactions using syntactic protocols (i.e. XML) while representing their internal vocabularies and documents semantically (using OWL). Partners and suppliers can freely exchange syntactic documents. Once an organization receives a syntactic document it is allowed to create a mapping between the elements from the document and concepts of an ontology that describes the domain of discourse of the organization (i.e., internal vocabulary). The organization that receives the documents can create any number of mappings.

4.2 Achieving B2B integration with JXML2OWL

As we can see, one of the main challenges of B2B integration is to find a solution to integrate internal and external vocabulary. This requires some expertise since these two types of vocabularies are specified using languages with different expressiveness. Nowadays, external vocabularies are usually specified using XML, while internal vocabularies recently started to be specified using RDF(S) and will possibly be specified using OWL in the near future.

While XML allows data exchange between distributed and heterogeneous applications, it does not guarantee the interoperability of systems. XML only provides syntax to structure the data exchanged in a B2B setting, since tags have no predefined meaning. This is only one level of interoperability that must be met in B2B transactions. Developers are still faced with the problem of semantic interoperability, i.e., the difficulty to integrate resources that were developed using different vocabularies and different perspectives on the data. When data is only defined syntactically, it is not possible to enable the automatic or semi-automatic integration of B2B information systems. These objectives can only be reached when considering the semantics of the data exchanged between organizations.

In these scenarios and contexts, the JXML2OWL tool can be effectively used to map external vocabularies to internal reference vocabularies (terminologies), and monitor, evaluate and correct deficiencies in external messages coming from outside suppliers, partners, and customers to conform to internal vocabularies. JXML2OWL narrows the gap between XML and OWL specifications proposing a strategy to map external vocabularies and documents represented with XML Schema to internal vocabularies and documents represented with existing OWL ontologies and transform XML data (instances of the mapped XML Schema) into instances of the ontology according to the performed mapping.

4.3 JXML2OWL implementation

As defined in Section 2, the JXML2OWL API and the Mapper tool were implemented in Java and the mapping rules are wrapped within an XSL document to automatically support the transformation of XML instances into individuals. The mappings process requires several steps. The first step consists of creating a new mapping project and loading both the XML schema related file (XSD or DTD) and the OWL ontology. If an XML schema is 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 concepts of the ontology. Once these mappings are created, it is possible to relate them to each other with the intent of creating 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. Regarding the mappings, JXML2OWL supports one-to-one, many-to-one, one-to-many and many-to-many mappings. This means that an element of the loaded XML schema can be mapped to several OWL classes and several elements of the schema can correspond to the same OWL class. These kinds of supported mappings allow the mapping between any XML schema to any ontology. Other features are provided such as the ability to save a mapping project state in an XML file to resume it later or the possibility to directly transform XML instances document to OWL individuals using JXML2OWL Mapper. The main methods provided by the API are directly derived from the notation proposed on Section 3. The Mapper supports all the features of the API in a user-friendly way. Figure 4 illustrates the JXML2OWL Mapper tool with several mappings created.

The JXML2OWL Mapper tool is divided into two main parts. 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. For example, the `/lecturers/lecturer` node from the XML schema is mapped to the OWL class `teacher:Teacher` of the ontology while `/lecturers/lecturer/teaches/course` is mapped to `teacher:Course`. Those two class mappings are related with an object property mapping (the selected item of the table). Datatype property mappings are also displayed such as `teacher:age` and `teacher:email`.

To assess the performance of the instance transformation process, we have created a mapping project with 9 class mappings, 14 datatype property mappings and also 14 object property mappings. We have transformed three XML documents with different sizes, which validates against the mapped schema, using the generated XSL document. The performance results are shown in Table [3].

Lines (XML File)	Size In (XML File)	Size Out (OWL instance File)	Time Processing
385	10.2 KB	28 KB	0.266s
3805	102 KB	254 KB	3.734s
38005	1068 KB	1943 KB	5m 14.609s

Table 3. Performance assessment

During our performance evaluation, we noticed that the processing time did not scale very well with the size of the XML input document. This is mainly due to the process of detecting and eliminating duplicate instances and properties. Such process

requires several passes (the exact number depends on the quantity of many-to-one mappings) through the XML instances document which is time-consuming.

4.4 Commercial applications already available

Commercial systems and tools that use RDF and OWL as a representation language are emerging. This section illustrates only a few of the most promising solution already available in the market to deploy semantic Web applications. These applications can be sued in conjunction with JXML2OWL.

For example, Altova SemanticWorks™ 2007 (www.altova.com) is the ground-breaking visual RDF/OWL editor. This tool allows the visual creation and editing of RDF, RDF Schema, OWL Lite, OWL DL, and OWL Full documents using an intuitive, visual interface and drag-and-drop functionality.

Oracle Spatial 10g (www-oracle.com) has introduced the industry's first RDF management platform. Based on a graph data model, RDF triples are persisted, indexed and queried, similar to other object-relational data types and allow deploying scalable and secure semantic applications. Oracle's RDF Database (11g) will support native OWL inferencing for an OWL subset that includes property characteristics, class comparisons, proprety comparisons, individual comparisons and class expressions. Metatomix (www.m3t4.com) has developed the Metatomix Semantic Toolkit, which is a set of Eclipse plugins that allow developers to create and manage ontologies based on the OWL standards.

TopQuadrant (www.topquadrant.com) has released its TopBraid Composer, a professional development environment for W3C's Semantic Web standards: RDF Schema, OWL, SPARQL Query Language and the Semantic Web Rule Language (SWRL).

5 Conclusions

Data storage technologies have evolved together with the needs of enterprises. Initially stored in flat files in a proprietary format, data stored in tables managed by RBDMS emerged in the 70's with the need for better performances, while SQL query language became a standard in the 80's. With the advent of the internet and XML as a de facto standard for B2B data exchange, traditional EDI (Electronic Data Interchange) solutions (such as Edifact) are being substituted by XML based EDI (such as ebXML and Rosettanet). To support these evolutions, mapping tools were developed to allow the mapping between distinct technologies and schemas. For instance, tools enabling mappings and data conversion from flat file do Relational Databases are available. Similarly, with the emergence of XML, applications supporting mappings from flat file to XML, and between Relational Databases and XML (through SQL) are getting more common. Since a few years ago, to better enable data exchange and integration, common database vendor enabled XML within their databases. MS SQL Server and IBM DB2 are examples of such databases which are usually referred to as XML-enabled databases. More recently, native XML databases emerged. They are document centered and are particularly suited to store, manage and query XML documents usually using XPath and / or XQuery language. XML technologies brought interoperability at a syntactic level, but today's organizations are again shifting (or it is expected of them to

do so) from a syntactic operability level to a semantic one [31]. Semantic Web technologies, such as RDF and OWL, play an important role in achieving this objective. Currently, not only databases (such as Oracle 10g), but also applications (like Adobe Creative Suite or Mozilla Firefox), are RDF-enabled. According to William Ruh of CISCO, before the end of 2004, RDF was applied under the covers of well over 100 identified products and over 25 information service providers [32]. Again, mapping applications were developed to support this evolution. For example, several XSL stylesheets, such as xml2rdf.xsl [33], enable conversion from XML to RDF. At the top of RDF stands OWL, a semantically richer and more expressive language. Since OWL recommendation is very recent, applications using OWL are very scarce and mainly related to academic projects. However, for its acceptance, tools supporting OWL standard are necessary. Mapping tools to OWL specification are also needed to assist enterprise migration from syntactic to semantic data structures.

In this paper we have presented an approach, which was successfully implemented in JXML2OWL, for mapping XML schema to existing OWL ontologies and transforming instances of the XML schema into individuals. This transformation is fundamental for organizations that plan to move from a syntactic representation of data (using XML) to a semantic one (using OWL). By using semantic domain models based on ontologies, enterprises acquire several benefits, such as the ability to perform inference on a knowledge base (ontology and its individuals) to derive potentially new knowledge or the capacity to share their domain model to easily exchange and integrate data.

JXML2OWL has been successfully employed in the context of a major project called SEED (SEmantic E-tourism Dynamic packaging) whose purpose is to integrate disparate and heterogeneous e-tourism data sources into a unique knowledge base. We believe the presented framework is appropriate to integrate any XML data into semantic information systems based on OWL ontologies. The JXML2OWL framework is ready-to-use and available for download (http://jxml2owl.projects.semwebcentral.org/).

We hope the research done to bridge the gap between XML and OWL as well as the successfully implemented prototype has demonstrated the need for semantic mapping tools and will stimulate R&D departments of software companies, mainly the ones developing mapping applications, to develop professional mapping tools supporting mappings and instances transformation to OWL ontologies.

6 Acknowledgments

This work was partially funded by grants from the FCT, POCTI-219, and FEDER.

7 References

[1] Berners-Lee T, Miller E. The Semantic Web lifts off. Special Issue of ERCIM News 2002; 51:9-11. http://www.ercim.org/publication/Ercim_News/enw51/berners-lee.html.

[2] Cardoso J. Semantic Web Services: Theory, Tools and Applications. New York: IGI Global; 2007.

[3] Davis M. The Business Value of Semantic Technologies. A TopQuadrant Special Report 2004. http://www.knowledgefoundations.com/pdf-files/BusinessValue_v2.pdf

[4] Oracle. Semantic Data Integration for the Enterprise (White paper); 2007. http://www.oracle.com/technology/tech/semantic_technologies/pdf/semantic11g_dataint_twp.pdf

[5] Fensel D. Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce. 2nd ed. Berlin: Springer-Verlag; 2003.

[6] Yager T. The Future of Application Integration. InfoWorld; 2002. http://www.infoworld.com/article/02/02/22/020225feintro_1.html.

[7] White C. Data Integration: Still a Barrier for Most Organizations. April 2006 Issue of DM Review. http://www.dmreview.com/article_sub.cfm?articleId=1051163.

[8] Alexie V, Breu M, de Bruijn J, Fensel D, Lara D, Lausen H. Information Integration with Ontologies: Experiences from an Industrial Showcase. John Wiley & Sons; 2007.

[9] Roth M. A, Wolfson D. C, Kleeweln J. C, Nelin C. J. Information integration: A new generation of information technology. IBM Systems Journal 2002; 41(4):563-577.

[10] Gold-Bernstein B. Enterprise Information Integration – What was old is new again. ebizQ, 2004. http://www.ebizq.net/topics/eii/features/4371.html?page=1.

[11] Sheth A. A Semantic Meta Data Approach to Enterprise Information Integration. July 2003 Issue of DM Review. http://www.dmreview.iproduction.com/issues/20030701/6962-1.html

[12] Gruber T. A translation approach to portable ontology specifications. Knowledge Acquisition 1993; 5(2): 199-220.

[13] Silva B, Cardoso J. Semantic Data Extraction for B2B Integration. International Workshop on Distributed Applications for B2B Integration (DABI) 2006. Lisboa, Portugal, IEEE Computer Society.

[14] Cardoso J. Integrating HAD Organizational Data Assets using Semantic Web Technologies. 3rd International Conference Interoperability for Enterprise Software and Applications (I-ESA 2007), Funchal, Portugal. In: Gonçalves R, Müller J, Mertins K, Zelm M. Enterprise Interoperability II 2007. Springer. p. 333-344.

[15] Bussler C. B2B Integration: Concepts and Architecture, Springer-Verlag; 2003.

[16] Horrocks I, Patel-Schneider P, Boley H, Tabet S, Grosof B, Dean M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. 2003. http://www.daml.org/2003/11/swrl/.

[17] Horrocks I, Patel-Schneider P. A Proposal for an OWL Rules Language (Draft Version of 16 October 2003). http://www.cs.man.ac.uk/~horrocks/DAML/Rules/.

[18] Cardoso J, Miller J, Su J, Pollock J. Academic and Industrial Research: Do their Approaches Differ in Adding Semantics to Web Services. In: Cardoso J, Sheth A. Semantic Web Process: powering next generation of processes with Semantics and Web services. Heidelberg: Springer-Verlag; 2005.

[19] Ehrig M, Sure Y. FOAM - Framework for Ontology Alignment and Mapping: Results of the Ontology Alignment Initiative. In: Ashpole B, Ehrig M, Euzenat J, Stuckenschmidt H. Workshop on Integrating Ontologies. Alberta, Canada: CEUR-WS. 156:72-6; 2005.

[20] Aumueller D, Do H, Massmann S, Rahm E. Schema and ontology matching with COMA++. In: International Conference on Management of Data 2005; Baltimore: ACM Press, p. 906-8

[21] Ferdinand M, Zirpins C, Trastour D. Lifting XML Schema to OWL. In: Koch N, Fraternali P, Wirsing M. 4th International Conference Web Engineering 2004. Heidelberg: Springer, p. 354-8.

[22] Garcia R, Perdrix F, Gil R. Ontological Infrastructure for a Semantic Newspaper. In: First International on Semantic Web Annotations for Multimedia Workshop (SWAMM'06); 2006.

[23] Bohring H, Auer S. Mapping XML to OWL Ontologies. In: Jantke K, Fähnrich K, Wittig W. Marktplatz Internet: Von e-Learning bis e-Payment: 13. Leipziger Informatik-Tage (LIT2005). Leipzig, p. 147-156.

[24] Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F. Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation; 2004. http://www.w3.org/TR/REC-xml/.

[25] Bechhofer S, Van Harmelen F, Hendler J, Horrocks I, McGuinness D, Patel-Schneider P, et al. OWL Web Ontology Language Reference. W3C Recommendation; 2004. http://www.w3.org/TR/owl-ref/.

[26] Clark J, DeRose S. XML Path Language (XPath). W3C Recommendation; 1999. http://www.w3.org/TR/xpath.

[27] Klyne G, Caroll J, McBride B. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation; 2002. http://www.w3.org/TR/rdf-concepts/.

[28] Cardoso J, Sheth A. Semantic Web Services, Processes and Applications. Springer; 2006.

[29] Bray T, Hollander D, Layman A, Tobin R.Namespaces in XML 1.0 (Second Edition). W3C Recommendation; 2006. http://www.w3.org/TR/REC-xml-names/#NT-NCName.

[30] Mitem, Information Integration for the Public Sector, 2002, http://www.mitem.com/customers/documents/MITEM-InformationIntegrationforthePublicSector.pdf

[31] EBizQ. Semantic Integration: A New Approach to an Old Problem. Software AG; 2005. http://www.sagus.com/media/PDFs/SoftwareAG_eBizQ_062005.pdf.

[32] Ruh W. The Web of Meaning: The Business Value of the Semantic Web. Cisco Systems; 2004. http://www.w3.org/2004/Talks/w3c10-WebOfMeaning/Originals/Ruh.ppt.

[33] DuCharme B. Converting XML to RDF. O'Reilly xml.com; 2004. http://www.xml.com/pub/a/2004/09/01/tr.html.

8 Vitae

Toni Rodrigues, a software designer and developer, joined SQLI (France) in 2006. Since then he is working as a software consultant in financial messaging and SWIFT payment systems. He previously was a member of the SEED Laboratory, a group working with Emergent Information Systems. As a member of this laboratory, he wrote his Master thesis on semantic data integration and was graduated by the University of

Madeira in Computer Science. Current areas of interest are predominantly concerned with service oriented architectures, web services as well as semantic web.

Pedro Rosa, a software developer joined Expedita – Arquitectura e Gestão de Sistemas de Informação in 2006.Working as consultant and developer of web based applications related to Tourism and Maritime Transportation sector. His interest areas are mainly service oriented architecture, system integrations and Semantic Web. Graduated in Computer Science by University of Madeira, he worked there as monitor and wrote is master thesis on semantic data integration as a member of the SEED laboratory.

Prof. Dr. Jorge Cardoso joined SAP Research (Germany) in 2007. He is also Professor at the University of Madeira (Portugal). He previously gave lectures at the University of Georgia (USA) and at the Instituto Politécnico de Leiria (Portugal). In 1999, he worked at the Boeing Company on enterprise application integration. Dr. Cardoso was the organizer of several international conferences on Semantic Web and Information Systems. He has published over 80 refereed papers in the areas of workflow management systems, semantic Web, and related fields. He has also edited 3 books on semantic Web and Web services. He is on the Editorial Board of the Enterprise Information Systems Journal, the International Journal on Semantic Web and Information systems, and the International Journal of Information Technology.

9 Figure Captions

10 Table Captions