International Journal of Applied Engineering and Technology ISSN: 2277-212X (Online)
An Open Access, Online International Journal Available at http://www.cibtech.org/jet.htm
2014 Vol. 4 (1) January-March, pp.17-31/Bahaj and Baroudi
Research Article

INTEGRATING ONTOLOGIES INTO EBXML REGISTRIES FOR EFFICIENT SERVICE DISCOVERY

***Mohamed Bahaj[1] and Salim Baroudi[1]**
[1]*Department Of Computer Science, Faculty of Science And Technology, University Hassan I, Settat, Morocco*
**Author for Correspondence*

ABSTRACT
In our days, ebXML and Web Services are two modern technologies used to perform collaboration and interactions between businesses in B2B contexts. In this paper, we expose an efficient way to improve the discovery of the services mechanism by introducing semantic in the ebXML registries. This semantic is represented by storing OWL constructs in the ebXML registries, making the business context more comprehensible by the other businesses and then making the business collaboration more powerful and efficient for a Web Services more more and Semantic (Akkiraju, 2005).

Keywords*: ebXML, Web Services, ebXML Registries, Semantic, Web Ontology Language (Owl), Business To Business (B2b) Interactions, Web Service Semantic.*

INTRODUCTION
Electronic Business XML (ebXML) (Romin, 2002) is a standard from OASIS (Antonio *et al.,* 2008) and United Nations Center for Trade Facilitation and Electronic Business, UN/CEFACT that is technology who provides an infrastructure permitting to the enterprises to find each other's services, products, business processes, and documents in a standard way and thus helps to facilitate conducting electronic business.
The most important characteristic of ebXML registries that we focused in this article is that they provide mechanisms to define and associate semantics with registry objects.
ebXML 'Registry' and 'Repository' are two different components used to store semantic - meta-data and data- for the service. The registry meta-data are stored in the 'Registry', and the documents pointed at by the registry objects reside in a 'Repository'.
Different approach are used to bring semantic to services based ebXML registries, in this article we use classification hierarchies that consist on designing and Ontology for the business domains using OWL (Mcguinness & Harmelen, 2007), and applying a mapping mechanism to this Ontology to get semantically constructs who will be stored in the Registry, and we will show how this approach can be useful to resolve to problems related to services discovery that can compromise all the collaboration process within the volume of the business environment and their heterogeneity.
This article is divided into five sections; in the introduction we introduce our work domain and context. In the second and third section, we define the ebXML and OWL with all important characteristics and aspects that we will use or exploit in this work (ebXML RIM specifications, Description Logic ... etc.). In the fourth part, we develop and explain the proposed approach for improving the service discovery mechanism that is divided into four steps. And finally we will conclude by citing the related works and perspectives.

ebXML
ebXML facilitates electronic business as follows:
- In order for enterprises to conduct electronic business with each other, they must first discover each other and the products and services they have to offer. ebXML provides a registry where such information can be published and discovered,

- An enterprise needs to determine which business processes and documents are necessary to communicate with a potential partner. A Business Process Specification Schema (BPSS) in ebXML provides the definition of an XML document that describes how an organization conducts its business,
- After this phase, the enterprises need to determine how to exchange information. The Collaboration Protocol Agreement (CPA) specifies the details of how two organizations have agreed to conduct electronic business.

ebXML registry architecture and information model

ebXML registry (Antonio *et al.*, 2008) provides a persistent store for registry content. It's consists of both a registry and a repository. The repository is capable to store any type of electronic content, while the registry is capable of storing meta-data that describes content. The content within the repository is referred to as 'repository items' while the meta-data within the registry is referred to as 'registry objects'. The repository store registry data in relational databases. ebXML Registry Services Specification defines a set of Registry Service interfaces which provide access to registry content. There are a set of methods that must be supported by each interface. Like shown in the ***Figure 1*** a registry client program utilizes the services of the registry by invoking methods on one of these interfaces. The Query Manager component also uses these methods to construct the objects by obtaining the required data from the relational database through SQL queries. In other words, when a client submits a request to the registry, registry objects are constructed by retrieving the related information from the database through SQL queries and are served to the user through the methods of these objects.

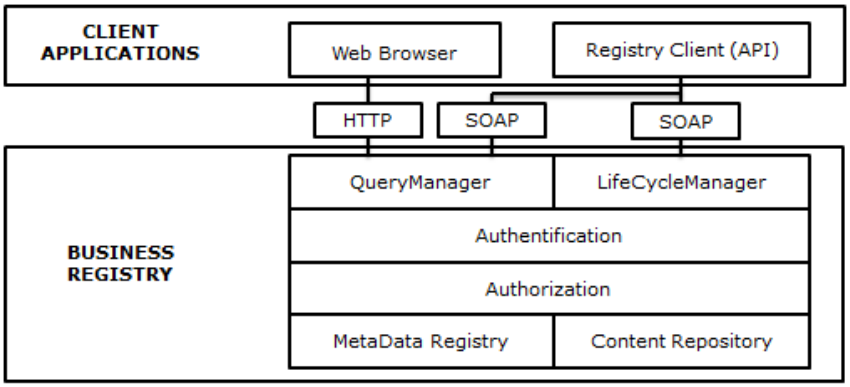

Figure 1: View of ebXML Registry Architecture

An ebXML registry allows us to define semantics basically through two mechanisms: first, it allows properties of registry objects to be defined through 'slots' and, secondly, meta-data can be stored in the registry through a 'classification' mechanism. This information can then be used to discover the registry objects by exploiting the ebXML query mechanisms.

It should be noted that within the scope of this paper, we address how to bring semantic to the ebXML registry using the Ontology concept.

ebXML registry information model (RIM)

ebXML registry information model. The ebXML registry defines a Registry Information Model (RIM) which specifies the standard meta-data that may be submitted to the registry. This complements the ebXML Registry API which defines the interface clients may use to interact with the registry. The ***Figure 2*** presents the part of the ebXML RIM related with storing meta-data information. The main features of the information model include:

- *Registry Object*: The top level class in RIM is the *Registry Object*. This is an abstract base class used by most classes in the model. It provides minimal meta-data for registry objects,
- *Object identification* : All *Registry Objects* have a globally unique id, a human friendly name and a human friendly description,

- *Slot* : *Slot* instances provide a dynamic way to add arbitrary attributes to *Registry Object* instances,
- *Object Classification*: Any *Registry Object* may be classified using *Classification Schemes* and *Classification Nodes* which represent individual class hierarchy elements. A *Classification Scheme* defines a tree structure made up of *Classification Nodes*. The *Classification Schemes* may be user-defined,
- *Object Association*: Any *Registry Object* may be associated with any other *Registry Object* using an Association instance where one object is the *source Object* and the other is the *target Object* of the Association instance. An Association instance may have an *association Type* which defines the nature of the association,
- *Object Organization*: *Registry Objects* may be organized in a hierarchical structure using a familiar file and folder metaphor. The *Registry Package* instances serve as folders while *Registry Objects* serve as files in this metaphor. In other words *Registry Package* instances group logically related *Registry Object* instances together,
- *Service Description*: The Service, *Service Binding* and *Specification Link* classes provide the ability to define service descriptions including WSDL and ebXML CPP/A.

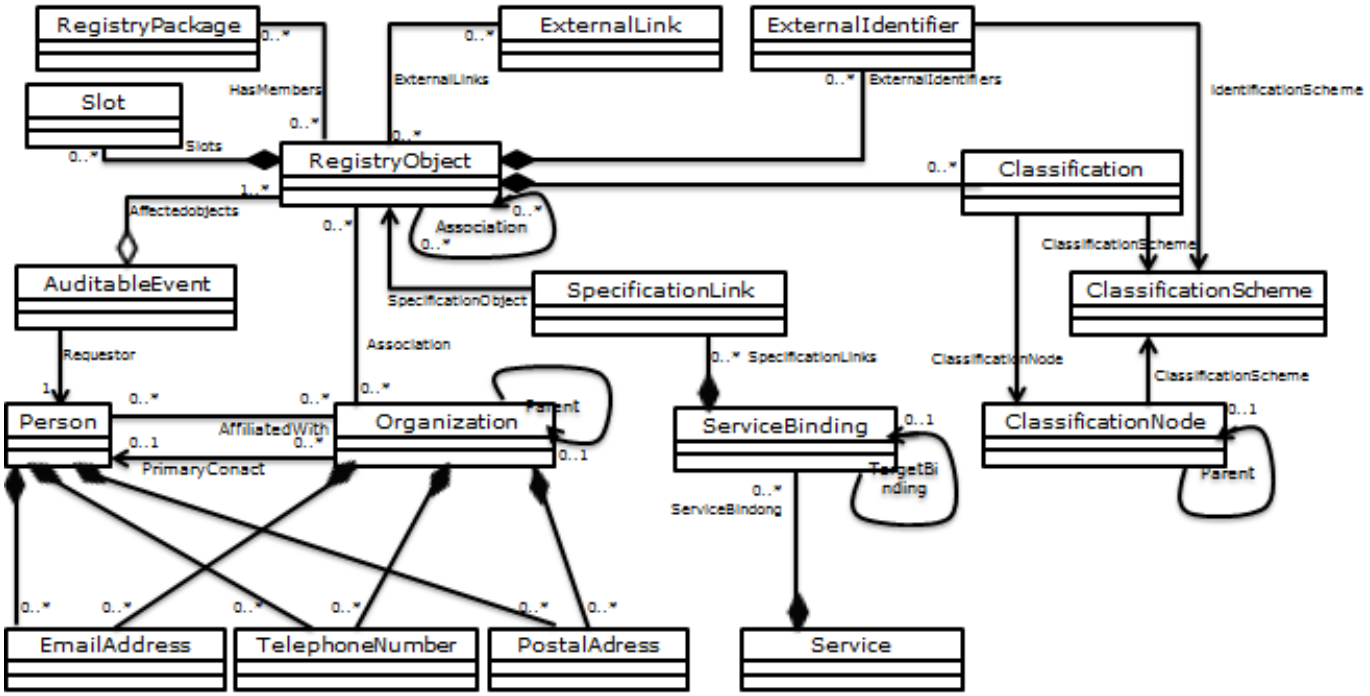

Figure 2: ebXML RIM - Information Model Relationships view

Ontology and Web Ontology Language (OWL)

Ontology

Ontology is a description of a content and relationship expressed as a formal vocabulary. This can be used to classify domain specific information in different areas such as in medicine, education, engineering, etc. Most of the business information stored in repositories in Business to Business (B2B) infrastructure frameworks is difficult to extract due to unstructured domain classifications in business registries (Gunatilaka, 2004).

Therefore use of ontology as an information source to formalize domain knowledge and finding a way of mapping this information to registry content is a practical solution to enhance B2B communication. We have used OWL ontology language to represent the web semantic content in the repository.

Web Ontology Language (OWL)

Web Ontology Language (OWL) (Mc Guinness & Harmelen, 2007) is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is derived from the DAML+OIL Web Ontology Language (Antonio & Harmalen, 2004; Mcguiness *et al.*, 2001), and builds upon the Resource Description Framework (RDF) (Miller, 1998; Pan, 2009). It uses Description Logic (Gagnon, 2009) as a main construct to specify information related to a specific domain.

OWL describes the structure of a domain in terms of classes, data type and object properties. Classes can be names (URIs) or expressions and the following set of constructors are provided for building class expressions: *owl: intersection of, owl: union of, owl: complement Of, owl: one Of, owl: all Values From, owl: some Values From, and owl:* has Value -***Figure 3***-. Properties can have multiple domains and multiple ranges. Multiple domain (range) expressions restrict the domain (range) of a property to the intersection of the class expressions.

Another aspect of the language is the axioms supported. These axioms are specials properties that make possible to assert subsumption or equivalence with respect to classes or properties. The following are the set of OWL axioms : *rdfs: sub Class Of, owl:sameClassAs, rdfs:subPropertyOf, owl:samePropertyAs, owl:disjointWith, owl:sameIndividualAs, owl:differentIndividualFrom, owl:inverseOf, owl:transitiveProperty, owl:functionalProperty, owl:inverseFunctionalProperty* ***Figure 3***.

RDF Schema Features	**(In)Equality**	**Property Characteristics**
– *Class (Thing, Nothing)* – *rdfs: subClassOf* – *rdf:Property* – *rdfs: subPropertyOf* – *rdfs:domain* – *rdfs:range* – *individual*	– *equivalentClass* – *equivalentProperty* – *sameAs* – *differentFrom* – *AllDifferent* – *distinctMembers*	– *ObjectProperty* – *DatatypeProperty* – *inverseOf* – *TransitiveProperty* – *SymmetricProperty* – *FunctionalProperty* – *InverseFunctionalProperty*
Property Restrictions – *Restriction* – *onProperty* – *allValuesFrom* – *someValuesFrom*	**Restricted Cardinality** – *minCardinality* – *maxCardinality* – *cardinality*	**Datatypes** –*xsd datatypes* **Class Intersection** –*intersectionOf*

Figure 3 : Example of OWL Constructs and Properties

According to the complexity of the business domain, OWL is subdivided to three sub-languages: OWL Lite, OWL DL, and OWL Full.

OWL Lite: It's the elementary level of OWL. It's based on a low hierarchy class level.

OWL DL: It's more expressive than the OWL Lite, and support a maximum of expressiveness, decidability and completeness based on Description Logic defined in ***section 3.3***.

OWL Full: Support a maximum of expressiveness and freedom is given to users to define their own RDF formats.

Description Logic (DL)

What we mean by description logic is actually a family of formalisms for representing a basic knowledge of an application domain. More specifically, these logical can represent concepts -Table I- (also called classes) of a domain and the relationships -Tableau II- (also called roles) that can be established between the instances of these classes.

The Constructors include Union, Intersection, Negation, Existential restriction, Value Restrictions ... etc. Axioms are categorized into two types, namely, Definition axiom and Inclusion axiom. Definition axioms introduce names for concepts and Inclusion axiom asserts subsumption relations.

Table I : OWL Constructors and their equivalent in DL

Constructor	DL Syntax
interSectionOf	$C1 \bigcap C2$
unionOf	$C1 \bigcup C2$
complementOf	$\neg C1$
oneOf (enumeration)	$\{x_1, x_2, \ldots x_n\}$
allValueFrom	$\forall P.C$
someValueFrom	$\exists P.C$
hasValue	$\exists P.\{X\}$
minCardinality	$\exists >= nP$
maxCardinality	$\exists <= nP$
cardinality	$\exists = nP$

Tableau II : OWL Axioms and their equivalent in DLs

Axioms	DL Syntax
subClassOf	$C1 \subseteq C2$
equivalentClasses	$C1 \equiv C2$
subProperty	$P1 \subseteq P2$
equivalentProperty	$P1 \equiv P2$
sameAs	$\{X1\} \equiv \{X2\}$
disjointWith	$C1 \subseteq \neg C2$
differentIndividualAs	$\{X1\} \subseteq \neg\{X2\}$
inverseAs	$P1 \equiv P2^{-1}$
transitiveProperty	$P+ \subseteq P$

Proposed approach for improving the service discovery mechanism

In this section, and in order to introduce semantic in ebXML registries, we describe how OWL ontologies can be used and stored in ebXML registries.

As known, OWL is an ontology language widespread to structure and represent knowledge in the Web. Our mapping approach consist on using the benefit of the semantic bringing of this language by introducing it into the ebXML registries by doing a mapping mechanism like shown in the ***section 4.1***.

Our approach consists in three primordial steps:

i. Designing the Ontology for the business domain,

ii. Writing this Ontology with OWL DL ontology language,
iii. Applying the proposed mapping approach based classification - ***section 4.1***- to map this ontology to a ebXML RIM specifications,
iv. Storing the resulting Constructs -semantically rich- in the ebXML registry.

In this article, we use the OWL DL ontology language to assume a medium level of expressiveness and decidability.

Mapping OWL to ebXML RIM specifications proposed approach

In this section, we will develop the mapping approach used for bringing semantic to the ebXML registries. Our approach consists on three big steps mapping:

i. Classification hierarchies mapping,
ii. Properties mapping,
iii. Individuals mapping,
iv. Organizations mapping,
v. Collection hierarchies mapping,
vi. Service mapping.

Classification hierarchies mapping

The first step of this approach consist on mapping the OWL hierarchical classes constructs to classes constructs over the ebXML classification hierarchies.

The OWL hierarchy classes are classes built over relationship with other classes -The opposite of simple classes-.

The OWL relationships between classes and subclasses will be mapped to classification hierarchies using the registry objects *ClassificationScheme*, *ClassificationNoes* and Classification ***Figure 2*** as shown in the example

Figure 4.

```
<ClassificationScheme id="Goods">
    <ClassificationNode id="Book" parent="Goods" code="book" />
    <ClassificationNode id="CD" parent="Goods" code="cd" />
</ClassificationScheme>
```

Figure 4 : Classification hierarchies mapping example

Properties maping

The OWL -OWL DL- properties mapping consist of applying a conversion to this properties to get an ebXML Association Object -***Figure 2***-.

The OWL *DataTypeProperty* and *ObjectProperty* relationships are mapped to object registry Associations. The type of the Association is specified depending of the predefined types of ebXML RIM Association (Contains, Functional, Restriction ... etc).

The *SourceObject* of the Association is the domain of the OWL property, and the *TargetObject* is the range of the property, as shown in the example -***Figure 5*** and ***Figure 6***-.

To make difference between the *DataTypeProperties* and *ObjectProperties* Associations type, we choose to add a label to the beginning of the id of the Association to have for the examples -***Figure 5*** and ***Figure 6***- *id="DataType-title"* and *id="Object-autor"*.

Moreover, the Restriction properties mapping, are mapped to Restriction *AssociationType* that used slots to enumerate Object.

This mapping is usually used to map the properties like *hasValue*, *oneOf*, *sameAs* ... example -***Figure 7***-.

```
<Association id="DataType-title"
    associationType="contains"
    sourceobject="Book"
    targetObject="http://www.w3.org/2001/XMLSchema#string"
/>
```

Figure 5: DataTypeProperty mapping example

```
<Association id="Object-autor"
    associationType="contains"
    sourceobject="Book"
    targetObject="Autor"
/>
```

Figure 6: ObjectProperty mapping example

```
<Association id="DataType-isBook"
    associationType="restriction"
    sourceobject="Book"
    targetObject="http://www.w3.org/2001/XMLSchema#boolean">
    <slot name="hasValue">
        <slotList>
            <value>true</value>
            <value>false</value>
        </slotList>
    </slot>
</Association>
```

Figure 7: HasValue Property mapping example

Individuals mapping

The individuals mapping consist on mapping all the physical classes' implementations Classes instances. The Classes individuals are mapped to registryObjects of the Individual Class like show in **Figure 8** and ***Figure 9***.

```
<registryObject id="Book-id">
    <ClassificationNode id="Book"/>
</registryObject>
```

Figure 8 : Individual -Simple class implementation- mapping example

```
<registryObject id="Goods-id">
    <ClassificationScheme id="Goods"/>
</registryObject>
```

Figure 9 : Individual -Hierarchical class implementation- mapping example

Organizations mapping

The Organization mapping consists on mapping all the information related to the organization that publishes her Service. It must contain all the information about the organization (name, contact, address, phone number) as represented in the object organization of the schema ebXML RIM.

This step, don't represent a mapping of a pure OWL constructs, is based on structuring information about organizations to facilitate identification and organization of big business domains. The mapping of the Organization represents the Organizations classes' hierarchy -***Figure 10***-.

```
<organization id="FSTS">
    <nameOrganisation>
        <localizedString lang="fr" value="fst settat"/>
    </nameOrganisation>
    <serviceIndustry>
        <localizedString lang="fr" value="Education institut"/>
    <serviceIndustry>
    <address
        country="MORROCO"
        city="SETTAT"
        street="Km 3, B.P.:577 Route de Casablanca"
    />
    <telephoneNumber
        contryCode="212"
        number="0523400736"
    />
    <webSite
        domain="ac.ma"
        site="www.fsts.ac.ma"
    />
    ...
</organization>
```

Figure 10: Organization mapping example

Collection hierarchies mapping

The Collection hierarchies are classes built by applying a collection property -like *unionOf*, *complementOf* ... properties- to other classes. These types of Collection hierarchies are mapped to *RegistryPackage* object in the resgistry ebXML ***Figure 11***.

```
<registryPackage id="goodsUnion">
    <registryObject id="Book-id">
        <ClassificationNode id="Book"/>
    </registryObject>
    <registryObject id="CD-id">
        <ClassificationNode id="CD"/>
    </registryObject>
    <registryObject id="Laptop-id">
        <ClassificationNode id="Laptop"/>
    </registryObject>
</registryPackage>
<Association id="Object-goodsAssociation"
    associationType="unionOf"
    sourceobject="goods"
    targetObject="goodsUnion"
/>
```

Figure 11: Collection hierarchies mapping example

Research Article

Service mapping

In this step of mapping, we convert all the information about the organization services, (Description of the service -WSDL file-, information about the methods of the service and their inputs and outputs, URI of the service, etc.). The Service Object is an encapsulation of a *servicebinding* -that contains the access information to the service (i.e.: URI) and *specificationLink* -that contains the description of the service (wsdl file)- like shown in the example ***Figure 12***.

The mapped service will be stored directly in the registry and can be visible for all the inter-operating business domains. For this, the domain service will be very rich of semantic (Result of the OWL constructs mapping that are semantically significant).

```
<service id="FSTS-service">
    <name>
        <localizedString lang="fr" value="Registre du service de l'FSTS"
    </name>
    <description>
        <localizedString lang="fr" value="FSTS-services"
    </description>
    <serviceBinding
        id="FSTS-serviceBinding"
        accessURI="http://services.fsts.ac.ma:8080/soapservices"
    >
        ...
        <specificationLink
            id="FSTS-serviceSpecificationLink"
            specificationObject="FSTS-serviceWSDLFile"
        >
            ...
        </specificationLink>
    </serviceBinding>
<service>
```

Figure 12: Service mapping example

Proposed tool

For this goal, we developed a tool that allows us to get the mapped OWL constructs to ebXML RIM following the work flow presented in the ***Figure 13***.

The conversion process is organized into two big steps that are, Loading structure and Mapping these structures to the OWL Constructs within the ebXML RIM specification using the Java API Jena, for parsing and manipulation the Ontology.

The Loading structure step that consist on loading all the ontology data to the Java objects. Examples in the Figure 14, Figure 15, Figure 16, Figure 17, Figure 18,

Figure ***19*** and Figure 20.

The Mapping step that consists on converting the ontology to the get the ebXML semantic constructs. Examples in the figures Figure 21, Figure 22, Figure 23, Figure 24 and Figure 25.

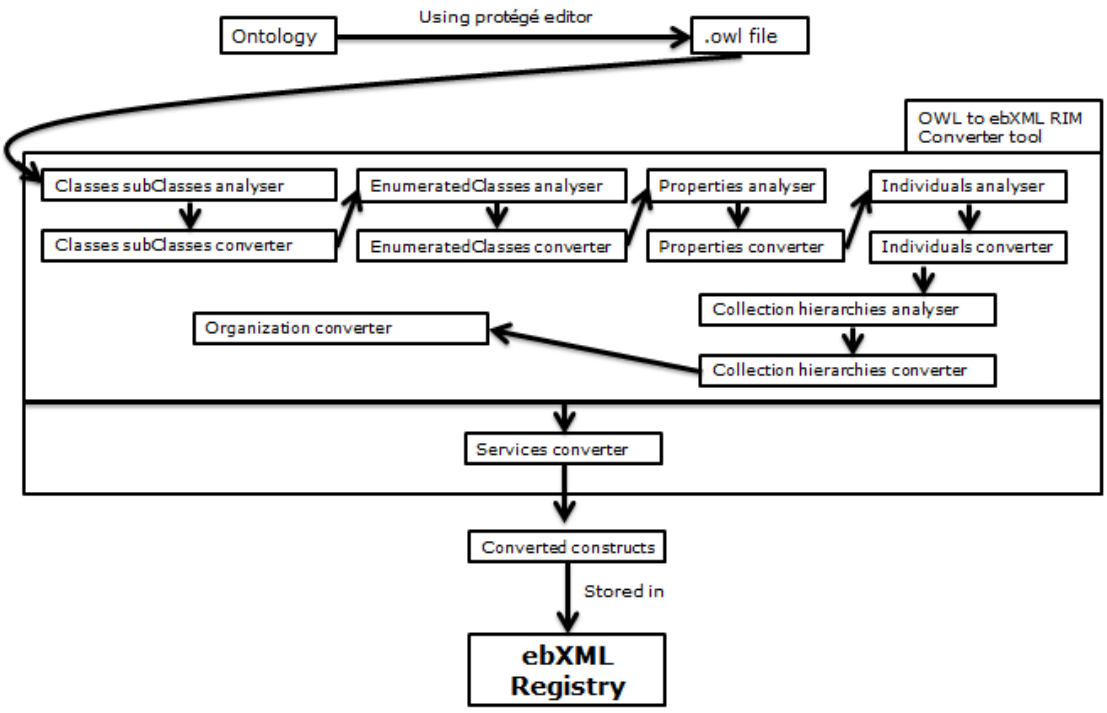

Figure 13: Mapping tool workflow

```
function ontologyFileLoader(ontologie) {
    Model model = ModelFactory.createOntologyModel();

    InputStream in = FileManager.get().open("ontologie.owl");
    if (in == null) {
    throw new IllegalArgumentException(
    "File: ontologie.owl not found");
    }
    ontologyModel = (OntModel) model.read(in, "http://onto.ui.sav.sk/agents.owl", "RDF/XML");
}
```

Figure 14: Loading the ontology model stored in an .owl File function

```
    function loadOntology() {
        ExtendedIterator<Ontology>  ontology = this.ontologyModel.listOntologies();
        List<Ontology> ontologyList = ontology.toList();

        Iterator ontologieIt = ontologyList.iterator();

        while(ontologieIt.hasNext()){
            Ontology ontologyTemp = (Ontology) ontologieIt.next();
            ontology.add(ontologyTemp);
        }
    }
```

Figure 15: Loading the ontology into the java's structure function

```
function loadAllClasses() {
    loadClasses();
    loadUnionClasses();
    loadIntersectionClasses();
    loadComplementClasses();
    loadEnumerationClasses();
    ...
}
```

Figure 16: Loading the ontology all classes into the java's structure function

```
function loadClasses() {
    ExtendedIterator<OntClass> classe = this.ontologyModel.listClasses();
    List<OntClass> classeList = classe.toList();

    Iterator classeIt = classeList.iterator();

    while(classeIt.hasNext()){
        OntClass classeTemp = (OntClass) classeIt.next();
        this.classes.add(classeTemp);
    }
}
```

Figure 17: Loading the ontology simple classes into the java's structure function

```
function loadUnionClasses() {
    ExtendedIterator<UnionClass> classe = this.ontologyModel.listUnionClasses();
    List<UnionClass> classeList = classe.toList();

    Iterator classeIt = classeList.iterator();

    while(classeIt.hasNext()){
        UnionClass classeTemp = (UnionClass) classeIt.next();
        this.unionClasses.add(classeTemp);
    }
}
```

Figure 18: Loading the ontology union classes into the java's structure function

```
function loadProperties() {
    loadDatatypeProperties();
    loadObjectProperties();
}
```

Figure 19: Loading the ontology dataType and object properties into the java's structure function

```
function loadIndividus() {
    ExtendedIterator<Individual> individus = this.ontologyModel.listIndividuals();
    List<Individual> individusList = individus.toList();

    Iterator individusIt = individusList.iterator();

    while(individusIt.hasNext()){
        Individual individuTemp = (Individual) individusIt.next();
        this.individus.add(individuTemp);
    }
}
```

Figure 20: Loading the ontology individuals into the java's structure function

```
function classesMapper(Vector<OntClass> classes) {
    for(OntClass classe : classes) {
        if(!classe.getSubClass().toString().equals(classe.toString())) { // CLASSE HIERARCHIQUE
            Element classificationScheme = new Element("ClassificationScheme");
            classificationScheme.setAttribute("id", classe.toString());
            for(OntClass classeN2 : classes) {
                if(classeN2.getSuperClass().toString().equals(classe.toString())) {
                    Element classificationNode = new Element("ClassificationNode");
                    classificationNode.setAttribute("id", classeN2.toString()+"-id");
                    classificationNode.setAttribute("code", classeN2.toString());
                    classificationNode.setAttribute("parent", classeN2.getSuperClass().toString());
                    classificationScheme.addContent(classificationNode);
                }
            }
            classificationShemes.add(classificationScheme);
        }
        else { // CLASSE NON HIERARCHIQUE
            Element classificationNode = new Element("ClassificationNode");
            classificationNode.setAttribute("id", classe.toString());
            classificationShemes.add(classificationNode);
        }
    }
}
```

Figure 21: Mapping the ontology classes to the ebXML RIM objects function

```
function propertiesMapper(Vector<DatatypeProperty> datTypeProperies, Vector<ObjectProperty> objectProperties) {
    for(DatatypeProperty dataTypeProperty : datTypeProperies) {
        Element property = new Element("Association");
        property.setAttribute("id", dataTypeProperty.toString());
        property.setAttribute("sourceObject", dataTypeProperty.getDomain().toString());
        property.setAttribute("targetObject", dataTypeProperty.getRange().toString());

        if(dataTypeProperty.isFunctionalProperty())
            property.setAttribute("associationType", "Functional");
        if(dataTypeProperty.isSymmetricProperty())
            property.setAttribute("associationType", "Symetric");
        if(dataTypeProperty.isTransitiveProperty())
            property.setAttribute("associationType", "Transitive");
        else
            property.setAttribute("associationType", "Contains");
        associations.add(property);
    }
    for(ObjectProperty objectProperty : objectProperties) {
        Element property = new Element("Association");
        property.setAttribute("id", objectProperty.toString());
        property.setAttribute("sourceObject", objectProperty.getDomain().toString());
        property.setAttribute("targetObject", objectProperty.getRange().toString());
        associations.add(property);
    }
}
```

Figure 22: Mapping the ontology properties to the ebXML RIM objects function

```
function individualsMapper(Vector<Individual> individus) {
    for(Individual individu : individus) {
        Element individual = new Element(individu.toString());
        for(DatatypeProperty dp : dataTypeProperties) {
            if(dp.getDomain().equals(individu)) {
                Element individuData = new Element(dp.toString());
                individuData.setAttribute("rdf:datatype", dp.getRange().toString());
                individuData.setText(individu.getPropertyValue(dp).toString());

                individual.addContent(individuData);
            }
        }
        individuals.add(individual);
    }
}
```

Figure 23: Mapping the ontology individuals to the ebXML RIM objects function

```
private void organisationMapper(String organisationName, String lang) {
    organization.setAttribute("id", organisationName + " Service provider-id");
    Element organizationName = new Element("name");
    Element localizedStringOrganizationName = new Element("localizedString");
    localizedStringOrganizationName.setAttribute("lang", lang);
    localizedStringOrganizationName.setAttribute("value", organisationName);

    organizationName.addContent(localizedStringOrganizationName);
    organization.setAttribute("id", organisationName + " Service provider-id");

    organization.addContent(organizationName);
}
```

Figure 24: Mapping the organization to the ebXML RIM objects function

```
private void serviceMapper(String serviceId, String serviceName, String servicePath
    , String wsdlPath, String lang) {
    service.setAttribute("id", serviceId);
    Element name = new Element("name");
    Element localizedStringName = new Element("localisezString");
    localizedStringName.setAttribute("lang", lang);
    localizedStringName.setAttribute("value", serviceName + " Registry Service");
    name.addContent(localizedStringName);
    service.addContent(name);
    Element description = new Element("description");
    Element localizedStringDescription = new Element("localisezString");
    localizedStringDescription.setAttribute("lang", lang);
    localizedStringDescription.setAttribute("value", serviceName + " Services");
    description.addContent(localizedStringDescription)
    service.addContent(description);
    Element serviceBinding = new Element("ServiceBinding");
    serviceBinding.setAttribute("id", serviceName+"SOAPBinding");
    serviceBinding.setAttribute("accessURI", servicePath);
    Element specificationLink = new Element("specificationLink");
    specificationLink.setAttribute("id", serviceName+"SOAPBinding SpecificationLink");
    specificationLink.setAttribute("sepecificationObject", wsdlPath);
    Element registryObject = new Element("registryObject");
    registryObject.addContent(organization);
    specificationLink.addContent(registryObject);
    serviceBinding.addContent(specificationLink);
    service.addContent(serviceBinding);
}
```

Figure 25: Mapping the service to the ebXML RIM objects function

Research Article

Background and Related Work

An important trend in the evolution of the World Wide Web (WWW) has been the standardization of the Web Service technologies, by which a service provider can make a service available to consumers through an advertised protocol. The function of a Web Service is often to supply information, but can also involve exchange or sale of goods or obligations. Web Services provide a basis for interoperability between service providers and consumers, based on the reliable exchange of messages.

For the purpose of making the service discovery more easier and automated, many practices has been developed for introducing semantic into the business registries on registries based domains by storing meta-data in it registries. Mapping the domain ontology and using OWL-S (Martin *et al.,* 2004) for constructing the meta-data are the most popular and developed practices.

In this article, we adopt the mapping process for enhancing the discovery of the services by storing semantic constructs resulting from a mapping process of the domain ontology to the ebXML registries RIM specifications.

Conclusion and Outlook to Future Work

Ontologies have been a research topic being addressed in a rather small research community. This has changed drastically in the late nineties by the insight that a conceptual, yet executable model of an application domain provides a significant value. The impact has increased with the Semantic Web initiative and the Web Ontology Language (OWL).

The importance of semantics is also recognized in the Web services area and there have been several efforts to improve the semantics support for Web services.

In this work, we use OWL DL, since we are using ontologies to get knowledge through querying rather than reasoning. We investigate the possible ways of making the registry OWL very rich in semantic and describe an approach that minimizes the changes on the ebXML specification.

After this experience, there is a very important observation, that is, Ontologies in this work, provide a source of shared and precisely defined terms which can be used formalizing knowledge and relationship among objects in a domain of interest, and they are the main subject for representing knowledge in the ebXML registries by mapping it to the class hierarchy and storing it in the registry.

As a future work, we intend to improve the Mapping approach to express more complex OWL classifications and properties on the ebXML registries for more semantic expressiveness, and improve the Query Manager component by making it a real standalone reasoner.

REFERENCES

Akkiraju R, Farrell J & Miller J (2005). *Web Service Semantics WSDL-S* **10(**3) 243-277.

Antonio Pereira A, Cunha F, Malheiro P & Azevedo A (2008). EBXML-Overview, Initiatives and Applications, Innovation in Manufacturing Networks. *IFIP - The International Federation for Information Processing* Volume 266, pp 127-136, 2008

D McGuinness and F Harmelen (2007). OWL web ontology language, Semantic Web: Concepts, Technologies and Applications. *NASA Monographs in Systems and Software Engineering* 2007 (Springer-Verlag London Limited) 81-103.

Dogac A, Cingil I, Laleci GB & Kabak Y (2002). *Importing the Semantic Web in UDDI, Web Services, E-Business, and the Semantic Web* Lecture Notes in Computer Science **2512** 225-236.

G Antoniou & F Harmalen (2004). Web ontology language: OWL, *Handbook on Ontologies* (Springer-Verlag) 76-92.

Gagnon M (2009). *Description Logics, Reasoning Web. Semantic Technologies for Information Systems* Lecture Notes in Computer Science **5689** 1-39.

Gunatilaka M, Wikramanayake G & Karunaratne D (2004). Implementation of Ontology Based Business Registries to Support e-Commerce. *23rd National Information Technology Conference.*

Gunatilaka M, Wikramanayake G & Karunaratne D (2004). Improving B2B Transactions by Exploiting Business Registries. *6th International Information Technology Conference.*

Je Z Pan (2009). Resource Description Framework, *International Handbooks on Information Systems* 2009, pp 71-90, 2009
Ling Liu L & Tamer O (2009). *OASIS, Encyclopedia of Database Systems* 1927.
Martin D, Paolucci M, McIlraith S, Burstein M, McDermott D, McGuinness D, Parsia B, Payne T, Sabou M, Solanki M, Srinivasan N & Sycara K (2004). *Bringing Semantics to Web Services: The OWL-S Approach*, First International Workshop, SWSWPC, (Springer-Verlag Berlin Heidelberg) 26-42.
Mcguinness DL, Fikes R, Hendler J & Stein LA (2001). DAML+OIL: an ontology language for the Semantic Web. *Intelligent Systems, IEEE* **17(**5) 72-80.
Medjahed B & Bouguettaya A (2011). *Service Composition for the Semantic Web* (Springer Science + Business Media, LLC).
Miller E (1998). An Introduction to the Resource Description Framework, *Bulletin of the American Society for Information Science and Technology* **25(**1) 15-19.
Romin I (2002). *An Introduction To ebXML*, Web Services Business Strategies and Architectures 220-235.