Semantic Annotation of XML-Schema for Document Transformations

Julius Köpke and Johann Eder

Semantic Annotation of XML-Schema for Document Transformations

Julius Köpke and Johann Eder

Department of Informatics-Systems, University of Klagenfurt, Austria
firstname.lastname@uni-klu.ac.at
WWW home page: http://isys.uni-klu.ac.at

Abstract. The W3C recommendation Semantic Annotations for WSDL and XML-Schema (SAWSDL [7]) has the primary goal of annotating web service descriptions with a semantic model. In addition to the annotation of WSDL documents it can also be used to annotate arbitrary XML-Schemas. In this paper we will discuss the application of SAWSDL to create declarative annotations of XML-Schema. We will show problems that arise and present a solution that creates SAWSDL compliant annotations with the required expressiveness. Such an annotation method can then be used to assist automatic document transformations between different schemas. The transformations can act as an enabler for interoperable applications that exchange XML-documents.

1 Introduction

Semantic annotation is proposed to be a good solution to enable interoperable applications [11]. Semantic annotations represent the relationships between an annotated artifact (web page, XML document, schema, web service, etc.) and a reference ontology. Semantic annotation at the instance level (web pages, XML-documents, ...) received a lot of attention [11], however annotation at the XML-Schema level [7] is used in a much lesser degree. Semantic annotations of XML-Schema can be used to lift data from XML documents to some semantic representation such as RDF [8] or OWL [3]. A transformation of a document from a source XML-Schema to a document that complies with the target XML-Schema can therefore be created by lifting the data from the source document to it's semantic representation, (e.g. ontology instances) make some computations on the ontology-level and lower it back to the XML representation of the target schema. Such an approach is very flexible but requires the computation of every single instance document on the ontology level. Preliminary tests have shown that even with a very small ontology the speed of a direct XSLT transformation is significantly faster compared to the loading of the instance data, reasoning over the ontology and lowering the data back to the target XML representation. To achieve industry-scale performance we therefore propose to generate XSLT transformation scripts with the knowledge from the ontology. This requires the annotation of the source and the target schema in a declarative way. Such an annotation allows the matching of the source and the target schema at build

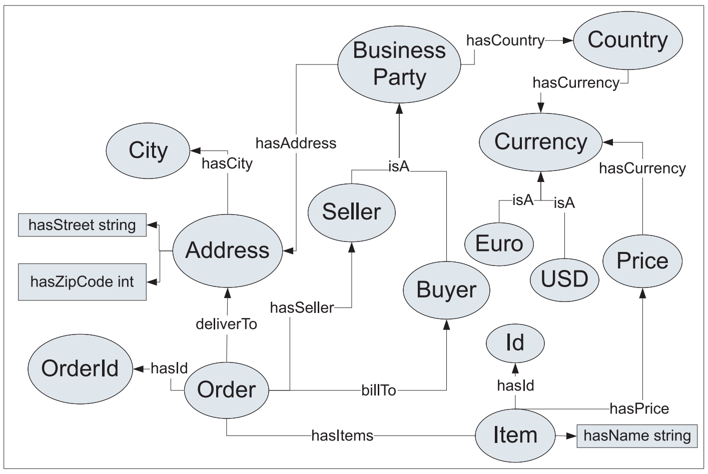


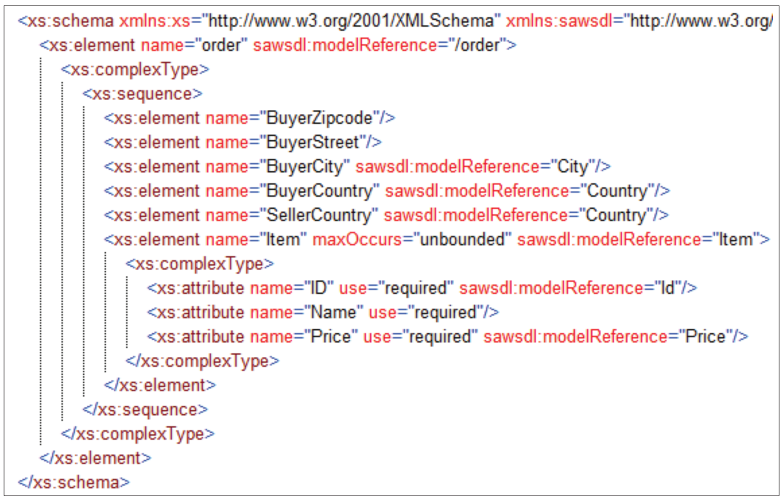
Fig. 1. Example reference ontology

time. This matching can then be used to create transformation scripts [4] (e.g. XSLT) that directly operate on the XML documents without the need to lift instance data to the ontology.

Both approaches are addressed in the W3C recommendation SAWSDL. The lifting and lowering approach is realized by the specification of references to arbitrary scripts that perform the lifting or lowering of instance data. The declarative annotations can be realized by so called *model-references*. A model-reference forms a reference between a schema element (XML-Element, XML-Type or XML-Property declaration) and a concept of some semantic model. In this paper we will investigate the applicability of SAWSDL model-references for the declarative annotation of XML-Schemas with a reference ontology. We discuss shortcomings and present an annotation method that solves these problems while being compatible with SAWSDL. Such an annotation can then be used as a basis for a system that transforms instance documents from an annotated source schema to an annotated target schema.

2 Motivating example

In order to show shortcomings of model-references we will first introduce an example. We will try to use model-references for the direct annotation of a simple XML-Schema of a business document (see figure 2) with a small reference ontology (see figure 1). The domain of a SAWSDL model-reference is an XML-Element, XML-Type or XML-Attribute declaration of an XML-Schema. The range is a list of URIs that point to concepts of a semantic model. If multiple URIs are specified every URI applies to the annotated element. No further relationships between the different URIs can be specified.



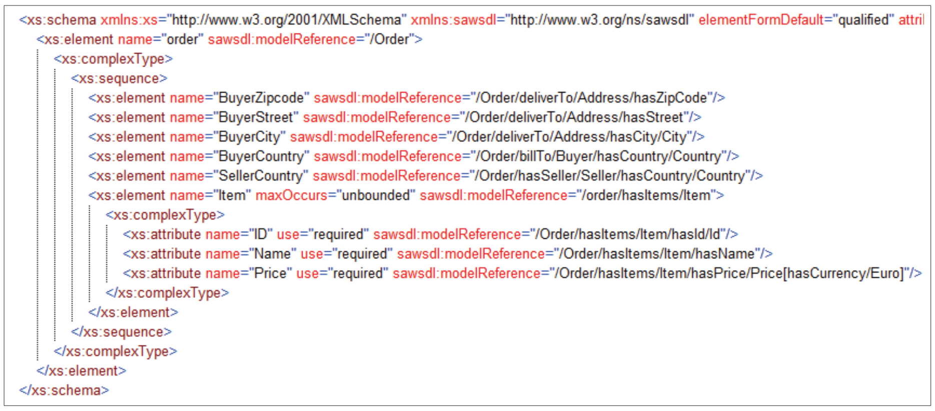
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:sawsdl="http://www.w3.org/
  <xs:element name="order" sawsdl:modelReference="/order">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BuyerZipcode"/>
        <xs:element name="BuyerStreet"/>
        <xs:element name="BuyerCity" sawsdl:modelReference="City"/>
        <xs:element name="BuyerCountry" sawsdl:modelReference="Country"/>
        <xs:element name="SellerCountry" sawsdl:modelReference="Country"/>
        <xs:element name="Item" maxOccurs="unbounded" sawsdl:modelReference="Item">
          <xs:complexType>
            <xs:attribute name="ID" use="required" sawsdl:modelReference="Id"/>
            <xs:attribute name="Name" use="required"/>
            <xs:attribute name="Price" use="required" sawsdl:modelReference="Price"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Fig. 2. Sample XML-Schema with model-references

In figure 2 an example order document is shown. It is directly annotated with the reference ontology (see figure 1). We will now investigate whether the correct meaning of each element can be defined.

– The element *BuyerZipcode* could not be annotated at all because the zip-code is modeled in form of a data-type property and not by a concept in the ontology. The same problem exists for the *BuyerStreet* element and for the name of an item.
– The *BuyerCountry* element is annotated with the concept country. This does not fully express the semantics because we do not know that the element should contain the country of the buying-party. In addition the *Seller-Country* element has exactly the same annotation and can therefore not be distinguished.
– The attribute *Price* is annotated with the concept *Price*. Unfortunately this does not capture the semantics. We do not know the subject of the price (an item) and we do not know the currency.

In the examples above we assumed that we have only annotated data-carrying elements. If we would in addition also annotate the parent elements in this case the *order* element we could add a bit more semantic information. It would be clear that the annotations of the child-elements of the order-element can be seen in the context of an order. Unfortunately this would not help for the ambiguities between the *BuyerCountry*- and the *SellerCountry* element. In general it would require a very strong structural relatedness between the ontology and the annotated XML-Schema which we cannot guarantee when many different schemas are annotated with a single reference ontology. In addition SAWSDL does not define that there are any relations between the annotations of parent and child



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:sawsdl="http://www.w3.org/ns/sawsdl" elementFormDefault="qualified" attril
  <xs:element name="order" sawsdl:modelReference="/Order">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BuyerZipcode" sawsdl:modelReference="/Order/deliverTo/Address/hasZipCode"/>
        <xs:element name="BuyerStreet" sawsdl:modelReference="/Order/deliverTo/Address/hasStreet"/>
        <xs:element name="BuyerCity" sawsdl:modelReference="/Order/deliverTo/Address/hasCity/City"/>
        <xs:element name="BuyerCountry" sawsdl:modelReference="/Order/billTo/Buyer/hasCountry/Country"/>
        <xs:element name="SellerCountry" sawsdl:modelReference="/Order/hasSeller/Seller/hasCountry/Country"/>
        <xs:element name="Item" maxOccurs="unbounded" sawsdl:modelReference="/order/hasItems/Item">
          <xs:complexType>
            <xs:attribute name="ID" use="required" sawsdl:modelReference="/Order/hasItems/Item/hasId/Id"/>
            <xs:attribute name="Name" use="required" sawsdl:modelReference="/Order/hasItems/Item/hasName"/>
            <xs:attribute name="Price" use="required" sawsdl:modelReference="/Order/hasItems/Item/hasPrice/Price[hasCurrency/Euro]"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Fig. 3. Sample XML-Schema document with proposed annotation method

elements. Nevertheless such annotations could help to give additional knowledge to structural XML-matching methods such as [10]. Another solution would be the usage of a more specific reference ontology, which contains concepts that fully match the semantics of each annotated element. For example it would need to contain the concept *InvoiceBuyerCountry* and *InvoiceBuyerZipCode*. Enhancing a general reference ontology with all possible combinations of concepts leads to a combinatorial explosion. This is definitely not suitable for the annotation with a general reference ontology but can nevertheless be used if very specific ontologies are used for the annotation.

3 Annotation Method

As shown in the introduction the direct usage of model-references is not suitable for the direct annotation of an XML-Schema with a reference ontology. To overcome this shortcoming we propose to create the required more specific ontology concepts out of well defined path expressions at runtime of the schema-matching engine. We will first introduce the path expressions with some examples and provide the formal definitions in section 3.1. The example schema document in figure 3 is annotated with the proposed path expressions. We will discuss some examples: The element *BuyerZipcode* is annotated with /***Order***/*deliverTo*/***Address*** /*hasZipCode*. The annotation of the *BuyerCountry* element is /***Order***/*billTo*/ ***Buyer***/*hasCountry*/***Country***. The steps that are marked bold refer to concepts. The other steps refer to object-properties or data-type properties of the reference ontology. Now the *BuyerCountry* element can be clearly distinguished from the *SellerCountry* element and the elements *BuyerZipcode* and *BuyerStreet* can be annotated. The shown paths refer to concepts, object properties and data-type properties. Another requirement could be to address instances of the ontology. For example the path /***Order***/*billTo*/***Buyer***[*Mr_Smith*]/*hasCountry*/***Country*** defines that the Buyer is restricted to one specific buyer with the URI *Mr_Simth*.

In most cases we assume that a simple annotation path as shown in the examples above is sufficient for an annotation. Nevertheless there can be cases where additional restrictions are required: When using a simple path expressions as shown above the *Price* attribute of the example schema could be annotated with */**Order**/hasitems/**Item**/hasPrice/**Price***. Unfortunately this does not express the currency of the price. Since the example ontology has no specialized price-concept for each currency we need to define the price within the annotation. The correct currency of a price can be defined by a restriction of the price concept. This restriction is denoted in square brackets and expresses that the price must have a *hasCurrency* property that points to the concept *Euro*. This leads to the full annotation of the *Price* attribute: */**Order**/hasitems/**Item**/hasPrice/ **Price**[hasCurrency/**Euro**]*.

3.1 Formal definition of the annotation method

In order to define the annotation method we will first introduce definitions for the reference ontology and an annotated schema.

Definition 1 ***Ontology:***

An ontology O is a tuple $O = (C, DP, OP, I, R)$, where C is a set of concepts (also often referred as classes), DP as set of data-type-properties, OP a set of object-properties, I a set of individuals and R a set of restrictions. Each element in C, DP and OP is a tuple $(uri, definition)$. All URIs of concepts can be obtained by $C.uri$, $URIs$ of properties by $DP.uri$ and $OP.uri$ and URIs of instances by $I.uri$ respectively.

Definition 2 ***Annotated XML-Schema:***

An annotated XML-Schema S is a tuple $S = (T, E, A)$, where T is a set of types, E a set of elements, and A is a set of semantic annotations. An XML-Schema forms a tree structure. Each $t \in T$ has a type $e.type = \{simple \mid complex\}$, an optional name $t.name$ and an optional SAWSDL mode reference $t.annotation \in A$. An element $e \in E$ can have an optional type $e.type \in T$, an optional SAWSDL model-reference $e.annotation \in A$ and a set of attributes $e.attribute$. Each attribute $a \in e.attribute$ can have a simple type $a.type \in T$ and an optional model-reference $a.annotation \in A$. Each annotation must be a valid annotation path according to definition 3 and 4.

Definition 3 ***Annotation Path:***

The set of all annotation path expressions is P. An annotation path $p \in P$ is a sequence of steps. Each step is a tuple s=(*uri*, *type*, *res*). The value *s.uri* of a step is some URI of an element of the reference ontology O. The type *s.type* can be *cs* for a concept-step, *op* for an object-property step or *dp* for a data-type-property step. The URI *s.uri* determines the type of the step: $s.uri \in C.uri \Rightarrow s.type = cs$; $s.uri \in OP.uri \Rightarrow s.type = op$; $s.uri \in DP.uri \Rightarrow s.type = dp$.

Only concept-steps may have a set of restrictions $s.res$. Each restriction $\in s.res$ can either be an individual $\in I.uri$ or a restricting path expression. Such a path expression adds a restriction to the corresponding step s. If $s.res$ contains multiple restrictions they all apply to the corresponding step s (logical and). The succeeding step of s in p can be obtained by $s.succ$, the previous step by $s.prev$. The first step of p is denoted f_s and the last step l_s.

Definition 4 ***An annotation path is structurally valid iff:***

– $f_s.type = cs$ - The first step must refer to a concept.
– $l_s.type = \{dp|cs\}$ - The last step must refer to a concept or a data-type property.
– $\forall s \in p|s.type = cs \wedge s \neq l_s \Rightarrow s.succ.type = \{dp|op\}$ The successor of a concept-step must be an object-property or data-type-property step.
– $\forall s \in p|s.type = op \Rightarrow s.succ.type = \{cs\}$ An object-property step must be followed by a concept-step.
– $\forall s \in p|s.type = cs \wedge s \neq f_s \Rightarrow s.prev.type = op$ The previous step of a concept-step must be an object-property step (except the fist step).
– $\forall s \in p|s.type = op \Rightarrow s.prev.type = cs$ The previous step of an object-property step must be a concept-step.
– $\forall s \in p|s.type = dp \Rightarrow s = l_s$ Only the last step can refer to a data-type property.

3.2 Reuse of global types or elements

If the annotated XML-Schema reuses types or elements (via type or ref properties) and both the element and the referenced element or type are annotated then the semantics need to be constructed based on the annotation of the element and the annotation of the referenced element. Due to the hierarchical structure of XML this needs to be applied recursively.
Let e be an element with the annotation $e.annotation$ and the XML-Type $e.type$. Let s be an annotated sub-element of the XML-Type $e.type$. Then the complete path of s needs to be constructed by combining the annotation $e.annotation$ and $s.annotation$. In particular the combination is achieved by removing the last step of $e.annotation$ and concatenating it with $s.annotation$.
As an example we may have an XML-element called *DeliveryAddress*. It is itself annotated with the annotation path */**Order**/deliverTo/**Address***. It has a type definition *address*. The address type itself contains various elements. One of them is *street* which is annotated with */**Address**/hasStreet*. In order to construct the complete semantics of the *street* element that has the parent element *DeliveryAddress* we need to build the path */**Order**/deliverTo/**Address**/hasStreet*. This path combination needs to be performed by the schema matching engine. It must be noted that this path combination adds structural dependencies between the schema and the reference ontology. Therefore one XML-Type should only be reused for semantically related entities. This approach is a specialization of the SAWSDL model reference propagation.

4 Transformation of an Annotation Path to an Ontology Concept

In the last section we have defined an annotation path expression as a sequence of steps. In order to specify the semantics of such a path expression it has to be represented as an ontology concept. This allows concept-level reasoning over the annotated elements in order to assist the matching of schema elements. The name/URI of such a concept is the corresponding path expression and can therefore directly be used as a SAWSDL model-reference. OWL defines concepts with logical expressions in form of restrictions over it's individuals. We will illustrate the generation of concepts with an example.

```
Class: Order/billTo/Buyer[Mr_Smith]/hasCountry/Country
EquivalentClasses(
     Country and inv
       (hasCountry) some
          (Buyer and {Mr_Smith} and inv (billTo) some (Order)
       ))
```

Listing 1.1. Representation of an annotation path in OWL

In listing 1.1 the OWL representation of the path */**Order**/billTo/**Buyer** [Mr_Smith]/hasCountry/**Country*** is depicted. It creates a specialization of a *country* concept. In particular a *country* that has an inverse *hasCountry* object-property to a *Buyer*. This buyer must be an individual of the enumerated class {Mr_Smith} and must have an inverse *billTo* relation to an *Order*.
Obviously such a translation can be achieved fully automatically by iterating over the steps of the path. The generation of the concepts can be realized by the schema-matching engine. The generated concepts are only required as long as a matching is created.
In general a standard annotation path as shown in the example always creates a sub-concept of the last concept-step. Therefore the inverse of the object properties must be used. Annotation paths p that are used in restrictions $p \in s.res$ of some concept-step s always create sub-concepts of the corresponding concept with the URI $s.uri$. Thus the object-properties can directly be used.

5 Validation of Annotations

In the last sections we have defined the structure of an annotation path and have shown how an annotation path can be transformed to an OWL concept. This does not guarantee that the generated concepts do not introduce contradictions to the ontology. As an example we may have a path:
*/**Order**/deliverTo/**PoBox*** and the ontology defines that the *deliverTo* may never point to a post office box. When this path is represented as an OWL concept it can never contain individuals and thus introduces contradictions to the ontology.
In addition the ontology may contain data-type restrictions that restrict values of

data-types to specific types such as string or integer. If such restrictions exist in the ontology they must also exist in the schema. The constraints in the schema must be at least as restrictive as in the ontology. Another type of restriction that may occur in the ontology are cardinality restrictions. For example the ontology may define that an invoice must have a maximum of one order address $hasAddress$. The schema must then also restrict the max-occurs value of the corresponding element. Due to the Open World Assumption of OWL it is not needed to check if an element occurs often enough but it must be ensured that an element can not occur too often in the schema.

The considerations above lead to the definition of the consistency of an annotated schema:

Definition 5 *A schema S and a set of annotation paths P are consistent with an ontology O iff:*

1. Every annotation path $p \in \mathrm{P}$ is structurally valid (see definition 4).
2. Every annotation path $p \in \mathrm{P}$ can be expressed as an OWL-concept in O that is a sub-concept of OWL-Thing.
3. All annotated elements in S are more ore equally constraining the values as the corresponding data-type properties in O do.
4. No cardinality restrictions in O are violated by S.

Obviously these requirements can be checked fully-automatically: The first check can be realized on the structural level. Each referenced concept and property must be a concept or property of the reference ontology and the restrictions from definition 4 must not be violated. The second check is a typical reasoning task that can be done by any OWL reasoner. Checks 3 and 4 can be realized by traversing the schema and querying the restrictions from the ontology.

6 Related Work

In contrast to the annotation of web resources there is only a small number of related work in the field of XML-Schema annotation. To the best of our knowledge there is no comparable approach for a concrete and formal definition of declarative semantic annotations for XML-Schema that allows class-level reasoning over the annotated elements. We see the application of the annotations in the possibility to create more precise schema matchings than traditional structure based approaches [10]. As soon as a matching can be found transformations can be created that transform the actual documents on the XML-level [4]. In [2] an annotation method for RDFS-Schemata is presented. The annotation method is similar to our's as it also expresses the annotations as simple paths that are transformed to ontology concepts. In contrast to our approach it directly supports the definition of operators like split or join in order to allow the annotation of elements which carry data that needs to be transformed before they can be linked to the ontology, which we plan to realize in an additional ontology layer. In contrast to our approach it is not formally described, it does not directly

address XML-Schema and it does not allow direct document transformations on the XML-layer. In [1] another approach for the annotation of models is proposed. At first a meta-model that is expressed in OWL is created for every type of model (Relational database, XML-Schema, ...) which should be annotated. Afterwards an individual for a specific schema is created. This means there is a representation of the concrete schema as an instance in the ontology. Annotations are just mappings between the reference ontology and the individuals of the schema. This solution is therefore not based on direct XML-level annotations as proposed by SAWSDL.
In [12] an approach is presented that automatically discovers mappings between XML-Schemas and ontologies with the help of a given set of simple correspondences between the schema and the ontology. It assumes a structural relatedness and the discovered mappings are expressed in form of rules. Since first order logic rules can only modify instances this approach is well suited for a lifting approach that transforms XML-Data to ontology instances. In contrast our method creates ontology concepts that form declarative descriptions which are a basis to build XML-level transformations without the need of lifting instance data to the ontology at runtime. In [5] the differences between ontologies and XML-Schemas are discussed. The authors propose to model the domain via an ontology and transform this specification to an XML-Schema or database schema. In [6] a system is proposed that automatically creates annotations for Web Service descriptions. It can use a reference ontology but does not need one. If no reference ontology is provided the ontology is created during the approach. The provided annotations are basically enhancements of the schema-elements with vocabulary of the ontology. They do not provide a complete declarative description. Nevertheless approaches that automatically generate annotations like [12] or [6] can possibly be a basis to semi-automatically create annotations for our annotation method.

7 Conclusion

In this paper we have proposed a method for the declarative semantic annotation of XML-Schema that enhances the semantic expressiveness of SAWSDL-model-references. The annotation method has two representations. On the XML-level there are well-defined annotation paths that can be added to XML-Schemas by schema designers without deep ontology engineering skills. These annotation paths can automatically be transformed to ontology concepts. These concepts provide a declarative description of the annotated elements and can be used for class-level reasoning over schema elements in order to create mappings between XML-schemas. These mappings can then be used for the generation of scripts that transform instance-data from one schema to another. We are currently working on a prototype that realizes these transformations. It must be noted that our annotation method does not directly resolve all possible conflicts [9] between the schema and the reference ontology. For example if the attribute granularity of the ontology and the schema differs no direct annotation is possible. We plan to solve such heterogeneities with an additional ontology layer that also adds knowledge for explicitly defined transformations. In our scenario the actual transformation of instance documents takes place on the XML-level without the need to interact with the ontology and is therefore well-suited for applications with a huge amount of instance documents. In addition to our annotation method we have provided mechanisms to check whether the annotations are valid with regard to the ontology.

References

1. Domenico Beneventano, Sabina El Haoum, and Daniele Montanari. Mapping of heterogeneous schemata, business structures, and terminologies. In *DEXA '07: Proceedings of the 18th International Conference on Database and Expert Systems Applications*, pages 412–418, Washington, DC, USA, 2007. IEEE Computer Society.
2. Giorgio Callegari, Michele Missikoff, Osimi M, and Francesco Taglino. Semantic annotation language and tool for information and business processes - appendix f: User manual, athena deliverable d.a3.3 available at the leks (laboratory for enterprise knoweldge and systems) web site http://leks-pub.iasi.cnr.it/astar/. Technical report.
3. Mike Dean and Guus Schreiber. OWL web ontology language reference. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/.
4. Haifeng Jiang, Howard Ho, Lucian Popa, and Wook-Shin Han. Mapping-driven xml transformation. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1063–1072, New York, NY, USA, 2007. ACM.
5. Michael Klein, Dieter Fensel, van Harmelen Frank, and Ian Horrocks. The relation between ontologies and xml schemata. In *Workshop on Applications of Ontologies and Problem-Solving Methods*, August 2000.
6. Peep Küngas and Marlon Dumas. Cost-effective semantic annotation of xml schemas and web service interfaces. *Services Computing, IEEE International Conference on*, 0:372–379, 2009.
7. Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6):60–67, 2007.
8. Eric Miller and Frank Manola. RDF primer. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.
9. Michele Missikoff and Francesco Taglino. Semantic mismatches hampering data exchange between heterogeneous web services. In *W3C Workshop on Frameworks for Semantics in Web Services*, 2005.
10. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
11. Victoria Uren, Philipp Cimiano, José Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegnac. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(1):14–28, 2006.
12. An Yuan, Alex Borgida, and John Mylopoulos. Discovering and Maintatining Semantic Mappings between XML Schemas and Ontologies. *Journal of Computer Science and Engeneering*, 5:1–29, December 2007.