Enabling Semantic Mediation for Business Applications: XML-RDF, RDF-XML and XSD-RDFS transformations

I. Miletic[1,2], M. Vujasinovic[1,2], N. Ivezic[1], and Z. Marjanovic[2]
[1] Manufacturing Engineering Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
{igor.miletic, marko.vujasinovic, nenad.ivezic}@nist.gov
[2] Faculty of Organizational Sciences, University of Belgrade, Jove Ilica 154, 11000 Belgrade, Serbia
marjanovic.zoran@fon.bg.ac.yu

Keywords: transformation, xml, rdf, semantic interoperability, business applications

Abstract

In this paper, we describe results of our work in developing transformation tools that enable use of RDF-based Semantic Mediation tools for integration of business applications that have implemented XML Schema-based interfaces. Specifically, we are concerned with validating advanced Semantic Mediation solutions that required XML to RDF, RDF to XML, and the XML Schema to the RDF Schema transformations to be used by these business applications. To use the advanced integration solutions we developed all three of these transformations. We discuss the requirements for the transformation tool posed by the Semantic Mediation approach in the context of a business-to-business scenario. We analyze the related work and various related approaches. We describe our implemented transformation approach and explain the intended use of the resulting tool within a typical usage scenario.

1 Introduction

A number of approaches exist today to perform transformations from the XML [1] to the RDF format [2] and back, as well as the transformation from the XML Schema [3, 5] to the RDF Schema [4]. These transformations have been the subject of a number of previous publications [12, 13, 14, 16, 17, 18, 19]. In this paper, we describe results of our work in the business-to-business enterprise applications

integration area that required all three of these transformations – XML to RDF, RDF to XML, and the XML Schema to the RDF Schema – to use advanced integration solutions.

In our work, we are concerned with validating advanced Semantic Mediation solutions developed within the ATHENA (Advanced Technologies for Interoperability of Heterogeneous Networks and their Applications) Integrated Project [11]. To use the ATHENA Semantic Mediation tools, the interoperating applications need to represent semantically their documents using the RDF Schema and be able to generate RDF document instances. Presently, many legacy systems have already developed XML interfaces in support of B2B (business-to-business) integration scenarios, but very few of them have implemented an RDF(S) interface. One objective of our work is to develop a tool that will help the business community to utilize easily the Semantic Mediation approach developed by the ATHENA community. To accomplish this, we require a tool that can easily transform XML documents into semantic-based (i.e., RDF-based) documents to help utilize the Semantic Mediation tools. In this way, we open a door to using the emerging Semantic Web technologies for business application integration.

In the rest of the paper, we start with the requirements for the transformation tool posed by the ATHENA Semantic Mediation approach in the context of a business-to-business scenario. Then, we analyze the related work and various existing approaches. Next, we describe our implemented transformation approach and explain the intended use of the resulting tool in a usage scenario. Finally, we outline future work and give concluding remarks.

2 Requirements

To use the ATHENA Semantic Mediation approach in a B2B message exchange, the transformation tool needs to provide three main functions: XML Schema to RDF Schema (XSD2RDFS)[1] transformation, XML to RDF transformation (XML2RDF), and RDF to XML (RDF2XML) transformation. The tool needs to fulfill requirements that we have identified in Table 1 and Table 2.

Table 1. Transformation tool requirements specification

1. XSD2RDFS requirements	a) XSD2RDFS component has to be able to transform any given XML Schema. b) If XML Schema imports or includes another XML Schema, the tool has to provide the importing schema functionality. For efficiency reasons, the tool has to transform only necessary (minimal) set of elements that appears in XML Schemas.
2. XML2RDF requirements	a) XML2RDF tool has to be able to transform an XML document into an RDF document without loosing any information. b) XML2RDF has to do transformation without relying on the XML Schema during transformation.

[1] The XML Schema language is also referred to as XML Schema Definition (XSD).

	c) Attained RDF instance has to be valid against the RDF Schema that is obtained from XSD2RDFS transformation.
3. RDF2XML requirements	a) RDF2XML transformation has to transform an RDF instance (output form the ATHENA runtime tool) into an XML instance. b) Output XML instance has to be valid against the XML Schema.

Table 2. Additional limitations on the RDFS documents imposed by the ATHENA tools

List of supported RDFS constructs	**List of unsupported RDFS constructs**
rdfs:Resource, rdfs:Class, *rdf:Property, rdfs:range,* *rdfs:domain, rdf:type,* *rdfs:subClassOf, rdfs:comment* *rdf:Bag*: It has to be used as range of a property for representing an enumeration set. *rdf:Statement, rdf:subject,rdf:predicate, rdf:object, rdf:value, rdfs:seeAlso, rdfs:isDefinedBy,* XMLSchema Datatypes: a subset of them is supported*: xsd:string, xsd:integer, xsd:float, xsd:double, xsd:boolean*	a) *rdfs:subPropertyOf* b) *rdf:label (Currently, a Class or a Property is visualized by showing the ID)* c) *rdf:Alt and rdf:Seq* d) *rdf:List* e) *rdf:first, rdf:rest, rdf:nil* f) *rdfs:Literal* g) *rdfs:Datatype* h) *rdf:XMLLiteral* i) *rdfs:Container,* j) *rdfs:ContainerMembershipProperty* k) *rdfs:member*

In the following section, we analyze relevant efforts and existing tools that could potentially help us in developing our transformation tool.

3 Related work

By analyzing the current state of the art and practice, we have identified three main approaches to XML2RDF, RDF2XML, and XSD2RDFS transformations. This categorization is made according to the rules needed to transform one document structure into another.

1. Concept approach – an XML element becomes an RDFS class (e.g., [13]).
2. Relation approach – an XML element or attribute becomes an RDF property. Depending whether an element has sub-elements and/or attributes, or whether the element contains only data-type values, the element becomes object property or data property (e.g., [12,14]). The attributes become datatype properties.
3. Model approach – depending on the XML Schema definition, an element can become an RDFS class or an object/data-type property. Attributes always become RDF properties (e.g., [17, 18]).

We can consider the first two categories to define structure-mapping approaches. The third category defines model-mapping approaches. A structure-mapping approach maps one type of structure to another type (e.g., a tree into a graph). In this case, it is often not critical to lose some information (such as cardinality of a relation, or part-of relations, or has relations). When using a model-mapping approach, we care about semantics of a data model and the goal is to precisely describe semantics of an XML structure. However, when the objective is to integrate different data structures while delaying definition or resolution of data semantics, then it may be advantegous to take a structure-mapping approach.

There are several efforts and tools that tackle the whole or a part of the X2R2X[2] transformation. Some deal with simple XML instance to RDF instance transformation [13, 16, 20], while others try to resolve both XML2RDF and XSD2RDFS transformations [14, 17, 18, 19, 20]. One approach explains how to get XML from RDF [19].

Battle describes the Gloze approach that uses an XML Schema to describe how XML is mapped into RDF and back again [12]. The Gloze approach follows the rule that every element and attribute maps to an RDF property. Gloze interprets the XML structure as a relational model between parent nodes and their children [12]. This was adopted from Traustor et. al [14]. Both of these solutions fail to satisfy requirement 2.b from our requirements table. Further, Gloze doesn't include transformation from the XML Schema into RDFS (requirements 1.a and 1.b); consequently, the requirement 2.c is not satisfied either. Traustor's approach fails requirement 2.c. as well.

An alternative approach to describe structure-mapping relation approaches has been proposed by Melnik [13]. In particular, he has suggested using element names to classify the content of the element and only attributes to be identified as RDF properties.

Several existing tools use XSLT [23] to achieve desired transformations. One XML2RDF transformation type was, however, specific to an Amazon Web Services implementation [21]. Another interesting approach is proposed by Hannes and Soren that assumes the schema is available during transformation [17]. Even if the XML Schema is not present during the XML2RDF transformation, their tool will generate automatically a new XML Schema and finish transformation based on the generated XML Schema. It is obvious that the result from the RDF2XML will not satisfy requirement 2.c. Eric Miller in [19] explains how to develop XSLT to transform XML to RDF.

We have identified several XSD2OWL implementations that can be easily adopted for an XSD2RDFS transformation. The OWL language is based on RDFS and it is easy to implement steps to assure the resulting OWL model is a valid RDFS model with some semantic information loss. Classes, object properties, and datatype property relations may be readily retained in this process. Hannes and Soren tackle XSD2OWL transformation in their paper by providing transformation rules [17]. Their solution, however, does not resolve the naming collision issue (i.e., it does not define a naming convention for elements/attributes with the same name within an XML Schema). Hence, the requirement 1.a isn't satisfied.

[2] We will use X2R2X to reference both the bidirectional transformation between XML and RDF as well as the XSD2RDFS transformation.

Backward transformation from RDF2XML is not supplied at all. A similar approach was given by Anicic et. al with a focus on transformation rules to extract all semantics represented in the XML Schema and to formalize the semantics in the OWL format by using the superimposed meta-model [18]. The current implementation of their work does not support XML Schema inner complex/simple type constructs and does not have an appropriately defined naming convention to resolve the naming collision problem. We tried to adopt this solution by converting inner complex/simple type to global complex/simple type but then we experienced problems with renaming/restructuring XML instance elements prior to XML2RDF instance transformation. In that case, XSD is required during XML2RDF transformation and the tool needs to 'adopt' an XML instance to be valid against 'the adopted' XML Schema. Rules for inverse transformation haven't been defined. Hence, Anicic's approach doesn't satisfy requirements 1.a, 2.b, 3.a and 3.c.

Garcia and Celma developed their ReDeFer approach that combines a transformation from the XML Schema to the web ontology language (OWL) with a transparent transformation from XML to RDF [20]. The ontologies generated by XSD2OWL are used during the XML to RDF transformation step in order to generate semantic metadata that makes XML Schema semantics explicit [20]. They have stressed that the only adjustment done to the automatically generated ontology is to resolve a name collision between OWL classes and RDF properties. This approach does not support XML Schema inner complex/simple type constructs and does not address the naming collision problem. Consequently, the ReDeFer does not meet requirements 1.a, 2.b, 3.a, 3.c.

Following the analysis of the related work, we decided to develop a new X2R2X tool following the structure-mapping approach. Following this approach, all ATHENA requirements will be satisfied and RDF(S) can be generated easily. Our work leans towards the approach proposed by Melnik [15].

4 X2R2X Approach

The goal of our approach was to define rules that will guarantee that any XML Schema will be transformed to an appropriate RDF Schema; that any XML document will be transformed into a valid RDF document; and that any RDF document will be transformed into an XML document conformant with the original XML Schema. The defined rules must provide a mechanism that will maintain structure of the document format throughout the transformation so that the information exchange is supported succesfully by the ATHENA tools.

According to the requirements described in Section 2, we had to focus on the required output from the XML2RDF transformation. By observing the XML Schema and XML instance documents, we noticed that only XML Schema constructs, such as *xsd:element* and *xsd:attribute*, occur in the XML instance document. In other words, the XML instance document is composed only of elements and attributes. This is similar to the observation made by Melnik [13].

Before we transform an XML Schema into an RDFS, we build an internal representation of the XML Schema. This internal representation we call "*internal*

tree". This representation is an XML document that contains only necessary elements and attributes from the XML Schema organized in a tree structure. The *internal tree* is an XML document where every node represents an element or an attribute that appears in the XML instance. The node encapsulates information such as name, type, extension, restriction, and namespace. Our rules transform the *internal tree* into RDFS. The rules that we have defined apply to any existing construct in an *internal tree* and build a corresponding RDFS construct. Also, the rules follow naming convention to generate names of the RDFS constructs.

Building an internal tree

To build an internal tree, we found that it is necessary to develop a mechanism that will be able to find any element or attribute inside an XML Schema. Often, an XML Schema imports or includes one or more additional XML Schemas. Frequently, elements are nested inside of some *"ComplexType"* or they are defined in different XML Schemas and in different namespaces. In our approach, we load all imported and included schemas and search for definition of a given element or attribute. Using this search mechanism, we are able to find any element or attribute defined anywhere in the imported or included schemas. After we find elements or attributes we insert them into the appropriate place within the internal tree.

```
<xsd:element name="Kanban">
  <xsd:complexType>
    <xsd:complexContent>
     <xsd:extension base="xsd:string">
      <xsd:sequence>
       <xsd:element name="Size">
        <xsd:simpleType>
         <xsd:restriction base="xsd:int">
          <xsd:enumeration value="5" />
          <xsd:enumeration value="10" />
         </xsd:restriction>
        </xsd:simpleType>
       </xsd:element>
      </xsd:sequence>
     <xsd:attribute name="type"
                    type="xsd:string" />
     </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="Kanban"
                extension ="xsd:string"
namespace="http://www.nist.gov/kanban#">
    <xsd:element name="Size"
                restriction ="xsd:int"
namespace="http://www.nist.gov/kanban#">
            <xsd:enumeration value="5"/>
            <xsd:enumeration value="10"/>
    </xsd:element>
    <xsd:attribute name="type"
                    type="xsd:string"
namespace="http://www.nist.gov/kanban#/>
</xsd:element>
```

Figure 1. Example of the XML Schema element and its internal tree representation

The *internal tree* keeps the structure of XML instance document which, is an instance of the transformed XML Schema. As we can see from the example in Figure 1, we keep information only about elements, attributes and their types within the *internal tree*. Also, we add namespaces to every node in the *internal tree*.

Generating RDFS from an internal tree

Our approach doesn't require the XML Schema to be present when transforming an XML instance into an RDF instance or when transforming an RDF instance into an XML instance. To provide this functionality, we introduce a naming convention that we follow when defining the transformation rules. The naming convention used to create the transformation rules guarantees that the transformed documents will be valid. We defined the transformation rules to provide a unique way to serialize an *internal tree* into an RDFS. Table 3 describes the rules and naming conventions.

Table 3. Transformation rules and naming convention

Internal tree	**RDFS**	**Description and naming convention**
Nodes		
xsd:element	*rdfs:Class*	Every element becomes *rdfs:Class*. Name of the class depends on the position of the element.
xsd:attribute	*rdf:Property*	Every attribute becomes *rdf:Property*. Domain of the property is *rdfs:Class* that represents parent element of the attribute in the internal tree. Range of the property is one of the XML Schema datatypes.
xsd:enumeration	*rdf:Bag*	The *rdfs:Bag* contains enumerated values.
Parent-child relation between two elements	*rdf:Property*	This relation becomes a property where domain of the property is *rdfs:Class* that represents the parent element and range of the property is *rdfs:Class* that represents the child element. Name of the property is created following the next pattern: parentName_childName_PROP.
Attributes		
name/ref	*rdf:about*	Attributes *name* or *ref* of <*xsd:element*> are used to create a unique name of *the rdfs:Class*. Name of the *rdfs:Class* is composed from concated names of elements found in the branch from the *internal tree* where transformed element is coming from. If attribute *name* is a part of <*xsd:attribute*> then it is used to create name of the *rdf:Property* which is represented using the *rdf:about* construct.
type	*rdf:Property* and *rdf:range*	If *type* appears in the node, which is an <*xsd:element*>, then we create *rdf:Property* that contains postfix "_sValue". Range of this property is the value of the attribute *type.* Domain of the property is *rdfs:Class* that represents the element that contains the *type* attribute. In case where *type* appears in the node which is <*xsd:attribute*>, then domain of the property is *rdfs:Class* that represents parent element of the attribute node.

extension or *restriction*	*rdf:Property*	We create *rdf:Property*, which name contains postfix "_sValue". Range of this property is a value of the *extension* or *the restriction* attribute. Domain of the property is *rdfs:Class* that represents element that contains the *extension* or *the restriction* attribute.
namespace	*namespace*	We add namespace to every *rdfs:Class* and *rdf:Property* we create.

We want to emphasize that we ignore some of the XML Schema concepts such as minOccurs, maxOccurs, and pattern because they are not relevant to RDFS in the context of the ATHENA requirements. RDFS in the ATHENA Semantic Mediation context is used to model structure of a data set, but not to manipulate the structure for semantic purpose (e.g., for purposes of automated inferencing).

According to the transformation rules and the naming convention, the RDFS file for the example given in Figure 1 will look like as shown in Figure 2.

```
<rdfs:Class rdf:about="http://www.nist.gov/kanban#Kanban"/>
<rdfs:Class rdf:about="http://www.nist.gov/kanban#Kanban_Size"/>
<rdf:Property rdf:about="http://www.nist.gov/kanban#Kanban_Size_PROP">
        <rdfs:domain rdf:resource="http://www.nist.gov/kanban#Kanban"/>
        <rdfs:range
rdf:resource="http://www.nist.gov/kanban#Kanban_Size"/></rdf:Property>
<rdf:Property rdf:about="http://www.nist.gov/kanban#Kanban_sValue">
        <rdfs:domain rdf:resource="http://www.nist.gov/kanban#Kanban"/>
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/></rdf:Property>
<rdf:Property rdf:about="http://www.nist.gov/kanban#Kanban_type_attr">
        <rdfs:domain rdf:resource="http://www.nist.gov/kanban#Kanban"/>
        <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/></rdf:Property>
<rdf:Property rdf:about="http://www.nist.gov/kanban#Kanban_Size_sValue">
        <rdfs:domain rdf:resource="http://www.nist.gov/kanban#Kanban_Size"/>
        <rdfs:range><rdfs:Bag>
        <rdfs:li rdf:datatype="http://www.w3.org/2001/XMLSchema#int">5</rdfs:li>
        <rdfs:li rdf:datatype="http://www.w3.org/2001/XMLSchema#int">10</rdfs:li>
        </rdfs:Bag><rdfs:range>
</rdf:Property>
```

Figure 2. Part of the RDFS document attained after XSD2RDFS transformation

XML2RDF and RDF2XML transformation

We defined the transformation rules and special naming convention to simplify runtime transformations so that we don't have to reference the XML Schema or RDF Schema; yet, we can generate XML and RDF instances that are valid against their respective design-time models (i.e., schemas). At runtime, we load an instance document into the transformation tool and follow the naming convention rules to create an output file. The output file depends on the transformation we are executing. If we transform an XML instance, the output is a set of RDF

individuals. In the opposite direction, if we transform a set of RDF individuals, the output is an XML instance. Our very simple instance transformation, in the context of the previous example, would look as follows:

```
<nist:Kanban type="PickUp">TestData
  <nist:KanbanSize>5</KanbanSize>
</nist:Kanban>
```

Figure 3. Example of the XML instance document

This XML instance transformed by our transformation tool will give the following output:

```
<nist:Kanban_Size rdf:ID="Kanban_Size_1">
        <nist:Kanban_Size_sValue
                        rdf:datatype="http://www.w3.org/2001/XMLSchema#string">5
        </nist:Kanban_Size_sValue>
</nist:Kanban_Size>
<nist:Kanban rdf:ID="Kanban_1">
        <nist:Kanban_sValue
                rdf:datatype="http://www.w3.org/2001/XMLSchema#string">TestData
        </Kanban_sValue>
        <nist:Kanban_Size_PROP rdf:resource="#Kanban_Size_1"/>
        <nist:Kanban_type_attr
                rdf:datatype="http://www.w3.org/2001/XMLSchema#string">PickUp
        </nist:Kanban_type_attr>
</nist:Kanban>
```

Figure 4. Example of the RDF instance document

This example can be used to demonstrate the opposite transformation from RDF to XML. In that case, the input whould be the RDF document and the output would be the XML instance shown above.

5 Usage scenario

The ATHENA Semantic Mediation approach proposes a common ontology that formalizes concepts and relations from a business domain of interest. The common ontology in the ATHENA context is called the Reference Ontology (RO) [8]. The RO represents a common point of reference for local business systems involved in an interoperable business process. Concepts from a local system are mapped to concepts defined in the RO. The concepts from the RO should have the same semantic meaning for all local systems; i.e., the local systems ought to interpret concepts within the local models in the same way.

The ATHENA Semantic Mediation approach requires the local systems to express their local interfaces in the RDF format. An architecture to validate new tools in support of interoperability provisioning processes can be found in [7]. The tools developed by the ATHENA community enable interoperable data exchange among local systems by reconciling their models to and from the RO. The tools are categorized as design-time and runtime tools. Design-time tools are used once to

prepare an execution environment that will be used by runtime tools. As shown in Figure 5., ATHOS is used to build an RO; A* is used to annotate RDFS models using the RO; THEMIS is the repository of the RDFS models; and ARGOS is used for creating reconciliation rules from RDFS models to the RO and vice versa.

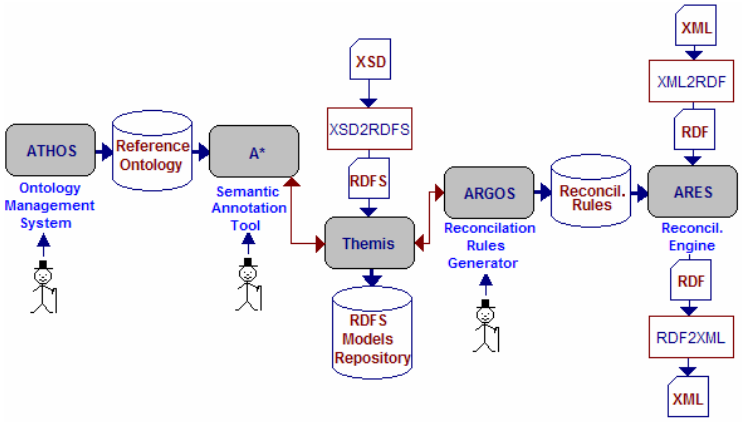

Figure 5. ATHENA design-time and run-time of Semantic Mediation environment[3]

We apply our XSD2RDFS transformation at design time when a local system has an existing XML Schema-based interface. We automatically transform that XML Schema into an RDFS model that obeys all ATHENA requerements in Table 1 and Table 2. In this way, the transformed XML Schema can be easily uploaded into the Themis repository and, later, annotated using the A* tool. The design time phase is completed after the forward and backward reconciliation rules are defined using the ARGOS tool [9]. These rules will be used at runtime by the ARES execution engine.

Once we have built the reconciliation rules for all local models, we can start the message exchange process among the local systems. By the message exchange process, we mean the process where one local system sends a message to the ATHENA runtime environment, the ARES tool executes the forward rules (i.e., transforms the message to the RO), then the ARES executes backward rules (i.e., transforms the message from the RO to the target message format) and, finally, sends the transformed message to the target local system. This process is inhibited in the case where the local system is only capable of sending XML (not RDF) messages, or if the receiving local system expects an XML (not RDF) message. We solved this runtime problem using our XML2RDF and RDF2XML transformations.

If a local system sends a message in the XML format, we interrupt the message flow and transform it using the XML2RDF transformation. If successful, we obtain the original message in the RDF format that is input to ARES. Another transformation is necessary if the target system expects the message in the XML

[3] The picture is adapted from [22], Figure 1 – A3 Semantic Reconciliation Streamline

format. Now, we apply the RDF2XML transformation to transform the RDF output from the ARES tool to obtain the target XML format.

6 Conclusion and future work

In this paper we described an approach for transformation of XML instances to RDF instances and vice-versa as well as transformation of an XML Schema to an RDF Schema. Our solution was driven by the requirements posed by the Semantic Mediation approach developed in the ATHENA IP project. We started with an analysis of related efforts relevant to this transformation problem and we showed that the existing approaches do not meet our requirements. In our approach to transformation, we straightforwardly mapped every XML element to an RDF class and maintained 'the tree' of inter-element relations by mapping these relations to RDF properties. The developed transformation tool can be used to allow the ATHENA-mediated interoperability scenarios and, furthermore, to easily expose proprietary XML-based data-models and data using the Semantic Web technologies. This may be achieved without any additional programming. In addition, currently available tools did not support the full set of features provided in the developed transformation tool that includes XML Schema import, multiple namespaces management, XML transformation without a XML Schema presence, generation of RDFS-conformant RDF individuals, and management of identical names in the schema definition.

A current limitation of the developed approach is that it does not preserve information about ordering of the XML elements during a round-trip transformation from XML to RDF and back to XML. We plan to add the order information where one solution can be based on a naming convention (i.e., to enrich the *rdfs:Class* name by adding order number) where the convention should be applied during the XSD2RDFS transformation. In that case, XML Schema has to be present during XML2RDF transformation. Another solution can be based on adding XML Schema information to the RDF2XML transformation process and this approach can be applied in any context for any given RDF instance. Development of such a transformation tool would make the Semantic Mediation tools more general and useful to the community.

A significant next challenge is to transform an XML Schema into RDF Schema so that the implicit, semantically rich XML metadata representation may be maintained. In this case, the model approach is a more appropriate solution. However, it would be very difficult to fulfill all the requirements in our project, which are essentially following from the tool-design decisions and would, consequently, require redesign of the tools.

Disclaimer

Certain commercial software products are identified in this paper. These products were used only for demonstration purposes. This use does not imply approval or

endorsement by NIST, nor does it imply these products are necessarily the best available for the purpose.

7 Reference

[1] Extensible Markup Language (XML) 1.0 (Fourth Edition) – http://www.w3.org/TR/REC-xml/

[2] Resource Description Framework (RDF) - http://www.w3.org/RDF/

[3] XML Schema Part 1: Structures Second Edition - http://www.w3.org/TR/xmlschema-1/

[4] RDF Vocabulary Description Language 1.0: RDF Schema - http://www.w3.org/TR/rdf-schema/

[5] XML Schema Part 2: Datatypes Second Edition. http://www.w3.org/TR/xmlschema-2

[6] Web Ontology Language (OWL). http://www.w3.org/2004/OWL/

[7] Ivezic N, Barkmeyer E, Kulvatunyou B, Jones A, Snack P, Marjanovic Z, Cho H, (2006) A Validation Architecture for Advanced Interoperability Provisioning.

[8] Missikoff M, Schiappelli F, (2005) A Method for Ontology Modeling in the Business Domain. Proceedings of the Open Interop Workshop on Enterprise Modelling and Ontologies for Interoperability, Porto (Portugal).

[9] ATHENA Technical documentation. ARGOS User Manual, draft version 0.1

[10] W3C Web Services Activity - http://www.w3.org/2002/ws/

[11] ATHENA European Integration Project - www.athena-ip.org

[12] Battle S, (2006) Gloze: XML to RDF and back again. Jena User Conference, Bristol (United Kingdom)

[13] Melnik S, (1999) Bridging the Gap between RDF and XML. Technical report, Stanford University. http://www-db.stanford.edu/~melnik/rdf/fusion.html

[14] Trastour D, Ferdinand M, Zirpins C, (2004) Pragmatic Reasoning-Support for Web-Engineering: Lifting XML-Schema to OWL. ICWE 2004, Munich (Germany).

[15] Melnik S, (1999) Simplified Syntax for RDF. http://www-db.stanford.edu/~melnik/rdf/syntax.html

[16] DuCharme B, (2004) Converting XML to RDF. Web page - http://www.xml.com/pub/a/2004/09/01/tr.html

[17] Bohring H,Soren A, (2006) Mapping XML to OWL Ontologies. Web page - http://www.semanticscripting.org/XML2OWL_XSLT

[18] Anicic N, Ivezic N, Marjanovic Z, (2006) Mapping XML Schema to OWL. I-ESA Conference, Bordeaux (France).

[19] Sperberg-McQueen CM, Miller E, (2004) On mapping from colloquial XML to RDF using XSLT. Presented at Extreme markup languages web page - http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Sperberg-McQueen01/EML2004Sperberg-McQueen01.html

[20] García R, Celma O. (2005) Semantic Integration and Retrieval of Multimedia Metadata. Presented at the 5th Knowledge Markup and Semantic Annotation Workshop, Galway (Ireland). http://www.iua.upf.edu/mtg/publications/d450b9-ISWC2005-GarciaCelma-SemAnnot2005.pdf

[21] Amazon Web Services web page - http://docs.amazonwebservices.com/

[22] ATHENA Technical documentation, Guideline for A3 pilots (2006), version 1.3

[23] XSL Transformations (XSLT) Version 1.0 - http://www.w3.org/TR/xslt