Interoperability of XML Schema Applications with OWL Domain Knowledge and Semantic Web Tools

Chrisa Tsinaraki[1] and Stavros Christodoulakis[1]

[1]TUC/MUSIC, Technical University of Crete Campus, 73100 Kounoupidiana, Crete, Greece
{chrisa, stavros}@ced.tuc.gr

Abstract. Several standards are expressed using XML Schema syntax, since the XML is the default standard for data exchange in the Internet. However, several applications need semantic support offered by domain ontologies and semantic Web tools like logic-based reasoners. Thus, there is a strong need for interoperability between XML Schema and OWL. This can be achieved if the XML schema constructs are expressed in OWL, where the enrichment with OWL domain ontologies and further semantic processing are possible. After semantic processing, the derived OWL constructs should be converted back to instances of the original schema. We present in this paper XS2OWL, a model and a system that allow the transformation of XML Schemas to OWL-DL constructs. These constructs can be used to drive the automatic creation of OWL domain ontologies and individuals. The XS2OWL transformation model allows the correct conversion of the derived knowledge from OWL-DL back to XML constructs valid according to the original XML Schemas, in order to be used transparently by the applications that follow XML Schema syntax of the standards.

Keywords: Interoperability, Standards, XML Schema, OWL, Ontologies.

1. Introduction

The development of the Web and the emergence of advanced network infrastructures that allow the fast and efficient information delivery allow the end-users to access Web documents and Web applications. In such an open environment, the applications developed by different vendors interoperate on the basis of the emergent standards. The default standard for data exchange in the Internet today is the *eXtensible Markup Language (XML)* [3], which allows the representation of structured Web documents. The classes of the Web documents are described using the *XML Schema Language* [6], which uses XML syntax and supports very rich structures and datatypes for XML documents. Due to its structural capabilities and the central role in the data exchange in the Internet, many standards in different application areas are directly specified in XML Schema. Examples include several important standards in multimedia, like MPEG-7 [5] and MPEG-21 [15], in e-learning, like IEEE LOM [11] and SCORM [1], in Digital Libraries, like METS [10], and many others.

However, several Web applications that utilize XML-based standards would benefit from advanced semantic support. These applications typically need to integrate domain ontologies and do further semantic processing including reasoning within the constructs that the standards provide. For example, consider the MPEG-7 applications. MPEG-7, which is described using XML Schema syntax, provides rich language constructs for describing the structure and the content of multimedia data such as video. Retrieval, browsing, personalization and delivery applications in MPEG-7 would benefit from the use of domain knowledge, and MPEG-7 includes general-purpose constructs that could be used for describing domain knowledge, albeit in a rather cumbersome way [18]. Programmers that are going to introduce domain knowledge in MPEG-7 are much more likely to be familiar with the ontology description mechanisms of the *Web Ontology Language (OWL)* [14] than those of MPEG-7. In addition, some MPEG-7 applications, like, for example, knowledge acquisition from video streams, may significantly benefit from the use of logic-based reasoners such as the ones available for OWL. There is then a strong motivation for some MPEG-7 applications to be able to work with the semantics of MPEG-7 expressed in OWL and integrated with OWL domain ontologies. This way, additional acquired knowledge expressed in OWL such as metadata acquired during knowledge acquisition from video streams can be encoded using the semantics of the standard. The resulting knowledge should be converted back to standard MPEG-7/XML constructs in order to be used transparently by applications that follow the standard.

We present in this paper XS2OWL, a model and system software that allow applications using XML Schema based standards to use the Semantic Web methodologies and tools while maintaining the compatibility with the XML schema versions of the standards. With XS2OWL we express each XML Schema based standard in OWL-DL as a *Main Ontology* so that the constructs of the standard become Semantic Web objects. The main ontology allows integrating the constructs of the standard with domain knowledge expressed in the form of OWL domain ontologies. It also allows all the OWL-based Semantic Web tools, including reasoners, to be used with the standard-based descriptions. The integration of the domain ontologies and the utilization of the OWL-based Semantic Web tools may result in deriving additional knowledge useful to the applications, expressed as OWL individuals. The XS2OWL transformation model also supports the conversion of the OWL constructs back to the XML Schema based constructs of the standard for use by all the applications that are compatible with the original XML Schemas. This is achieved through a *Mapping Ontology* that systematically captures the semantics of the XML Schema constructs that cannot be directly captured in the main ontology.

Very limited research has been reported in the literature in this area. We had first observed the need for such methodology and software in conjunction with MPEG-7 [16, 17, 18]. For the use of some MPEG-7 applications we developed a manual methodology for expressing the MPEG-7 semantics in OWL. An Upper OWL-DL ontology capturing the *MPEG-7 Multimedia Description Schemes (MDS)* [13] and the *MPEG-21 Digital Item Adaptation (DIA) Architecture* [12] was defined [16]. This ontology could then be extended with domain ontologies. A soccer ontology and a Formula 1 ontology have been developed as extensions of the Upper ontology [17] for capturing knowledge from video streams in these two domains. We also defined and implemented a set of transformation rules [17] that allow the transformation of the OWL metadata (individuals) that describe the multimedia content defined using the Upper ontology and the domain ontologies into the original MPEG-7/21 constructs. The transformation rules relied on a mapping ontology that systematically captured the information of the MPEG-7/21 schemas that cannot be captured in the Upper ontology. Although this work is an important motivating example for the need of the general-purpose mechanism described in this paper, it is only a special case applicable to MPEG-7/21 and to the applications that use them. In addition, the conversion of the XML Schema constructs describing the MPEG-7/21 to OWL-DL was done manually and the transformation back to XML Schema based syntax was focusing on the needs of MPEG-7/21.

A methodology and a tool have been developed in the context of the MapOnto project [2] for the heuristic definition of complex semantic mappings between the attributes of an XML Schema and the datatype properties of an OWL ontology. The tool input consists of the XML Schema, the OWL ontology and a set of correspondences between the attributes of the former and the datatype properties of the later.

In a recent work [7], almost automatic one-way transformation of XML Schema constructs to OWL constructs has been proposed. The transformations produce OWL-Full ontologies that partially capture the XML Schema semantics. The methodology proposed in [7] looses information during the transformation process from XML Schema to OWL, and it can not be used to transform OWL individuals which may be produced later back to the original XML Schema constructs. This is however needed in all the applications mentioned above. In addition, human intervention is needed in order to preserve the validity of the ontologies produced when homonym top-level XML Schema constructs exist in the source XML Schema (i.e. elements, attributes, types or groups with the same name). Finally, some transformations in [7] do not follow closely the semantics of the XML Schema.

In contrast to the work reported in [7], the XS2OWL model presented in this paper encapsulates methodology and rules that allow automatically transforming the XML Schema constructs to OWL-DL constructs (not OWL-Full). Thus, it guarantees computational completeness and decidability of reasoning in the OWL ontologies produced. In addition, the XS2OWL model allows the representation of all the knowledge needed to transform the individuals generated or added later on to the OWL ontologies back to the original XML Schema based constructs. This way, they can be used by the standard-compatible applications. Finally, we have implemented and extensively tested the XS2OWL model with very large standards based on XML, and verified manually and automatically the correctness of the model and software.

The rest of the paper is structured as follows: In section 2 we provide background information. The XS2OWL transformation model and system are outlined in section 3 and the XS2OWL model is detailed in section 4. In section 5 we present the XS2OWL model evaluation. The paper conclusions and our future research directions are presented in section 6.

2. Background

In this section we present some background information needed in the rest of the paper, including the *XML Schema Language* and the *Web Ontology Language (OWL)*.

The XML Schema Language. The *XML Schema Language* [6] allows the definition of classes of XML documents using XML syntax and provides datatypes and rich structuring capabilities. An XML document is composed of *elements*, with the root element delimiting the beginning and the end of the document. The XML Schema elements belong to XML Schema *types*, specified in their “type” attribute, and are distinguished into complex and simple elements, depending on the kind (simple or complex) of the types they belong to. Reuse of element definitions is supported by the *substitutionGroup* attribute, which states that the current element is a specialization of another element. The elements may either have a predefined order (forming XML Schema *sequences*) or be unordered (forming XML Schema *choices*). The main difference between sequences and choices is that all the sequence items must appear within the containing sequence in their specified order, while the choice items may appear at any order. Both sequences and choices may be nested. The minimum and maximum number of occurrences of the elements, choices and sequences are specified, respectively, in the “minOccurs” and “maxOccurs” attributes (absent “minOccurs” and/or “maxOccurs” correspond to values of 1). Reusable complex structures, combining sequences and choices, may be defined as *model groups*.

The *simple XML Schema types* are usually defined as restrictions of the basic datatypes provided by XML Schema (i.e. strings, integers, floats, tokens etc.). Simple types can neither contain elements nor carry attributes. The *complex XML Schema types* represent classes of XML constructs that have common features, represented by their elements and *attributes*. The attributes describe features with values of simple type and may form *attribute groups* comprised of attributes that should be used simultaneously. The elements represent features of the complex XML Schema types with values of any type. Default and fixed values may be specified for both attributes and simple type elements, in the *default* and *fixed* attributes respectively. Inheritance is supported for both simple and complex types, and the base types are referenced in the “base” attribute of the type definitions.

All the XML Schema constructs may have textual annotations, specified in their “annotation” element. The top-level XML Schema constructs (attributes, elements, simple and complex types, attribute and model groups) have unique *names* (specified in their “name” attribute). The nested elements and attributes have unique names in the context of the complex types in which they are defined, while the nested (complex and simple) types are unnamed. All the XML Schema constructs may have unique identifiers (specified in their “id” attribute). The top-level constructs may be referenced by other constructs using the “ref” attribute.

The Web Ontology Language (OWL). The *Web Ontology Language (OWL)* [14] is the dominant standard in ontology definition. OWL has been developed according to the description logics paradigm and uses *RDF (Resource Description Framework)/RDFS (Resource Description Framework Schema)* [9, 4] syntax. Three OWL species of increasing descriptive power have been specified: (a) *OWL-Lite*, which is intended for lightweight reasoning but has limited expressive power; (b) *OWL-DL*, which provides the description logics expressivity and guarantees computational completeness and decidability of reasoning; and (c) *OWL-Full*, which has more flexible syntax than OWL-DL, but does not guarantee computational completeness and decidability of reasoning.

The basic functionality provided by OWL is: *(a) Import of XML Schema Datatypes* that extend or restrict the basic datatypes (e.g. ranges etc.). The imported datatypes have to be declared (using the rdfs:Datatype construct), as *RDFS datatypes*, in the ontologies they are used; *(b) Definition of OWL Classes* (using the owl:Class construct), organized in subclass hierarchies (using the rdfs:subClassOf construct), for the representation of sets of individuals sharing some properties. Complex OWL classes can be defined via *set operators* (using the owl:intersectionOf, owl:unionOf and owl:complementOf constructs) or via *direct enumeration* of their members (using the owl:oneOf construct); *(c) Definition of OWL Individuals*, essentially instances of the OWL classes, following the restrictions imposed on the class in which they belong; and *(d) Definition of OWL Properties*, which may form property hierarchies (using the rdfs:subPropertyOf construct), for the representation of the features of the OWL class individuals. Two kinds of properties are provided by OWL: *(i) Object Properties*, defined using the owl:ObjectProperty construct, which relate individuals of one OWL class (the property domain, defined using the rdfs:domain construct) with individuals of another OWL class (the property range, defined using the rdfs:range construct); and *(ii) Datatype Properties*, defined using the owl:DatatypeProperty construct, which relate individuals belonging to one OWL class (the property domain) with values of a given datatype (the property range). Restrictions may be defined on OWL class properties (using the owl:Restriction construct), including type (using the owl:allValuesFrom construct), cardinality (using the owl:minCardinality, owl:maxCardinality and owl:cardinality constructs), and value (using the owl:hasValue construct) restrictions. OWL classes, (object and datatype) properties and individuals are identified by unique identifiers, that are specified in the "rdf:ID" attribute. They may also have labels, defined using the rdfs:label construct, and textual descriptions, defined using the rdfs:comment construct.

3. XS2OWL Overview

We present in this section the XS2OWL model for transforming XML Schema constructs in OWL-DL and its realization in the XS2OWL system. The XS2OWL system transforms every XML Schema it takes as input, through the implementation of the XS2OWL transformation model (outlined in Fig. 1), into: *(a)* A *main* OWL-DL ontology that directly captures the XML Schema semantics in OWL-DL; *(b)* A *mapping* OWL-DL ontology that keeps the mapping of the rdf:IDs of the OWL constructs of the main ontology with the names of the XML Schema constructs and systematically captures the semantics of the XML Schema constructs that cannot be directly captured in the main ontology, since they cannot be represented by corresponding OWL constructs; and *(c)* A *datatypes* XML Schema that contains the simple XML Schema datatypes defined in the source XML Schema that are imported in the main ontology.

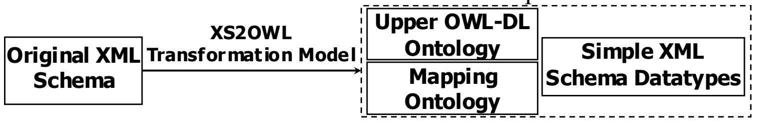

Fig. 1. Outline of the XS2OWL Transformation Model

The main OWL ontology essentially contains the OWL constructs to which the corresponding XML Schema constructs are transformed. As already mentioned, some of the XML Schema construct semantics cannot be expressed in OWL. The semantics of these constructs do not affect the domain ontologies that may extend the main ontology and they are not used by the OWL reasoners; however, they are important when individuals extending the main ontology have to be transformed back to valid

XML descriptions compliant with the source XML Schema. For example, the elements of an XML Schema sequence should appear in a predefined order, while the OWL properties that are the constructs corresponding to the elements are always organized in unordered sets. As a consequence, the ordering information cannot be directly captured in the main ontology.

In order to support this functionality, we have defined a model that allows transforming the OWL constructs back to XML Schema constructs. This model captures the semantics of any arbitrary XML schema that cannot be represented in OWL and is expressed as an OWL-DL ontology, the *OWL2XMLRules Ontology* (available at http://www.music.tuc.gr/ontologies/OWL2XMLRules/OWL2XMLRules). For a particular XML Schema that is being transformed to OWL-DL, XS2OWL generates a mapping ontology, which extends the OWL2XMLRules ontology with individuals, keeps the mapping of the rdf:IDs of the OWL constructs of the main ontology with the names of the XML Schema constructs and represents the constructs of the original schema that cannot be directly represented in OWL.

The classes of the OWL2XMLRules ontology that represent the semantics of the XML Schema constructs that cannot be directly mapped to OWL constructs during the XML Schema to OWL transformation are the following:

- The *DatatypePropertyInfoType* class, which captures information about the datatype properties that cannot be directly expressed in OWL.
- The *ElementInfoType* class, which captures information about the XML Schema elements that cannot be directly expressed in OWL.
- The *ComplexTypeInfoType* class, which captures information about the complex XML Schema types that cannot be directly expressed in OWL.
- The *ChoiceInfoType* and *SequenceInfoType* classes, which capture, respectively, information about the XML Schema choices and sequences that cannot be directly expressed in OWL.

This way, there is no information loss during the XS2OWL transformations of the XML Schema constructs to OWL-DL constructs. As a consequence, all the semantics of the XML Schemas can be utilized by OWL-based tools and applications. For example, consider the transformation of individuals formed according to the main ontologies to XML documents obeying the original XML Schemas. Since the semantics of the XML Schema constructs that cannot be directly expressed in OWL are captured in the mapping ontologies, such information (e.g. the sequence element order) will be used in order to guarantee that the produced documents will be valid.

The XS2OWL transformation model has been implemented using the *XML Stylesheet Transformation Language (XSLT)* [8].

4. The XS2OWL Transformation Model

We present in this section the XS2OWL model, which allows the transformation of the XML Schema constructs to OWL-DL constructs. An overview of the XS2OWL Transformation Model is provided in Table 1, while the transformations of the individual constructs are formally presented in the following paragraphs.

The XML Schema constructs are provided in the first column of Table 1, while the OWL constructs that represent them in the main ontology are provided in the second column. As shown in Table 1, the complex XML Schema types are mapped to OWL classes, since they both represent sets of entities with common features. The simple XML Schema datatypes are mapped to datatype declarations, since OWL does not directly support the definition of simple datatypes, but only allows using simple XML Schema datatypes that have been declared in the OWL ontologies. The attributes are mapped to datatype properties, since they both represent simple type features, while the (simple and complex type) elements are mapped to (datatype and object) properties. The sequences and the choices are represented by OWL unnamed classes formed using set operators and cardinality restrictions on the sequence/choice items. Finally, the annotations of the XML Schema constructs are mapped to OWL comments.

The mapping ontology constructs representing the semantics of the XML Schema constructs that cannot be expressed directly in OWL are presented in the third column and in the fourth column are shown the contents of the datatypes XML Schema.

Table 1. Overview of the XS2OWL Transformation Model.

XML Schema Construct	**OWL-DL Representation**		
	Main Ontology	**Mapping Ontology**	**Datatypes**
Complex Type	Class	*ComplexTypeInfoType* individual	
Simple Datatype	Datatype Declaration		Simple Type
Element	(Datatype or Object) Property	*ElementInfoType* individual	
Attribute	Datatype Property	*DatatypePropertyInfoType* individual	
Sequence	Unnamed Class - Intersection	*SequenceInfoType* individual	
Choice	Unnamed Class - Union	*ChoiceInfoType* individual	
Annotation	Comment		

Complex XML Schema Type Transformation. Let the complex XML Schema type *ct*, which is formally described in (1), where: (a) *name* is the name of *ct*; (b) *cid* is the (optional) identifier of *ct*; (c) *base* is the (simple or complex) type extended by *ct*; (d) *attributes* is the list of the attributes of *ct*; (e) *sequences* is the list of the sequences of *ct*; and (f) *choices* is the list of the choices of *ct*.

ct(name, cid, base, attributes, sequences, choices) (1)

The XS2OWL transformation of *ct* is different, depending on the type extended by *ct* (if it is simple or complex). The attributes and the elements that are defined or referenced in *ct* are transformed, in both cases, into properties.

If *ct* extends a complex type, it is represented in the main ontology by the OWL class *c*, formally described in (2), where:

c(id, super_class, label, value_restrictions, cardinality_restrictions) (2)

- *id* is the unique rdf:ID of *c* and has *name* as value if *ct* is a top-level complex type. If *ct* is a complex type nested within the definition of an element *e*, *id* is a unique, automatically generated name of the form *concatenate(ct_name, '_', name, '_UNType')*, where *ct_name* is the name of the complex type containing *e*. If *e* is a top-level element, *ct_name* has the 'NS' string as value. The *concatenate(...)* algorithm takes as input an arbitrary number of strings and returns their concatenation;
- *super_class* states which class is extended by *ct* and has *base* as value;
- *label* is the label of *ct* and has *name* as value;
- *value_restrictions* is the set of the value restrictions that represent the fixed values that may exist for some *ct* attributes and *ct* sequence/choice elements. The value of

the restrictions is the value of the “fixed” attribute of the *ct* attributes and the *ct* sequence/choice elements;

- *cardinality_restrictions* is the set of the cardinality restrictions assigned to the properties representing the *ct* attributes and the *ct* sequence/choice elements. The cardinality restrictions are generated as follows:
  - According to the value of the “use” attribute of the XML Schema attributes: (a) If “use” has the “required” value, a cardinality restriction of value 1 is generated; (b) If “use” has the “prohibited” value, a cardinality restriction of value 0 is generated; and (c) If “use” is absent or has the “optional” value, a maximum cardinality restriction of value 1 is generated;
  - Cardinality, minimum and maximum cardinality restrictions are generated for the elements of the complex type, according to the “minOccurs” and “maxOccurs” attribute values of the elements and/or the sequences, choices and model groups the elements are organized in.

The semantics of *ct* that cannot be represented in OWL are represented in the mapping ontology by the *ComplexTypeInfoType* individual *ct* that is formally described in (3), where: (a) *id* is the unique rdf:ID of *ct* and has *name* as value; (b) *type_id* represents the identifier of the OWL class *c* that represents *ct* in the main ontology. *type_id* is represented as the “typeID” datatype property of *ct*; (c) *dpi_list* is the list of the representations of the datatype properties of *c*; and (d) *container_list* is the list of the representations of the *ct* containers (sequences and/or choices).

ct(id, type_id, dpi_list, container_list) (3)

If *ct* extends a simple type, it is represented in the main ontology by the OWL class *c*, formally described in (4).

c(id, label, value_restrictions, cardinality_restrictions) (4)

The fact that *ct* extends the simple type *base* is represented in the main ontology by the datatype property *ep* that is formally described in (5), where: (a) *eid* is the unique rdf:ID of *ep* and has *concatenate(base, ‘_content’)* as value; (b) *range* is the range of *ep* and has *base* as value; and (c) *domain* is the domain of *ep* and has the *id* of *c* as value.

ep(eid, erange, edomain) (5)

The semantics of *ct* that cannot be represented in OWL are represented in the mapping ontology by the *ComplexTypeInfoType* individual *ct* that corresponds to *c* and is formally described in (3), and the *DatatypePropertyInfoType* individual *dpi* that corresponds to *ep* and is formally described in (6). *dpi* states that the *ep* represents the fact that *ct* extends the simple type *base* through the ‘Extension’ value of the *dpi_type*, represented by the “datatypePropertyType” datatype property.

dpi(id, did, dpi_type) (6)

As an example, consider the complex type “ct” that extends the string datatype and is shown in Fig. 2. The “ct” complex type is represented in the main ontology by the “ct” OWL class, shown in Fig. 3, together with the “content__xs_string” datatype property, which states that “ct” is an extension of xs:string. The information about “ct” in the mapping ontology is shown in Fig. 4.

```
<xs:complexType name="ct">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="a">
    <xs:simpleType>
```

```
      <xs:restriction base="xs:integer"/>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Fig. 2. Definition of the “ct” XML Schema simple datatype. “ct” extends the string datatype with the “a” attribute. “a” is of a simple anonymous type that is a restriction of integer.

```
<owl:Class rdf:ID="ct">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#a__ct_a_UNType"/>
   <owl:maxCardinality rdf:datatype="&xsd;integer">1</owl:maxCardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#content__xs_string"/>
   <owl:cardinality rdf:datatype= "&xsd;integer">1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <rdfs:label>ct</rdfs:label>
</owl:Class>
<owl:DatatypeProperty rdf:ID="content__xs_string">
 <rdfs:domain rdf:resource="#ct"/>
 <rdfs:range rdf:resource="&xs;string"/>
</owl:DatatypeProperty>
```

Fig. 3. Class representing the “ct” complex type of Fig. 2 in the main ontology

```
<ox:ComplexTypeInfoType rdf:ID="ct">
 <ox:typeID>ct</ox:typeID>
 <ox:DatatypePropertyInfo>
  <ox:DatatypePropertyInfoType rdf:ID="ct_a__ct_a_UNType">
   <ox:datatypePropertyID>a__ct_a_UNType</ox:datatypePropertyID>
   <ox:XMLConstructID>a</ox:XMLConstructID>
   <ox:datatypePropertyType>Attribute</ox:datatypePropertyType>
  </ox:DatatypePropertyInfoType>
 </ox:DatatypePropertyInfo>
 <ox:DatatypePropertyInfo>
  <ox:DatatypePropertyInfoType rdf:ID="ct_content__xs_string">
   <ox:datatypePropertyID>content__xs_string</ox:datatypePropertyID>
   <ox:datatypePropertyType>Extension</ox:datatypePropertyType>
  </ox:DatatypePropertyInfoType>
 </ox:DatatypePropertyInfo>
</ox:ComplexTypeInfoType>
```

Fig. 4. The ComplexTypeInfoType and DatatypePropertyInfoType individuals generated in the mapping ontology for the complex type “ct”, shown in Fig. 2

Simple XML Schema Datatype Transformation. Let the simple XML Schema type *st*, formally described in (7), where *body* is the body of the definition of *st*, *id* is the identifier of *st* and *name* is the name of *st*.

$$st(name,\ id,\ body) \qquad (7)$$

XS2OWL transforms *st* into the simple datatype *st’* in the *datatypes* XML Schema, formally described in (8), and the *dd* datatype declaration in the main ontology, formally described in (9).

$$st'(name',\ id,\ body) \qquad (8)$$

$$dd(about,\ is\_defined\_by,\ label) \qquad (9)$$

The *st'* simple datatype has the same *body* and *id* with *st*, while *name'* is formed as follows: If *st* is a top-level simple type, *name'* is the value of its “name” attribute. If *st* is a simple type nested in the *ae* XML Schema construct (that may be an attribute or an element), *name'* has the value *id* if the identifier of *st* is not null. If the identifier of *st* is null, *name'* has as value the result of *concatenate(ct_name, '_', ae_name, '_UNType')*, where *ae_name* is the name of the property that represents *ae* and *ct_name* is the name of the complex type containing *ae*. If *ae* is a top-level attribute or element, *ct_name* has the ‘NS’ string as value;

The *dd* datatype declaration carries the following semantics: (a) *about* is the URI of *st'* referenced by the datatype declaration and is of the form *concatenate(uri, name')*, where *uri* is the URI of the datatypes XML Schema; (b) *is_defined_by* specifies where the datatype definition is located and has the *url* value; and (c) *label* is the label of *dd* and has *name'* as value.

As an example, consider the nested simple datatype of Fig. 2, which is defined in the “a” attribute of the “ct” complex type. It is transformed to the top-level simple datatype shown in Fig. 5, and the OWL datatype declaration shown in Fig. 6.

```
<xs:simpleType name="ct_a_UNType">
 <xs:restriction base="xs:integer"/>
</xs:simpleType>
```

Fig. 5. Top-level datatype representing the nested datatype of Fig. 2 in the datatypes XML Schema

```
<rdfs:Datatype rdf:about="&datatypes;ct a UNType">
 <rdfs:isDefinedBy rdf:resource="&datatypes;"/>
 <rdfs:label>ct a UNType</rdfs:label>
</rdfs:Datatype>
```

Fig. 6. Declaration of the “ct_a_UNType” simple datatype of Fig. 5 in the main ontology

XML Schema Element Transformation. Let the XML Schema element *e*, formally described in (10), where *name* is the name of *e*, *eid* is the identifier of *e*, *type* is the type of *e*, *ct_name* is the name of the complex XML Schema type *c_type* in the context of which *e* is defined (if *e* is a top-level attribute, *ct_name* has the null value), *sub_group* is an (optional) element being extended by *e*, *fixed* is the (optional) fixed value of *e*, *default* is the (optional) default value of *e, min* is the minimum number of occurrences of *e*, *max* is the maximum number of occurrences of *e* and *pos* is the position of *e* if *e* is a sequence element.

$$e(name, type, eid, ct\_name, sub\_group, fixed, default, min, max, pos) \quad (10)$$

XS2OWL represents *e* in the main ontology as a (datatype if *e* is of simple type, object if *e* is of complex type) property *p*, formally described in (11), where: (a) *id* is the unique rdf:ID of *p* and has *concatenate(name, ‘__’, type)* as value; (b) *range* is the range of *p* and has *type* as value; (c) *domain* is the domain of *p* and takes the value of *ct_name*; (d) *label* is the label of *p* and has *name* as value; and (e) *super_property* is the specification of the property specialized by *p* and has *sub_group* as value.

$$p(id, range, domain, label, super\_property) \quad (11)$$

In the mapping ontology *e* is represented by the *ElementInfoType* individual *ei*, formally described in (12), where: (a) *id* is the unique rdf:ID of *ei* and has *concatenate(ct_name, ‘_’, name, ‘__’, type)* as value; (b) *pid* is the rdf:ID of the *p* property that represents *e* in the main ontology. *pid* is represented by the “propertyID” datatype property of *dpi*; (c) *xml_name* is the name of *e* and has *name* as value. *xml_name* is represented by the “elementID” datatype property of *dpi*; (d) *def_val* represents the default value of *e* and has *default* as value. *def_val* is represented as the “default-Value” datatype property of *dpi*; (e) *min_occ* represents the minimum number of occurrences of *e* and has *min* as value. *min_occ* is represented by the “minOccurs” datatype property of *e*; (f) *max_occ* represents the maximum number of occurrences of *e* and has *max* as value. *max_occ* is represented by the “maxOccurs” datatype property of *e*; and (g) *position* represents the position of *e* if *e* is a sequence element. *position* is represented by the “elementPosition” datatype property of *e*.

$$ei(id, pid, xml\_name, def\_val, min\_occ, max\_occ, position) \qquad (12)$$

In addition, if *e* is of simple type, a *DatatypePropertyInfoType* individual *dpi*, formally described in (13), is generated in the mapping ontology, where: (a) *id* is the unique rdf:ID of *dpi* and has *concatenate(ct_name, ‘_’, name, ‘__’, type)* as value; (b) *did* is the rdf:ID of the *p* datatype property that represents *e* in the main ontology. *did* is represented by the “datatypePropertyID” datatype property of *dpi*; (c) *xml_name* is the name of *e* and has *name* as value. *xml_name* is represented by the “XMLConstructID” datatype property of *dpi*; (d) *dpi_type* represents the construct which has been mapped to *p* and has the value *‘Element’*; and (e) *def_val* represents the default value of *e* and has *default* as value.

$$dpi(id, did, xml\_name, dpi\_type, def\_val) \qquad (13)$$

As an example, consider the “e1” element, shown in Fig. 7, of type “c_type2”, which is defined in the context of the complex type “c_type1”. The “e1” element is transformed to the OWL object property “e1__c_type2” of the main ontology (shown in Fig. 8) and the *ElementInfoType* individual “c_type1_e1__c_type2__ei” of the mapping ontology (shown in Fig. 9).

```
<xs:element name="e1" type="c_type2"/>
```

Fig. 7. XML Schema definition of the “e1” element, nested in the complex type “c_type1”

```
<owl:ObjectProperty rdf:ID="e1__c_type2">
 <rdfs:domain rdf:resource="#c_type1"/>
 <rdfs:range rdf:resource="#c_type2"/>
 <rdfs:label>e1</rdfs:label>
</owl:ObjectProperty>
```

Fig. 8. The object property representing the “e1” element of Fig. 7 in the main ontology

```
<ox:ElementInfoType rdf:ID="c_type1_e1__c_type2__ei">
 <ox:propertyID>e1__c_type2</ox:propertyID>
 <ox:elementID>e1</ox:elementID>
</ox:ElementInfoType>
```

Fig. 9. *ElementInfoType* individual representing the element of Fig. 7 in the mapping ontology

The XML Schema model groups essentially are sets of elements organized into (possibly nested) lists and choices. Let the XML Schema model group *g*, formally described in (14), which is comprised of n sequences/choices, where *g_name* is the model group name, *g_id* is the model group identifier and l_i with 1≤i≤n are the group sequences/choices.

$$g(g\_name, g\_id, (l_1 , ..., l_n)) \qquad (14)$$

If l_i is a sequence/choice of m elements formally described in (15), then for 1≤j≤m, $name_{ij}$, is the name of e_{ij}, eid_{ij}, is the identifier of e_{ij}, $type_{ij}$ is the value of the “type” attribute of e_{ij} and sub_group_{ij} is an element being extended by e_{ij}.

$$l_i(lid_i,\ e_{i1}(name_{i1},\ eid_{i1},\ type_{i1},\ sub\_group_{i1}),\ \ldots,\ e_{im}(name_{im},\ eid_{im},\ type_{im},\ sub\_group_{im})) \qquad (15)$$

XS2OWL represents *g* in the main ontology as the (datatype or object) properties p_{ij}, formally described in (16), where: (a) id_{ij} is the unique rdf:ID of p_{ij} and has *concatenate(g_name, '__',* $name_{ij}$*)* as value; (b) $range_{ij}$ is the range of p_{ij} and has $type_{ij}$ as value; (c) $label_{ij}$ is the label of p_{ij} and has $name_{ij}$ as value; and (d) $super\_property_{ij}$ represents the property specialized by *p* and has sub_group_{ij} as value.

$$p_{ij}(id_{ij},\ range_{ij},\ label_{ij},\ super\_property_{ij}) \qquad (16)$$

In the mapping ontology *g* is represented by an *ElementInfoType* individual *ei* for each element e_{ij}, formally described in (12). If e_{ij} is represented in the main ontology by a datatype property, a *DatatypePropertyInfoType* individual *dpi*, formally described in (13), is also generated in the mapping ontology for the e_{ij}.

XML Schema Attribute Transformation. Let the XML Schema attribute *a*, formally described in (17), where *name* is the name of *a*, *aid* is the identifier of *a*, *type* is the type of *a*, *ct_name* is the name of the complex XML Schema type *c_type* in the context of which *a* is defined (if *a* is a top-level attribute, *ct_name* has the null value), *fixed* is the fixed value of *a* and *default* is the default value of *a*.

$$a(name,\ aid,\ type,\ ct\_name,\ fixed,\ default) \qquad (17)$$

XS2OWL represents *a* in the main ontology as an OWL datatype property *dp*, formally described in (18), where: (a) *id* is the unique rdf:ID of *dp* and has *concatenate(name, '__', type)* as value; (b) *range* is the range of *dp* and has *type* as value; (c) *domain* is the domain of *dp* and takes the value of *ct_name*; and (d) *label* is the label of *dp* and has *name* as value.

$$dp(id,\ range,\ domain,\ label,\ comment) \qquad (18)$$

In the mapping ontology *a* is represented as a *DatatypePropertyInfoType* individual *dpi*, formally described in (13), where *dpi_type* represents the construct which has been mapped to *dp* and has the value *'Attribute'*.

As an example, consider the "a" attribute of Fig. 2, which is transformed to the datatype property of the main ontology shown in Fig. 10 and the *DatatypePropertyInfoType* individual of the mapping ontology shown in Fig. 11.

```
<owl:DatatypeProperty rdf:ID="a__ct_a_UNType">
 <rdfs:domain rdf:resource="#ct"/>
 <rdfs:range rdf:resource="&datatypes;ct a UNType"/>
 <rdfs:label>a</rdfs:label>
</owl:DatatypeProperty>
```

Fig. 10. The datatype property representing the "a" attribute of Fig. 2 in the main ontology

```
<ox:DatatypePropertyInfo>
 <ox:DatatypePropertyInfoType rdf:ID="ct_a__ct_a_UNType">
  <ox:datatypePropertyID>a  ct a UNType</ox:datatypePropertyID>
  <ox:XMLConstructID>a</ox:XMLConstructID>
  <ox:datatypePropertyType>Attribute</ox:datatypePropertyType>
 </ox:DatatypePropertyInfoType>
</ox:DatatypePropertyInfo>
```

Fig. 11. The DatatypePropertyInfoType individual representing the "a" attribute of Fig. 2 in the mapping ontology

XS2OWL transforms the XML Schema attribute groups into sets of datatype properties, each of which represents an attribute that belongs to the attribute group. Let *ag* be an XML Schema attribute group comprised of n attributes, formally described in

(19), where *ag_name* is the attribute group name, *ag_ id* is the attribute group identifier and a_i, with 1≤i≤n are the group attributes, formally described in (20). $name_i$ is the name of a_i, aid_i is the identifier of a_i and $type_i$ is the type of a_i.

$$ag(ag\_name, ag\_id, (a_1, ..., a_n)) \quad (19)$$

$$a_i(name_i, aid_i, type_i) \quad (20)$$

XS2OWL represents *ag* in the main ontology as a set of datatype properties dp_i, 1≤i≤n, formally described in (21), where: (a) id_i is the unique rdf:ID of dp_i and has *concatenate(ag_name, '__', name*$_i$*)* as value; (b) $range_i$ is the range of dp_i and has $type_i$ as value; and (c) $label_i$ is the label of dp_i and has $name_i$ as value.

$$dpi_i(id_i, range_i, label_i) \quad (21)$$

In the mapping ontology *ag* is represented by a *DatatypePropertyInfoType* individual *dpi* for each attribute, formally described in (13), with *dpi_type* having the value *'Attribute'*.

XML Schema Sequence and Choice Transformation. XS2OWL transforms both the sequences and the choices into OWL-DL unnamed classes formed using set operators and cardinality restrictions on the sequence/choice items. The classes that represent the complex types where the sequences/choices are defined or referenced are subclasses of the unnamed OWL classes that represent the sequences/choices. The sequence and the choice item cardinality must always be a multiple of an integer in the range [*i_min_occurs*, *i_max_occurs*], where *i_min_occurs* and *i_max_occurs* are the values of the "minOccurs" and the "maxOccurs" attributes of the item.

The sequences are represented in the main ontology as unnamed classes, formed from the intersection of the cardinality restrictions of the sequence items. The algorithm that transforms the XML Schema sequences to unnamed OWL classes is shown in Fig. 12. For the transformation of a sequence *s* the algorithm is initially called as *sequence_restr(s, 1, 1)*.

```
algorithm sequence_restr(sequence, max_p, min_p)
minOc=sequence/@minOccurs
maxOc=sequence/@maxOccurs
temp=''
if ((minOc*min_p=0) and (maxOc='unbounded' or max_p='unbounded'))
 return ''
else
 for each sequence item
  if the item is element
   if (maxOc='unbounded' or max_p='unbounded')
    temp=seq_element_restr(item, min_p*minOc, 'unbounded')
   else
    temp=seq_element_restr(item, min_p*minOc, max_p*maxOc)
   end if
  else if the item is sequence
   if (maxOc='unbounded' or max_p='unbounded')
    temp=sequence_restr(item, min_p*minOc, 'unbounded')
   else
    temp=sequence_restr(item, min_p*minOc, max_p*maxOc)
   end if
  else
   if (maxOc='unbounded' or max_p='unbounded')
    temp=choice_restr(item, min_p*minOc, 'unbounded')
   else
    temp=choice_restr(item, min_p*minOc, max_p*maxOc)
   end if
  end if
  ret=concatenate(ret, temp)
 end for
 return intersection(ret)
end if
end algorithm
algorithm seq_element_restr(element, max_p, min_p)
minOc=sequence/@minOccurs
maxOc=sequence/@maxOccurs
if ((minOc*min_p=0) and (maxOc='unbounded' or max_p='unbounded'))
 return ''
else if (maxOc='unbounded' or max p='unbounded')
 return min_car(item, minOc*min_p)
else if (maxOc*max p=minOc*min p)
 return cardinality(item, minOc*min_p)
else
 return intersection(min_car(item, minOc*min_p), max_car(item, maxOc*max_p))
end if
end algorithm
```

Fig. 12. Algorithm for the generation of unnamed OWL classes representing XML Schema Sequences, where: (a) The min_car(item,min) algorithm produces an owl:minCardinality restriction on the property that represents "item" with value "min"; (b) The cardinality(item,val) algorithm produces an owl:cardinality restriction on the property that represents "item" with value "val"; and (c) The max_car(item,max) algorithm produces an owl:maxCardinality restriction on the property that represents "item" with value "max".

As an example, consider the sequence shown in Fig. 13, which is defined in the context of a complex type *c*. The sequence is represented in the main ontology by the unnamed OWL class shown in Fig. 14, of which the class that represents the complex type *c* is a subclass.

```
<xs:sequence minOccurs="2" maxOccurs="2">
 <xs:element name="e1" type="xs:string"/>
 <xs:element name="e2" type="xs:string" maxOccurs="3"/>
</xs:sequence>
```

Fig. 13. XML Schema Sequence defined in the context of the complex type *c*

```
<owl:Class>
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Restriction>
   <owl:onProperty rdf:resource="#e1  xs string"/>
   <owl:cardinality rdf:datatype="&xsd;integer">2</owl:cardinality>
  </owl:Restriction>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#e2  xs string"/>
   <owl:minCardinality rdf:datatype="&xsd;integer">2</owl:minCardinality>
  </owl:Restriction>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#e2  xs string"/>
   <owl:maxCardinality rdf:datatype="&xsd;integer">6</owl:maxCardinality>
  </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

Fig. 14. Representation of the sequence of Fig. 13 in the main ontology

The choices are represented in the main ontology as unnamed classes, formed from the union of the allowed combinations of the choice elements. The algorithm that transforms the XML Schema choices to unnamed OWL classes is shown in Fig. 15.

```
algorithm choice_restr(choice, maxOc, minOc)
ret=''
if (minOc=0 and maxOc='unbounded'), return ret
else if maxOc='unbounded'
 nz=number of choice items with minOccurs<>0
 z=number of choice items with minOccurs=0
 nzi=choice items with minOccurs<>0
 mat=distribute(nz,0,minOc)
 for i=1 to mat rows
  for j=1 to nz
   min=nzi[j]/@minOccurs
   if nzi[j] is element, add min car(nzi[j], min*mat[i,j]) to ret
   else if nzi[j] is sequence, add sequence_restr(nzi[j], min*mat[i,j],
'unbounded') to ret
   else add choice restr(nzi[j],min*mat[i,j],'unbounded') to ret
   end if
  end for
  ret=intersection(ret)
 end for
 if z>0
  z temp=intersection of deep cardinality(nzi[i],0) (i from 1 to nz)
  return union(z temp, ret)
 else return ret
 end if
else
 nzu=number of choice items with (minOccurs<>0 and maxOccurs<>'unbounded')
 zu=number of choice items with (minOccurs=0 and maxOccurs='unbounded')
 nzui=choice items with (minOccurs<>0 and maxOccurs<>'unbounded')
 mat=distribute(nz,minOc,maxOc)
 for i=1 to mat rows
  for j=1 to nz
   min=nzui[j]/@minOccurs
   max=nzui[j]/@minOccurs
   if nzui[j] is element
    if (max='unbounded' or maxOc='unbounded'), add min car(nzui[j],
min*mat[i,j]) to ret
    else if (min*minOc=0), add max car(nzui[j], max*mat[i,j]) to ret
    else (if min*minOc=max*maxOc), add cardinality(nzui[j], max*mat[i,j])
to ret
    else add intersection(min_car(nzui[j], min*mat[i,j]), max_car(nzui[j],
max*mat[i,j])) to ret
    end if
   else if nzui[j] is sequence
    if (max='unbounded'), add sequence_restr(nzui[j], min*mat[i,j], 'un-
bounded') to ret
    else add sequence_restr(nzui[j], min*mat[i,j], max*mat[i,j]) to ret
    end if
   else
    if (max='unbounded'), add choice restr(nzui[j], min*mat[i,j], 'un-
bounded') to ret
    else add choice restr(nzui[j], min*mat[i,j], max*mat[i,j]) to ret
    end if
   end if
  end for
  intersection(ret)
 end for
 if zu>0
  zu_temp=intersection of cardinality(nzui[i],0) (i from 1 to nzu)
  return union(zu temp, ret)
 else return ret
 end if
end if
end algorithm
```

Fig. 15. Algorithm for the generation of unnamed OWL classes representing XML Schema Choices, where: (a) The deep_cardinality(item,val) algorithm produces an owl:cardinality restriction with value "val" on each property that represents "item" or a (sequence or choice) item of "item"; and (b) The distribute(item_num, min_val, max_val) algorithm calculates the allowed combination of the occurrences of "item_num" items so that the sum of their occurrences is between "min_val" and "max_val".

As an example, consider the choice shown in Fig. 16. The choice is represented in the main ontology by the unnamed OWL class shown in Fig. 17.

```
<xs:choice minOccurs="0">
 <xs:element name="e2" type="xs:string" minOccurs="2" maxOccurs="2"/>
 <xs:element name="e3" type="xs:string" maxOccurs="2"/>
</xs:choice>
```

Fig. 16. XML Schema choice defined in the context of a complex type

```
<owl:Class>
 <owl:unionOf rdf:parseType="Collection">
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Restriction>
     <owl:onProperty rdf:resource="#e2__xs_string"/>
     <owl:maxCardinality rdf:datatype="&xsd;integer">0</owl:maxCardinality>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="#e3__xs_string"/>
     <owl:maxCardinality rdf:datatype="&xsd;integer">2</owl:maxCardinality>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Restriction>
     <owl:onProperty rdf:resource="#e2__xs_string"/>
     <owl:maxCardinality rdf:datatype="&xsd;integer">2</owl:maxCardinality>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="#e3__xs_string"/>
     <owl:maxCardinality rdf:datatype="&xsd;integer">0</owl:maxCardinality>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </owl:unionOf>
</owl:Class>
```

Fig. 17. Representation of the XML Schema Choice of Fig. 16 in the main ontology

If the maximum number of occurrences of a sequence/choice has a large value (but is not unbounded), the manual generation of the unnamed classes is tedious and time-consuming and thus becomes error-prone and practically impossible.

Notice that the exact sequence/choice cardinalities cannot be computed when a choice item is contained in a sequence/choice with unbounded maximum number of occurrences. In addition, information regarding the sequence element ordering cannot be represented in OWL. This information is captured in the mapping ontology. Let *sc* be a sequence or choice formally described in (22), where *sc_id* is the identifier of *sc*, *c_type* is the complex type in which *sc* is defined or referenced, *min* is the minimum number of occurrences of *sc*, *max* is the maximum number of occurrences of *sc* and *elements* is the list of the elements of *sc*.

$$sc(sc\_id, c\_type, min, max, elements) \qquad (22)$$

XS2OWL represents *sc* in the mapping ontology by the (*SequenceInfoType* if *sc* is a sequence, *ChoiceInfoType* if *sc* is a choice) individual *st* formally described in (23), where: (a) *id* is the unique rdf:ID of *st* and has *concatenate(ct_name, '__', i)* as value, where *ct_name* is the name of the class that represents *c_type* in the main ontology and *i* is the index of *sc* in *c_type*; (b) *min_occ* represents the minimum number of occurrences of *sc* and has *min* as value; (c) *max_occ* represents the maximum number of occurrences of *sc* and has *max* as value; and (d) *e_rep* is the list of the representations of the *elements* of *sc*.

$$st(id, min\_occ, max\_occ, e\_rep) \quad (23)$$

5. Evaluation of the XS2OWL Model

The XS2OWL model and its implementation have been applied to several well-accepted standards. We present here the results of the XS2OWL evaluation.

In order to acquire extensive empirical evidence, we applied XS2OWL to several very large and well-accepted standards expressed in XML Schema: The MPEG-7 Multimedia Description Schemes (MDS) and the MPEG-21 Digital Item Adaptation (DIA) Architecture in the multimedia domain, the IEEE LOM and the SCORM in the e-learning domain and the METS standard for Digital Libraries. The result was the transformation of the XML Schema constructs to OWL for each one of those standards. We then enriched the OWL specifications with OWL domain ontologies and produced individuals following the ontologies. Finally, we converted the individuals to XML syntax, valid with respect to the original XML Schemas. The transformations were successful for these standards due to the utilization of the mapping ontologies. We have also found that in all cases the semantics of the standards were fully captured in the main and mapping ontologies generated by the XS2OWL system.

Then we looked closely the formal semantics of the generated OWL ontologies. Since we had in our previous research efforts [17] manually transformed the MPEG-7 MDS and the MPEG-21 DIA Architecture in OWL-DL, we compared the semantics captured in the manually defined ontologies with the semantics captured in the automatically generated ones. The comparison has shown that all the semantics captured during the manual transformations were also captured during the automatic transformations. Then, we compared the manually produced mapping ontology for the Semantic DS of the MPEG-7 MDS with the corresponding part of the automatically produced mapping ontology for the MPEG-7 MDS. Again, the comparison has shown that all the semantics captured in the manually created ontology are also accurately captured in the automatically produced one.

6. Conclusions – Future Work

We have presented in this paper the XS2OWL model that allows the automatic transformation of XML Schemas into OWL-DL ontologies. This transformation allows domain ontologies in OWL to be integrated and logic-based reasoners to be used for various applications, as for example for knowledge extraction from multimedia data. XS2OWL allows the conversion of the generated OWL information back to XML. We have presented also a system that implements the XS2OWL model. We have used the implemented system to validate our approach with a number of well-accepted and extensive standards expressed in XML Schema. The automatically created ontologies have been found to accurately capture the semantics of the XML Schemas.

Our future work in this area includes experimentation that will be conducted in order to evaluate the enhancement of the retrieval effectiveness that will be achieved through the utilization of the products of the XS2OWL model. In particular, we will pose the same queries on top of XML repositories containing (a) Standard-based XML descriptions that were created from scratch; and (b) Standard-based XML descriptions that were derived from the transformation of OWL constructs produced after the semantic processing of OWL individuals defined based on the OWL ontologies produced from the application of the XS2OWL methodology and software on the standards. Precision and recall will be calculated in both cases and will be compared.

Acknowledgments. The work presented here was partially funded in the scope of the DELOS II Network of Excellence in Digital Libraries (IST – Project Record #26059).

7. References

1. ADL Technical Team: Sharable Content Object Reference Model (SCORM), 2004.
2. An, Y., Borgida, A., Mylopoulos, J.: Constructing Complex Semantic Mappings Between XML Data and Ontologies. International Semantic Web Conference 2005: 6-20.
3. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., Cowan, J. (eds.): Extensible Markup Language (XML) 1.1. W3C Recommendation, 2006. (http://www.w3.org/TR/xml11/).
4. Brickley, D., Guha, R. V. (eds.): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004. (http://www.w3.org/TR/rdf-schema).
5. Chang, S.F., Sikora, T., Puri, A.: Overview of the MPEG-7 standard. In IEEE Transactions on Circuits and Systems for Video Technology 11:688–695, 2001.
6. Fallside, D., Walmsley, P. (eds.): XML Schema Part 0: Primer. W3C Recommendation, 2001. (http://www.w3.org/TR/xmlschema-0/).
7. García, R., Celma, O.: Semantic Integration and Retrieval of Multimedia Metadata. In the proceedings of the Semannot'05 Workshop, 2005.
8. Kay, M. (ed.) : XSL Transformations (XSLT) Version 2.0. W3C Recommendation, 2007. (http://www.w3.org/TR/xslt20/).
9. Manola, F., Milles, E. (eds.): RDF Primer. W3C Recommendation, 2004. (http://www.w3.org/TR/rdf-primer).
10. METS: Metadata Encoding and Transmission Standard (METS) Official Website. (http://www.loc.gov/standards/mets/).
11. IEEE LTSC 2002: IEEE 1484.12.1-2002 – Learning Object Metadata Standard. (http://ltsc.ieee.org/wg12/).
12. ISO/IEC: 21000-7:2004 – Information Technology – Multimedia Framework (MPEG-21) – Part 7: Digital Item Adaptation, 2004.
13. ISO/IEC: 15938-5:2003 – Information Technology –Multimedia content description interface – Part 5: Multimedia description schemes. First Edition, ISO/MPEG N5845, 2003.
14. McGuinness, D. L., van Harmelen, F. (eds.): OWL Web Ontology Language: Overview. W3C Recommendation, 2004. (http://www.w3.org/TR/owl-features).
15. Pereira, F.: The MPEG-21 standard: Why an open multimedia framework?. In the Proceedings of the 8th IDMS, LNCS 2158, Lancaster, September 2001, pp. 219–220.
16. Tsinaraki C., Polydoros P. and Christodoulakis S.: Interoperability support for Ontology-based Video Retrieval Applications. In the Proceedings of the CIVR 2004, pp. 582-591.
17. Tsinaraki, C., Polydoros, P. and Christodoulakis, S.: Interoperability support between MPEG-7/21 and OWL in DS-MIRF. Transactions on Knowledge and Data Engineering (TKDE), Special Issue on the Semantic Web Era, pp. 219-232, 2007.
18. Tsinaraki, C., Polydoros, P., Kazasis, F., Christodoulakis S.: Ontology-based Semantic Indexing for MPEG-7 and TV-Anytime Audiovisual Content. In Multimedia Tools and Application Journal (MTAP), 26:299-325, 2005.