Accelerating the Process Designing Domain Ontologies from Scratch Based on XML Schemas

Abstract: Domain ontologies and XML Schemas serve to describe domain data models although they follow different modeling goals. By lifting the syntactic level of XML documents and validating XML Schemas to the semantic level of OWL ontologies and their RDF representations in an automatic way, all the information located in the domains' XML Schemas can be reused by ontology engineers and domain experts to design domain ontologies from scratch. Because this approach supports all components of the XML Schema metamodel, it is ensured that unexceptionally any XML Schema can be converted to a generated ontology. As generated ontologies' structures might be quite complex, domain ontologies can be inferred automatically by means of SWRL rules. Saved time and effort can now be used to add domain-specific semantic information, not covered by underlying XML Schemas, to the domain ontologies.

Keywords: ontology design; domain ontologies; domain ontologies design; generated ontologies; XML Schemas; XSD; XML; semantic web; linked data; OWL; RDF; metadata; semantics; ontologies.

1 Introduction

XML documents are commonly used to store and transfer information in distributed environments. XML documents may be instances of XML Schemas (XSDs) determining their terminology and syntactic structure. XML represents a large set of information within the context of various domains and has reached wide acceptance as standard data exchange format. This has driven the development of the proposed approach. Both data and metadata, structured by ontologies, can be published in the increasingly popular and widely adopted LOD cloud to get linked with a huge number of other RDF datasets of different topical domains[1]. As RDF is an established standard, there is a plethora of tools which can be used to interoperate with data and metadata represented in RDF.

XSD and OWL follow different modeling goals. On the one hand, the XML data model describes the terminology and the syntactic structure of XML documents, a node labeled tree[2]. OWL, on the other hand, is based on formal logic and on the subject-predicate-object triples from RDF. OWL specifies semantic information about specific domains, describes relations between domain classes and thus allows the sharing of conceptualizations. More effective and efficient cooperations between individuals and organizations are possible if they agree on a common syntax (specified by XSDs) and have a common understanding of the domain classes (defined by OWL ontologies). XML is intended to structure and exchange documents (document-oriented), but is used to structure and exchange data (data-oriented), a purpose for which it has not been developed. Also, XSD languages like XSD concentrate on structuring documents instead of structuring data. As OWL is used for describing domain data models semantically, the information needed to depict parts of these data models can be extracted from underlying XSDs and reused as a basis to extend the knowledge representation of particular domains using OWL. I attempt to bridge the gap between XSD and OWL by lifting the syntactic level of XML documents to the semantic level of OWL ontologies.

Traditionally, ontology engineers work in close collaboration with domain experts to design domain ontologies in a manual manner which requires a lot of time and effort. Domain ontologies and XSDs describe domain data models. In many cases, XSDs are already defined and can therefore be reused in the process designing domain ontologies from scratch. Saved time and manpower could be used more effectively to enrich domain data models with additional domain-specific semantic information, not or not satisfyingly covered by the underlying XSDs. The main research question, how the time-consuming process designing domain ontologies based on already available XSDs could be accelerated, results from the stated problem. An extensive evaluation of the proposed approach verifies the appropriate hypothesis, that the effort and the time delivering high quality domain ontologies using the developed approach is much less than creating domain ontologies in a completely manual way.

2 Related Work

Several strategies lifting the syntactic level of XML documents to the semantic level of OWL ontologies can be distinguished. The author has clustered appropriate tools implementing these transformations into three classes depending on the kind of conversion either on the instance, the conceptual, or both the instance and the conceptual level.

On the instance level, Klein (2002) has developed the so-called RDF Schema mapping ontology enabling a one-way mapping of XML documents to RDF. Relevant XML documents' content can be identified. As extension to this approach, Battle has introduced a bidirectional mapping of XML components to RDF (Battle, 2006). The WEESA system implements an automatic transformation from XML to RDF using an OWL ontology, manually created from corresponding XSDs and manually defined rules. XML document instances are not mapped to OWL equivalents (Reif, Gall, and Jazayeri, 2005). O'Connor and Das developed an approach transforming XML documents to individuals of an OWL ontology describing the serialization of the XML document. SWRL is used to map these instances to individuals of a domain ontology (O'Connor and Das, 2010).

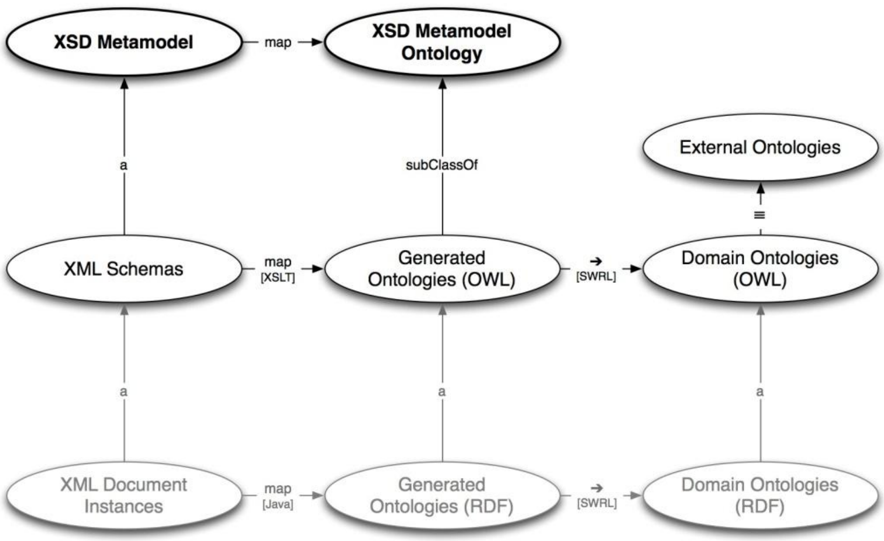
On the conceptual level you can distinguish between approaches converting XSD languages to RDFS or OWL. Several languages for writing schemas like DTD, XSD, DSD (Karlund, Moller, and Schwartzbach, 2000) and Relax NG (Clark et al., 2003) exist. The prototype OntoLiFT (Volz et al., 2003) offers a generic means for converting arbitrary XSD languages to RDFS ontologies semi-automatically. In a first step, XSD languages are transformed into regular tree grammars consisting of non-terminals, terminals, start symbols and production rules (Murata et al., 2005). In a second step, non-terminals and terminals are converted to RDFS classes and production rules are mapped to RDF properties. In comparison with the proposed approach, OntoLiFt converts any XSD language and not just the specific one XSD to ontologies. Anicic et al. evolved an approach based on meta-models transforming between the different models of XSD and OWL (Anicic, Ivezic, and Marjanovic, 2007).

On the instance and the conceptual level, there are methods transforming XML to RDF and XSD to either RDFS or OWL. Within the EU-funded project called 'Harmonise' the interoperability of existing standards for the exchange of tourism data has been achieved by the transformation of XML documents and XSDs into RDF and RDFS ontologies which have been mapped to each other (Dell'Erba et al., 2002). Using the approach of O'Connor and Das (2011), XML document instances are transformed to OWL ontologies even though associated XSDs not exist. As a consequence, unstructured contents can be mapped to OWL ontologies as well. XSDs can also be mapped to OWL ontologies, as XSD documents are represented in XML, too. New OWL ontologies can be generated from scratch and existing ones can be extended. O'Connor and Das evolved XML Master, a language describing OWL ontologies declaratively. XML Master combines the Manchester OWL Syntax[3] and XPath to refer to XML content. O'Connor and Das criticize the limited and unsatisfactory number of OWL constructs supported by current tools converting XSDs to OWL ontologies. Thus, all OWL constructs are covered. One shortcoming associated with this method is that you have to write mapping language expressions manually and therefore you cannot transform XML documents and XSDs to OWL ontologies automatically. Another drawback is that ontology engineers have to be familiar with the Manchester OWL Syntax and XPath to express the mappings. Ferdinand et al. propose both mappings from XML to RDF and XSD to OWL which are independent of each other. This means, OWL individuals do not necessarily correspond to the OWL conceptual model, since XML documents' declarations and definitions may be transferred to differing OWL constructs (Ferdinand, Zirpins, and Trastour, 2004). In addition, another system can be stated transferring XSD components to OWL language constructs at the terminological level and XML document instances to OWL individuals at the assertional level. XPath expressions are applied selecting XML documents' content (Kobeissy, Genet, and Zeghlache, 2007). Besides that, the approach of Tous et al. (2005) is very similar to the method of Kobeissy, Genet, and Zeghlache. Bohring and Auer (2005) devised a mapping between XML and RDF and between XSD and OWL. The authors assume that XML documents are structured like relational databases. Thus, XML documents' relational structures are discovered and represented in OWL. Relations correspond to classes, columns to properties, and rows to instances. XML data model elements are mapped automatically to components of the OWL data model. Named simple and complex types, for instance, are transferred to classes. Elements, containing other elements or having at least one attribute, are converted to classes and object properties between these classes. Both, elements, including neither attributes nor sub-elements, and attributes, assumed representing database columns, are transformed into datatype properties with the surrounding element as domain. Besides, XML cardinality constraints are transformed into equivalent OWL cardinality restrictions.

3 Proposed Approach

Figure 1 visualizes the concept of the devised generic multi-level approach for designing domain ontologies based on already available XSDs (Bosch and Mathiak, 2011).

Figure 1 Generic approach for designing domain ontologies based on XSDs



XSDs determine the vocabulary, the terminology and the syntactic structure of XML documents which are instances of these XSDs. XSDs are instances of the XSD metamodel. The components of the XSD abstract data model, also called element information items (EIIs) in the XML representation, are mapped to classes, universal restrictions on datatype and object properties of a generic ontology called the XSD Metamodel Ontology (XSDMO). The intension of the devised approach is to convert XSDs automatically to generated ontologies' classes, hasValue restrictions on XSDMO's datatype properties, and universal restrictions on XSDMO's object properties using XSLT transformations. As each component of the XSD abstract data model is covered by this approach, unexceptionally any XSD can be translated into a generated ontology. On the instance level, XML documents are mapped to generated ontologies' ABoxes using a Java program as XSLT is less powerful for this purpose. After these two transformation processes, taking only seconds, all the information located in the underlying XSDs of a particular domain is now expressed in the generated ontologies and their RDF representations can be published in the LOD cloud and be linked to resources within different topical domains in the web of data. As generated ontologies are not conform to the highest quality requirements of domain ontologies, generated ontologies' structures are quite complex, and OWL and XSD follow different modeling goals, the generated ontologies are not directly as useful as manually created domain ontologies. Thus, the generated ontologies' class axioms are intended to be further supplemented with additional domain-specific semantic information, not defined in underlying XSDs, in form of domain ontologies. These domain ontologies can be deduced automatically out of the generated ontologies using SWLR rules on the schema and on the instance level. As a consequence, all XML data conforming to XSDs can be imported automatically as domain ontologies' instances. The effort and the time, however, delivering high quality domain ontologies subsequently is much less than creating domain ontologies completely manually.

Novelty of Approach

In comparison to previous general-purpose tools for transforming XSDs into OWL ontologies, the novelty of the devised approach is that the translation of XSDs into generated ontologies is based on the XSD metamodel. The majority of the tools try transforming either XML into RDF on the assertional knowledge level or schemas into ontologies on the terminological knowledge level. The presented method follows a complete approach converting XML documents' content to OWL individuals and XSDs to OWL ontologies. Most tools try extracting semantics directly out of XSDs. The suggested approach, in contrast, only gains information about the terminology and the syntactic structure of XML document instances conforming to XSDs. Domain ontologies are supplemented with domain-specific semantic information in following steps. Many attempts convert XML to RDF and/or XSD languages to ontologies in a manual or at most in a semi-automatic way. This approach translates XSDs and XML into OWL ontologies and their RDF representations in a totally automatic way without any manual modifications of the generated ontologies after the translation process. In conjunction with associated domain ontologies, the resulting ontologies are as usable as ontologies that were built completely manual, but with a fraction of necessary effort. In addition, divers existing methods generate RDFS ontologies and not the more expressive OWL ontologies.

4 Mapping of XSD Metamodel to XSDMO

The XSD metamodel components are mapped to classes, universal restrictions on datatype and object properties of the generic XSDMO ontology. Table 1 sketches these mappings which are described in this section.

Meta-EIIs

Meta-EIIs have been mapped to XSDMO's classes representing the meta-EIIs. The class 'Element', for instance, stands for the XSD meta-model's meta-EII 'element'.

Attributes of meta-EIIs

Attributes of meta-EIIs have been mapped to datatype properties '<attribute>_<domain meta-EII>_String' with the class standing for the domain meta-EII as domain and the class representing the XSD built-in primitive datatype 'string' as range. Attributes of meta-EIIs have also been mapped to universal restrictions on these datatype properties: <domain meta-EII> ⊑ ∀ <attribute>_<domain meta-EII>_String.String. The universal restrictions express that the class including all domain meta-EII individuals is defined as the sub-class of the anonymous complex super-class of all the instances, which can only have relationships along the datatype properties '<attribute>_<domain meta-EII>_ String' with individuals of the type 'String' or have no relationships along these datatype properties.

Table 1 Mapping of XML Schema Metamodel to XML Schema Metamodel Ontology (XSDMO)

XML Schema Metamodel	XSDMO
meta-EIIs	classes: <meta-EII>
attributes of meta-EIIs	datatype properties and universal restrictions: <domain meta-EII> ⊑ ∀ <attribute>_<domain meta-EII>_String.String
any well-formed XML content of meta-EIIs Appinfo|Documentation	datatype properties and universal restrictions: <Appinfo|Documentation > ⊑ ∀ any_<Appinfo|Documentation>_String.String
texts contained in XML document instances' elements and attributes	datatype properties and universal restrictions: <Element|Attribute> ⊑ ∀ value_<Element|Attribute>_String.String
attributes of meta-EIIs referring to meta-EIIs (attributes 'ref', 'substitutionGroup', 'refer')	object properties and universal restrictions: <domain meta-EII> ⊑ ∀ <ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range meta-EII>
attributes of meta-EIIs referring to type definitions (attributes 'type' and 'base')	object properties and universal restrictions: <domain meta-EII> ⊑ ∀ <type|base>_<domain meta-EII>_Type.Type
attribute 'memberTypes'	object property and universal restriction: <union> ⊑ ∀ memberTypes_union_Type.Type
meta-EIIs' part-of relationships	object properties and universal restrictions: <domain meta-EII> ⊑ ∀ contains_<domain meta-EII>_<range meta-EII>.<range meta-EII>
sequence of in meta-EII 'sequence' contained meta-EIIs	object property and universal restrictions: <sequence> ⊑ ∀ sequence.<range meta-EII>

The attribute 'name' of the meta-EII 'element', for example, have been mapped to the datatype property 'name_Element_String' and to the datatype property's universal restriction Element ⊑ ∀ name_Element_String.String, as elements can only have 'name_Element_String' relationships to 'String' individuals.

Any well-formed XML content of meta-EIIs Appinfo|Documentation

The meta-EIIs Appinfo and Documentation may comprise any well-formed XML content such as XML elements, XML attributes, and plain text. For this reason, any well-formed XML content of the meta-EIIs Appinfo and Documentation is mapped to the datatype properties 'any_<Appinfo|Documentation>_String' and to the universal restrictions on these datatype properties: <Appinfo| Documentation > ⊑ ∀ any_<Appinfo|Documentation>_String.String.

Texts contained in XML document instances' elements and attributes

Elements and attributes of XML documents may comprise text. Thus, we have added the datatype properties 'value_<Element|Attribute>_String' and the datatype properties' universal restrictions <Element|Attribute> ⊑ ∀ value_<Element|Attribute>_String.String to the XSDMO. On the instance level, the XML document excerpt <Label lang="en">Age</Label> is converted to the property assertions value_Element_String (Label-Individual…, 'Age') and value_Attribute_String (Lang-Individual…, 'en').

Attributes of meta-EIIs referring to meta-EIIs (attributes 'ref', 'substitutionGroup', 'refer')

Meta-EIIs' attributes like 'ref', 'refer', or 'substitutionGroup' referring to other meta-EIIs have been mapped in the XSDMO to the object properties '<ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>' and to the universal restrictions: <domain meta-EII> ⊑

∀ <ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range meta-EII>. Elements, for instance, can only have 'ref_Element_Element' relationships to elements according to the object property's universal restriction Element ⊑ ∀ ref_Element_Element. Element.

Attributes of meta-EIIs referring to type definitions (attributes 'type' and 'base')

Meta-EIIs' attributes 'base' and 'type' refer to simple ur-type, simple type, or complex type definitions. As a consequence, these attributes would be mapped to six object properties '<type|base>_<domain meta-EII>_ SimpleType| AnySimpleType|ComplexType'. XSLT transformations, creating generated ontologies automatically out of XSDs, would have to determine the object properties' range classes 'AnySimpleType', 'SimpleType', and 'ComplexType' as part of the object properties' identifiers at runtime. If the attributes 'type' or 'base' either point to simple or complex type definitions, which are defined in external XSDs, these XSDs would have to be physically available to traverse their XML trees and to iterate over each simple and complex type definition. But in many cases, external XSDs are not physically available. Therefore, we have mapped the attributes 'type' and 'base' to the object properties '<type|base>_<domain meta-EII>_Type' with the range class 'Type' representing the super-class of all three more specific type definitions: simple ur-type, simple type, and complex type definitions. The attributes 'base' and 'type' have also been mapped to the object properties' universal restrictions <domain meta-EII> ⊑ ∀ <type|base>_<domain meta-EII>_Type.Type. Considering the object property's universal restriction Element ⊑ ∀ type_Element_Type.Type, elements can only have 'type_Element_Type' relationships to 'Type' individuals (simple or complex type definitions in this case) or have no such relations.

Attribute 'memberTypes'

The attribute 'memberTypes' of the EII 'union' may include simple ur-type and simple type definitions separated by blank characters. This attribute has been mapped to the object property 'memberTypes_union_Type' and to the object property's universal restriction <union> ⊑ ∀ memberTypes_union_Type.Type.

Meta-EIIs' part-of relationships

Meta-EIIs may contain other meta-EIIs. That's why the object properties 'contains_<domain meta-EII>_<range meta-EII>' and associated universal restrictions <domain meta-EII> ⊑ ∀ contains_<domain meta-EII>_<range meta-EII>.<range meta-EII> have been specified. In accordance with the object property's universal restriction Sequence ⊑ ∀ contains_Sequence_ Element.Element, sequences, can only include elements along the object property 'contains_Sequence_Element' and no instances of other classes.

Sequence of in meta-EII 'sequence' contained meta-EIIs

The universal restrictions on the object properties 'contains_Sequence_<range meta-EII>' state for each range meta-EII (annotation, element, group, choice, and sequence) that range instances have to be of the classes representing these range meta-EIIs. The object property 'sequence' and the object property's universal restriction <sequence> ⊑ ∀ sequence.<range meta-EII> have been added to the XSDMO enabling to capture the strict order of in EIIs 'sequence' contained EIIs by means of the mapping of XSDs to generated ontologies.

5 Mapping of XSDs to Generated Ontologies

XSDs are translated into classes, hasValue restrictions on XSDMO's datatype properties, and universal restrictions on XSDMO's object properties. Table 2 demonstrates the in this chapter delineated mappings of XSDs to OWL generated ontologies. Bosch and Mathiak (2012) explain the implementation of these mappings using XSLT transformations in detail.

Table 2 Mapping of XML Schemas to Generated Ontologies

XML Schemas	Generated Ontologies
EIIs	sub-classes of XSDMO’s classes: <EII> ⊑ <meta-EII>
values of EIIs’ attributes	hasValue restrictions on XSDMO’s datatype properties: <domain EII> ⊑ ∃ <attribute>_<domain meta-EII>_String.{<String>}
any well-formed XML content of EIIs Appinfo|Documentation	hasValue restrictions on XSDMO’s datatype properties: <Appinfo|Documentation > ⊑ ∃ any_<Appinfo|Documentation>_String.{<String>}
values of EIIs' attributes referring to EIIs (attributes 'ref', 'substitutionGroup', ‘refer’)	universal restrictions on XSDMO’s object properties: <domain EII> ⊑ ∀ <ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range EII>
values of EIIs' attributes referring to type definitions (attributes 'type' and 'base')	universal restrictions on XSDMO’s object properties: <domain EII> ⊑ ∀ <type|base>_<domain meta-EII>_Type.<range EII>
values of attribute ‘memberTypes’	univeral restriction on XSDMO’s object property: <union> ⊑ ∀ memberTypes_Union_Type.<union of Type EIIs>
EIIs’ part-of relationships	universal restrictions on XSDMO’s object properties: <domain EII> ⊑ ∀ contains_<domain meta-EII>_<range meta-EII>.<union of range EIIs>
sequence of in EII ‘sequence’ contained EIIs	universal restrictions on XSDMO’s object property: <sequence> ⊑ ∀ sequence.<union of EIIs>

EIIs

EIIs are transformed into sub-classes of the XSDMO’s super-classes: <EII> ⊑ <meta-EII>. To show an example, the XSD’s EII ‘element’ with the name ‘Label’ (<xs:element name="Label"/>) is converted to the class ‘Label-Element…’. This class is then defined as sub-class of the super-class ‘Element’ (Label-Element… ⊑ Element), as all ‘Label’ individuals are also part of the ‘Element’ class extension.

Values of EIIs’ attributes

Values of EIIs’ attributes are translated into hasValue restrictions on XSDMO’s datatype properties <domain EII> ⊑ ∃ <attribute>_<domain meta-EII>_String.{<String>}, as classes representing domain EIIs are defined as sub-classes of the anonymous complex super-classes of all the individuals which have at least one relationship along the datatype properties '<attribute>_<domain meta-EII>_String' to the specified individuals of the XSD’s primitive datatype 'string'. The value of the attribute ‘name’ of the EII ‘element’ (<xs:element name="Label"/>) is transformed into the datatype property hasValue restriction Label-Element… ⊑ ∃ name_Element_String.{'Label'}, as ‘Label’ elements must have at least one name which is ‘Label’ and of the type ‘string’.

Any well-formed XML content of EIIs Appinfo|Documentation

Any well-formed XML content of the EIIs Appinfo and Documentation such as XML elements, XML attributes, and plain text is converted to hasValue restrictions on XSDMO's datatype properties <Appinfo|Documentation> ⊑ ∃ any_<Appinfo|Documentation>_String.{<String>}. The text contained in the EII 'appinfo' (<xs:appinfo>This is an application information.</xs:appinfo>) is converted to the datatype property hasValue restriction Appinfo1… ⊑ ∃ any_Appinfo_String.{'This is an application information.'}.

Values of EIIs' attributes referring to EIIs (attributes 'ref', 'substitutionGroup', 'refer')

Values of EIIs' attributes 'ref', 'substitutionGroup', and 'refer' referring to other EIIs, are translated into XSDMO's object properties' universal restrictions <domain EII> ⊑ ∀ <ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range EII>. The reference to the global element 'Label' (<xs:element ref="Label"/>), for instance, is converted to the object property's universal restriction Label-Element-Reference1… ⊑ ∀ ref_Element_Element.Label-Element….

Values of EIIs' attributes referring to type definitions (attributes 'type' and 'base')

Values of EIIs' attributes 'type' and 'base' referring to simple ur-type, simple type, or complex type definitions are converted to universal restrictions on XSDMO's object properties: <domain EII> ⊑ ∀ <type|base>_<domain meta-EII>_Type.<range EII>. The value 'VariableType' of the attribute 'type' of the EII 'element' with the name 'Variable' (<xs:element name="Variable" type="VariableType"/>), for example, is transformed into the object property's universal restriction Variable-Element… ⊑ ∀ type_Element_Type.VariableType-Type….

Values of attribute 'memberTypes'

The attribute 'memberTypes' of the EII 'union' may contain multiple simple ur-type and simple type definitions separated by blank characters. Consequently, the value of this attribute is converted to the XSDMO's object property's universal restriction <union> ⊑ ∀ memberTypes_Union_Type.<union of Type EIIs>. The attribute 'memberTypes', for instance, contains references to one simple ur-type and two simple type definitions (<xs:union memberTypes = "SimpleType1 SimpleType2 xs:string"/>). The value of the attribute 'memberTypes' is translated into the object property's universal restriction Union1… ⊑ ∀ memberTypes_Union_Type . (SimpleType1-Type… ⊔ SimpleType2-Type… ⊔ string-Type…).

EIIs' part-of relationships

Because EIIs may include one to multiple EIIs, universal restrictions on XSDMO's object properties <domain EII> ⊑ ∀ contains_<domain meta-EII>_<range meta-EII>.<union of range EIIs> are used to map EIIs' part-of relationships. To state an example, the following sequence contains only one 'element' EII, a reference to the global element 'Label': <xs:sequence> <xs:element ref="Label"/> </xs:sequence>. According to the object property's universal restriction Sequence1… ⊑ ∀ contains_ Sequence_Element.Label-Element-Reference1…, the range of the object property can only comprise instances of one class representing the reference to the global element 'Label'. If EIIs have more than one EII as content, the domain EIIs can only have relationships along particular object properties to individuals of the anonymous complex super-class consisting of the union of multiple classes representing the contained range EIIs. The part-of relationship of the sequence (<xs:sequence> <xs:element ref = "VariableName"/> <xs:element ref="Label"/> </xs:sequence>) is transferred into the object property's universal restriction Sequence1… ⊑ ∀ contains_Sequence_Element.(VariableName-Element-Reference1… ⊔ Label-Element-Reference2…).

Sequence of in EII 'sequence' contained EIIs

According to the universal restrictions on the object properties 'contains_Sequence_Element|Sequence', sequence individuals can only have relationships along these object properties to element|sequence instances. Sequences may not only include either elements or sequences but also annotations, groups, and choices simultaneously. Furthermore, sequences are not only containers of multiple classes' individuals. They also store the strict order of contained EIIs. As instances of different classes may be contained and to store the strict order of included EIIs, we added the object property ‘sequence’ and the universal restrictions <sequence> ⊑ ∀ sequence.<union of EIIs> to the XSDMO. In our example, sequence individuals either contain VariableName or Label individuals: Sequence1… ⊑ ∀ sequence.(VariableName-Element-Reference1… ⊔ Label-Element-Reference2…). The sequence is extracted implicitly appropriate to the order of the union operands.

6 Derivation of Domain Ontologies and Linking to External Ontologies

So far, XSDs, describing specific domain data models and determining the syntactic structure of appropriate XML documents, are converted automatically to OWL generated ontologies using XSLT transformations and XML document instances are translated into an RDF representation of the generated ontologies. Subsequently, domain ontologies are inferred both on the schema and the instance level out of these generated ontologies in an automatic manner using SWRL rules which are executed by rule engines like Pellet, the OWL 2 reasoner for Java[4]. The antecedents of SWRL rules are specified according to the syntactic structures of XML document instances, storing particular domains' data and meta-data. The consequents of SWRL rules, however, are defined corresponding to the domain ontologies' conceptual models. Thus, to define SWRL rules, ontology engineers have to devise the domain ontologies' conceptual models with the help of domain experts in a first stage. Generated ontologies and therefore XSDs' structures can be very complex and thus are not intended for information retrieval tasks specified and executed by users. When domain ontologies are derived, users can perform queries on domain ontologies using intuitive semantics of the particular domain without knowledge of complex XSDs' structures. Although SWRL rules work completely on the schema level, both terminological and assertional knowledge can be deduced. Domain ontologies' classes can be annotated as equivalent to classes of existing accepted and widely adopted external ontologies such as SKOS or Dublin Core. The resulting benefit is that reasoners may use additional semantic information defined in external ontologies for their deductions (Kupfer et al., 2007).

7 Use Case

To get a better idea of how XSDs and XML documents are translated into generated ontologies' TBoxes and ABoxes and of how domain ontologies are derived out of these generated ontologies on the schema and on the instance level, we show a complete use case covering the most important mappings and motivating the approach's application. Bosch, Wira-Alam, and Mathiak (2011) have discussed use cases associated with a DDI ontology and Bosch et al. (2012) delineate its conceptual model. The Data Documentation Initiative (DDI)[5] is an acknowledged international standard for the documentation and management of data from the social, behavioral, and economic sciences. In this use case, excerpts of the DDI ontology are deduced out of the underlying XSDs describing this particular domain. More specifically, it will be derived that a certain resource is a social science variable with a particular variable label, if specific conditions are fulfilled.

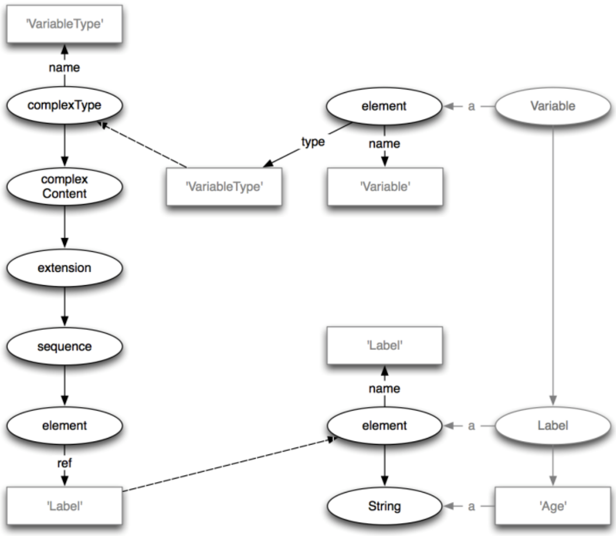
Domain ontologies are derived out of generated ontologies resulting from XSDs within seconds by means of XSLT. Because of the complexity of XSDs' structures and as XSDs may not contain all the information domain experts want to express in a domain ontology, the generated ontologies' complexity is then reduced and the generated ontologies are extended in form of domain ontologies by additional domain-specific semantic information not directly covered by the XSDs.

XML and XSD

Figure 2 visualizes the XML document (on the right), storing information about variables, and the XSD, determining the XML's syntactic structure.

XML elements ‘Variable’ may contain XML elements ‘Label’ corresponding to variable labels which may include plain text such as ‘Age’. ‘Variable’ is an instance of the XSD EII ‘element’ whose ‘name’ attribute has the value ‘Variable’ and whose ‘type’ attribute has the value ‘VariableType’ referring to the complex type definition ‘VariableType’.

Figure 2 XML and XSD

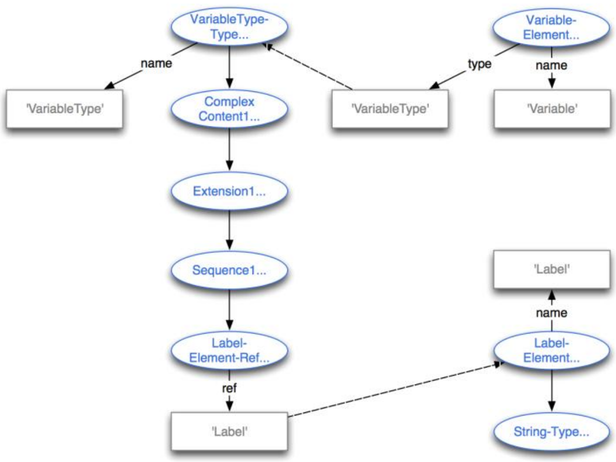


This complex type comprises the EII 'complexContent' including the XSD component 'extension' which contains a sequence. This sequence comprises a reference to the global element with the name 'Label' which is the type of the XML element 'Label'. The global XML element 'Label' may include XSD strings.

Map XSDs' EIIs to generated ontologies' classes

To be able to use this rich syntactic information for the ontology, instead of just using the instance data, we first transform the schema automatically with generic XSLT transformations to a generated ontology (see figure 3).

Figure 3 Map XSDs' EIIs to generated ontologies' classes

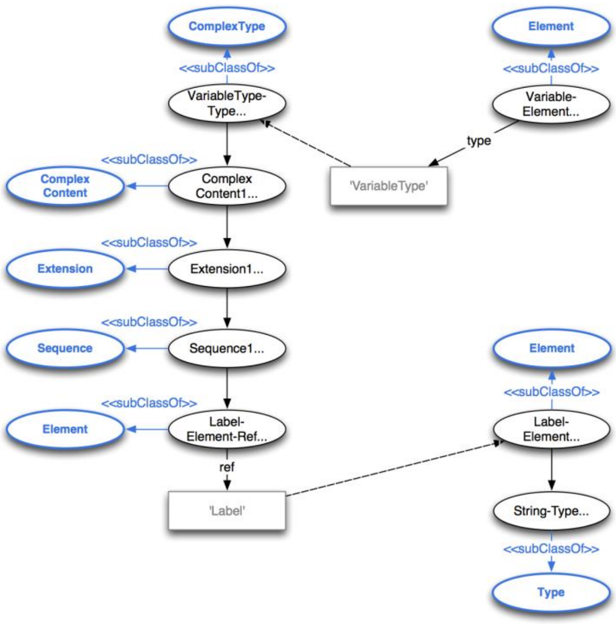


The first step is to convert each XSD's EII to a class. Therefore, the XSLT assigns class identifiers considering the naming conventions (see Bosch and Mathiak, 2011) which ensure the URIs' globally uniqueness. In contrast to OWL, XSD has very few restrictions on unique naming. Generated ontologies' URIs have to be quite long to be globally unique. The global element 'Variable' (<xs:element name="Variable"…/>), for example, is translated into the class 'Variable-Element…' with the meta-EII 'element' as part of its identifier.

Sub-Class relationships

Now, the XSD's EIIs are transformed into the generated ontology's classes with globally unique URIs. But so far, the transformation does not cover the XSDs' semantics. These semantics can be added by defining sub-class relationships (see figure 4).

Figure 4 Sub-class relationships



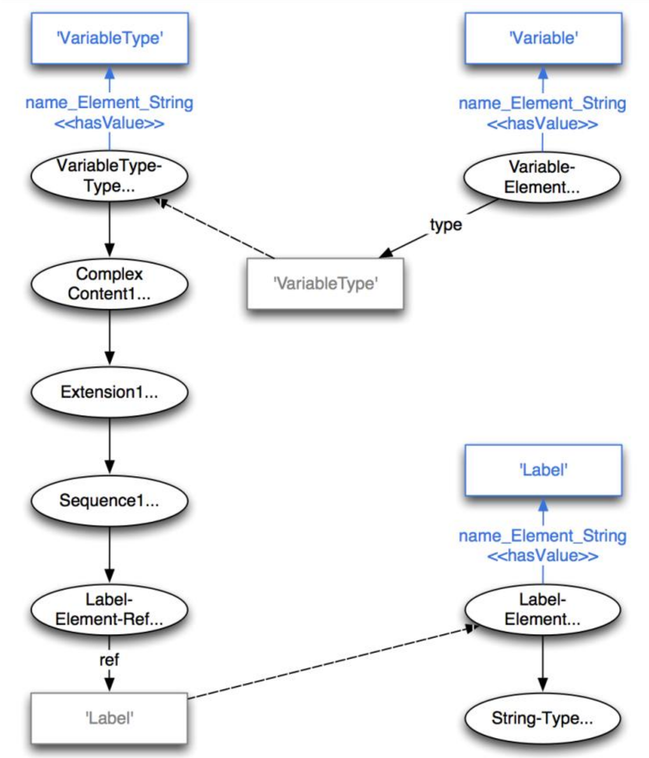
The classes are defined as sub-classes of the super-classes specified in the XSDMO: <EII> ⊑ <meta-EII>. Classes standing for specific XSD elements like 'Variable-Element…' are translated into sub-classes of the super-class 'Element' representing the meta-EII 'element' (<Variable-Element…> ⊑ <Element>), as each particular EII 'element' is also part of the 'Element' class extension. In more simple terms, each specific element is an element.

HasValue restrictions on datatype properties

So far, the XSD's EIIs are converted to sub-classes of the XSDMO's super-classes representing XSD meta-EIIs. As next step EIIs' attributes' values are converted to datatype properties '<attribute>_<domain meta-EII>_String' and to hasValue restrictions on

XSDMO's datatype properties: <domain EII> $\sqsubseteq$ $\exists$ <attribute>_<domain meta-EII>_String.{<String>} (see figure 5).

Figure 5 HasValue restrictions on datatype properties

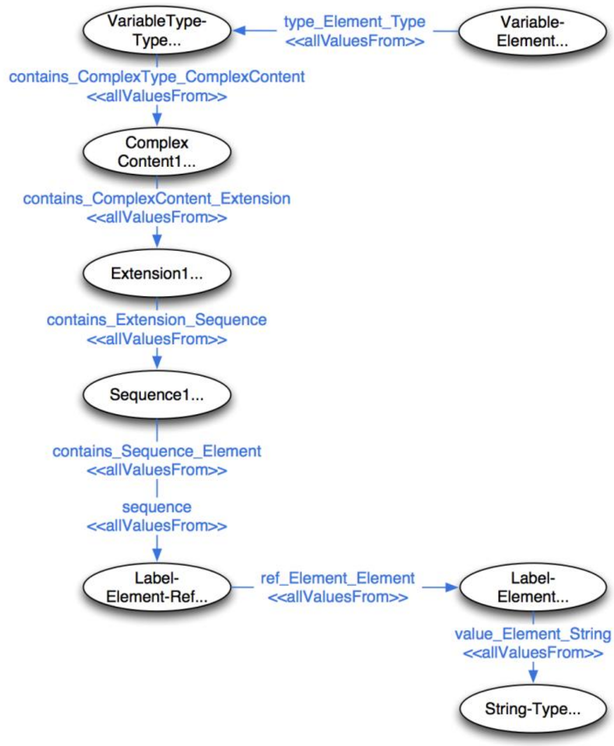


The value 'Variable' of the 'element' EII's attribute 'name' (<xs:element name="Variable"…/>) is translated into the datatype property 'name_Element_String', pointing from an element to a string, and into the XSDMO's datatype property's hasValue restriction Variable-Element… $\sqsubseteq$ $\exists$ name_Element_String.{'Variable'}, since 'Variable-Element…' resources must have at least one relationship along the datatype property 'name_Element_String' to the string 'Variable'. In other words, variable elements must have the name 'Variable'.

Universal restrictions on object properties

XSD's EIIs and XSD's EIIs' attributes' values are now translated. The last step is to map EIIs' part-of relationships, XML elements and attributes' content, and EIIs' attributes' values referring to either type definitions or other EIIs (see figure 6). Values of EIIs' attributes referring to other EIIs, are transformed into XSDMO's object properties' universal restrictions <domain EII> $\sqsubseteq$ $\forall$ <ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range EII>.

Figure 6 Universal restrictions on object properties



The value 'Label' of the 'element' EII's attribute 'ref' (<xs:element ref="Label"/>) referring to the EII 'element' with the name 'Label' is translated into the object property 'ref_Element_Element' and its universal restriction Label-Element-Reference1… $\sqsubseteq$ $\forall$ ref_Element_Element.Label-Element…. Values of EIIs' attributes referring to type definitions are translated into universal restrictions on XSDMO's object properties: <domain EII> $\sqsubseteq$ $\forall$ type|base_<domain meta-EII>_Type.<range EII>. The value 'VariableType' of the attribute 'type' of the EII 'element' with the name 'Variable' (<xs:element name="Variable" type = "VariableType"/>) is converted to the object property's universal restriction Variable-Element… $\sqsubseteq$ $\forall$ type_Element_Type. VariableType-Type…. The part-of relationship of the EII 'sequence' is translated into the object property's universal restriction Sequence1… $\sqsubseteq$ $\forall$ contains_Sequence_Element. Label-Element-Reference1…. The sequence includes only a reference to the global element 'Label'. The strict order of the in the sequence contained EIIs is expressed by the object property's universal restriction Sequence1… ⊑ ∀ sequence.Label-Element-Reference1…. As resources of the class 'Label-Element…' may have text as content, i.e. 'String-Type…' individuals, the datatype property 'value_Element_String' is introduced and the datatype property's universal restriction Label-Element… ⊑ ∀ value_Element_ String.String is defined. While this means that instead of three simple nodes, we suddenly have a plethora of classes with long names, it also means that we adequately model the complete semantic relationships. We can fully appreciate how components relate to other ones on all three levels of instance, schema and metamodel. Since this is all automatically generated, this multiplication of information is not detrimental, but instead allows us to use all this data in a way that is fully integrated with each other. At no cost of time for the ontology engineer. Now, all the information located in the underlying XSDs of a specific domain is also expressed in generated ontologies.

Derive domain ontology

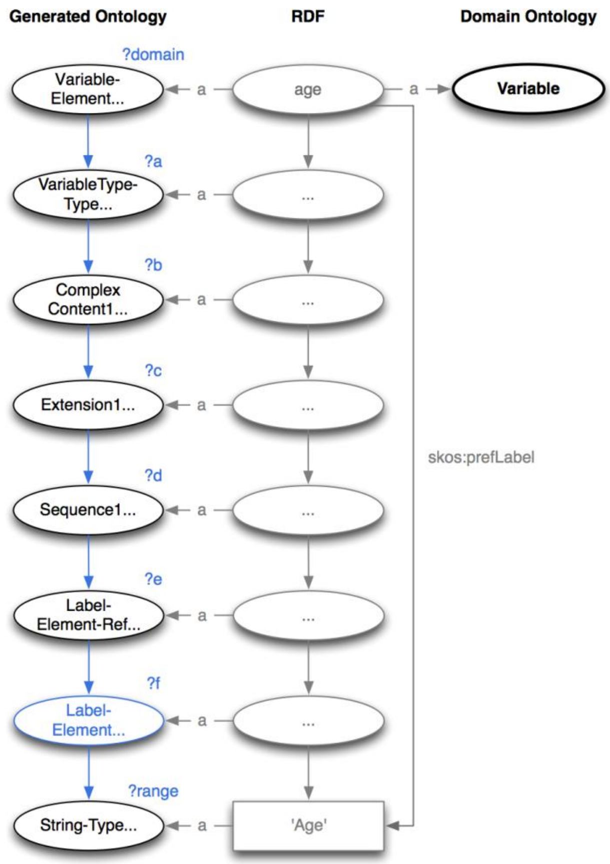
Structures of XSDs and generated ontologies are rather complex, generated ontologies do not correspond to the highest quality requirements of domain ontologies, and XSD and OWL have different modeling goals. Because of these reasons, our idea is not to use the generated ontologies directly. Instead, new classes, datatype and object properties are added based on the generated ontology. This happens automatically using SWRL rules. Figure 7 visualizes the generated ontology, its RDF representation, the domain ontology's extraction to be derived, and the SWRL rule's atoms.

We want to deduce an excerpt of DDI-RDF. More specifically, we want to derive that the resource with the URI 'age', assigned to the class 'Variable-Element…', is also a variable and that the same resource has the variable label 'Age' of the datatype 'string'. The datatype property 'skos:prefLabel' represents relationships between variables and variable labels.

The following program fragment demonstrates the antecedent and the consequent of the SWRL rule which is executed to derive the two statements explained before.

```
(?domain type_Element_Type ?a) ∧
(?a
 contains_ComplexType_ComplexContent
 ?b) ∧
(?b
 contains_ComplexContent_Extension
 ?c) ∧
(?c contains_Extension_Sequence ?d) ∧
(?d contains_Sequence_Element ?e) ∧
(?e ref_Element_Element ?f) ∧
(?f rdf:type Label-Element…) ∧
(?f value_Element_String ?range) ->
(?domain rdf:type Variable) ∧
(?domain skos:prefLabel ?range)
```

Figure 7 Derive domain ontology



The two statements can be derived since the individual 'age', substituting the SWRL variable '?domain', has a relationship along the object property 'type_Element_Type' to an individual replacing the variable 'a'. This resource is linked to an instance '?b' via the 'contains_ComplexType_ComplexContent' object property. Further, there's a navigation path from the '?b' individual to the '?f' instance through the stated object properties. As XML elements 'Label', which are instances of the EII 'element' with the name 'Label' (<xs:element name="Label"/>), may contain text nodes such as 'Age', the '?f' instance is assigned to the class 'Label-Element…', representing the EII 'element' with the value 'Label' of the attribute 'name'. This class assignment ensures that derived variable labels are only those strings contained in the element 'Label' and not in other elements. According to the SWRL rule, the '?f' resource must have a relationship along the datatype property 'value_Element_String' to a '?range' individual substituted by the string 'Age'. The concrete instances 'age' and 'Age' correspond to the antecedent of the SWRL rule, i.e. there is a navigation path from the resource 'age' to the string 'Age' through the stated object and datatype properties. Therefore, it can be inferred that the resource 'age' is a variable with the variable label 'Age'.

The advantage of this rule is that it works purely on the schema level and can thus be reused for any instance or document data we may encounter. By means of SWRL rules, generated ontologies' instances can be mapped to individuals of widely used and accepted ontologies like Dublin Core or SKOS. Another benefit is that all XML data conforming to XSDs can be imported automatically as domain ontologies' instances. In summary, the process of designing domain ontologies can now be supplemented with all of the XSDs' information about the domains of interest, which allows us to automate part of the modeling and also to keep it closer to the original intention of the XSD.

8 Evaluation

Generic test cases, derived from the components of the XSD abstract data model, enable to verify that any XSD can be translated into an OWL generated ontology and that XML documents corresponding to XSDs can be converted to RDF representations of generated ontologies. Bosch and Mathiak (2013) have evaluated the proposed approach extensively. We have published the evaluation results on a GitHub repository.

The first step of our method is to transform XSDs into generated ontologies completely automatically using XSLT transformations. We have converted multiple widely known and accepted XSDs from the academic and from the industry field. Our XSLT stylesheet has translated 10,000 XSD constructs contained in 20 DDI-XSDs in only 30 seconds. The XSD of Simple Dublin Core with its 40 constructs has been transformed in 1 second and the 5 XSDs of Qualified Dublin Core containing 250 XSD constructs in 7 seconds. All calculations can be made in under a minute. The effort in computing time is negligible in comparison with the time needed for the second step of the semi-automatic approach.

The second step of our approach is to define SWRL rules to derive domain ontologies automatically on the instance and on the schema level. We have specified SWRL rules for 3 different domain ontologies. For Simple Dublin Core[7], we have defined all SWRL rules. For Qualified Dublin Core[8] and the DDI-RDF Discovery Vocabulary[8] (an ontology of DDI), we have written a couple of representative SWRL rules for each of the SWRL rule types. For DDI-RDF, we estimate 15 person-hours to define 200 SWRL rules. As these SWRL rules are written by hand, a graphical user interface could assist users creating SWRL rules semi-automatically which would lead to time improvements.

To verify the hypothesis that the time and the effort delivering domain ontologies with high quality using the proposed approach is much less than creating domain ontologies completely manually, we have determined the effort and the expenses for both approaches. DDI-RDF serves as use case since we have had the honor to be part of creating this ontology manually.

For the evaluation of the semi-automatic approach, we have to distinguish between the time actually needed for the formalization of the domain ontology and for developing the conceptual ideas. As we see from our experience with DDI-RDF, the effort required for the development of the conceptual ideas would be 50 percent of the working time spent for the traditional approach. 95 person-days or 17,500€ would have to be invested to evolve the ontology's conceptual model. We would have to invest 2 person-days or 350€ for the formalization of DDI-RDF, i.e. the definition of the OWL axioms and the SWRL rules. In total, we would have to spend 18,000€ designing DDI-RDF based on the already available XSDs. Additionally, travelling, lodging, and board expenses have to be invested as domain experts have to come together discussing conceptual ideas. We calculate 20,000€ for these expenses, which is the half of the travelling, lodging, and board expenses spent for the traditional approach. In total, 38,000€ would have to be needed to design DDI-RDF using the semi-automatic approach. The total expenses creating DDI-RDF manually are 75,0000€ including the working times and travelling, lodging, and board expenses. For the semi-automatic approach only half of this amount is needed - namely 38,000€.

9 Conclusions, Results and Future Work

This approach aims to speed up the task developing domain ontologies from the ground up. XSDs, characterizing domain data models and already evolved by domain experts, serve as a basis since contained information is reused. Although RDF representations of generated ontologies, automatically created out of the XSDs within seconds, can be published in the LOD cloud and combined with other RDF datasets, our idea is to derive domain ontologies automatically out of the generated ontologies using SWRL rules. Additionally, resulting domain ontologies can be supplemented with semantic information not specified in the underlying XSDs.

The overall concept of the approach has been finalized and the mapping of the XSD abstract data model components to class axioms of the XSDMO has been defined and implemented. The mapping between XSDs and OWL generated ontologies has been specified and programmatically realized as well. Also the generality of the approach has been verified, since the generic test cases have shown that all meta-EIIs of the XSD meta-model are covered and thus each XSD can be transformed into an OWL generated ontology using the same transformation rules.

Currently, we are writing a Java program translating XML document instances into an RDF representation of the generated ontologies. Moreover, we will create additional Java code converting XML documents without corresponding XSDs. So far, the most relevant subsets of the DDI domain ontology are derived and appropriate SWRL rules are defined. As part of the approach's limitations we will define use cases for which an automatic approach is suitable (e.g. when XSDs do not represent the domain knowledge enough or when knowledge extraction is critical) and those for which it is not a good solution (e.g. when XSDs do not represent the domain knowledge correctly). We will extend our comprehensive evaluation by converting more XSDs from different heterogeneous domains to generated ontologies and by applying the semi-automatic approach to more domain ontologies from different and heterogeneous communities.

References

Anicic, N., Ivezic, N. and Marjanovic, Z. (2007) 'Mapping XML Schema to OWL' in *Enterprise Interoperability V*, pp. 243-252.

Battle, S. (2006) 'Gloze: XML to RDF and back again' in *1st Jena User Conference*, Bristol, UK.

Bohring, H. and Auer, S. (2005) 'Mapping XML to OWL Ontologies' in *Leipziger Informatik Tage 72*, Leipzig, Germany, pp. 147-156.

Bosch, T., Cyganiak, R., Wackerow, J. and Zapilko, B. (2012) 'Leveraging the DDI Model for Linked Statistical Data in the Social, Behavioural, and Economic Sciences'. Paper Presented at the DC-*2012 International Conference on Dublin Core and Metadata Applications*. 03-07 September 2012. Kuching, Sarawak, Malaysia.

Bosch, T. and Mathiak, B. (2013*) Evaluation of a Generic Approach for Designing Domain Ontologies Based on XML Schemas*. [online] Technical report, GESIS – Leibniz Institute for the Social Sciences, Mannheim. http://www.gesis.org/publikationen/gesis-technical-reports/ (Accessed 08 April 2013)

Bosch, T. and Mathiak, B. (2012) 'XSLT Transformation Generating OWL Ontologies Automatically Based on XML Schemas' in *IEEE Xplore Digital Library, 6th International Conference for Internet Technology and Secured Transactions, IEEE Xplore Digital Library*, pp. 660-667.

Bosch, T. and Mathiak, B. (2011) 'Generic Multilevel Approach Designing Domain Ontologies Based on XML Schemas' in *Proceedings of the Workshop Ontologies Come of Age in the Semantic Web, 10th International Semantic Web Conference, CEUR Workshop Proceedings*, Aachen, Germany, pp. 1-12.

Bosch, T., Wira-Alam, A. and Mathiak, B. (2011) ‘Designing an Ontology for the Data Documentation Initiative’ in *8th Extended Semantic Web Conference*.

Clark, J., Cowan, J., Fitzgerald, M., Kawaguchi, J., Lubell, J., Murata, M., Walsh, N. and Webber, D. (2003) ‘Information technology – Document Schema Definition Language (DSDL) – part 2: regular-grammar-based validation’ – *RELAX NG. ISO/IEC 19757-2:2003(E).*

Dell’Erba, M., Fodor, O., Ricci, F. and Werthner, H. (2002) ‘Harmonise: a solution for data interoperability’ in *Proceedings of the 2nd IFIP Conference on E-Commerce, E-Business, E-Government*, Lisbon, Portugal, pp. 433-445.

Ferdinand, M., Zirpins, C. and Trastour, D. (2004) ‘Lifting XML Schema to OWL’ in *Web Engineering - 4th International Conference*, Springer, Heidelberg, Germany, pp. 354-358.

Karlund, N., Moller, A. and Schwartzbach, M.I. (2000) ‚DSD: a schema language for XML’ in *ACM SIGSOFT Workshop on Formal Methods in Software Practice*, ACM, New York, NY.

Klein, M.C.A. (2002) ‘Interpreting XML documents via an RDF Schema ontology’ in *13th International Workshop on Database and Expert Systems Applications*, Springer, Heidelberg, Germany.

Kobeissy, N., Genet, M.G. and Zeghlache, D. (2007) ‘Mapping XML to OWL for seamless information retrieval in context-aware environments’ in *International Conference on Pervasive Services*, Istanbul, Turkey.

Kupfer, A., Eckstein, S., Störmann, B., Neumann, K. and Mathiak, B. (2007) ‘Methods for a synchronized evolution of databases and associated ontologies’. Paper Presented at the *2007 Conference on Databases and Information Systems IV*. 2007. Amsterdam, The Netherlands.

Murata, M., Lee, D., Mani, M. and Kawaguchi, K. (2005) ‘Taxonomy of XML schema languages using formal language theory’, *ACM Transactions on Internet Technology*, Vol. 5 No. 4.

O’Connor, M. J. and Das, A. K. (2011) ‘Acquiring OWL ontologies from XML documents’ in *Proceedings of the 6th International Conference on Knowledge Capture*, ACM, New York, NY.

O'Connor, M.J. and Das, A.K. (2010) ‘Semantic reasoning with XML-based biomedical information models’ in *13th World Congress on Medical Informatics*, Cape Town, South Africa.

Reif, G., Gall, H. and Jazayeri, M. (2005) ‘WEESA - web engineering for Semantic Web applications’ in *14th World Wide Web Conference*, ACM, New York, NY.

Tous, R., Garcia, R., Rodriguez, E. and Delgado, J. (2005) ‘Architecture of a semantic XPath processor. Application to digital rights management’ in *E-Commerce and Web Technologies: 6th International Conference, EC-Web*, Springer, Heidelberg, Germany, pp. 1-10.

Volz, R., Oberle, D., Staab, S. and Studer R. (2003) *OntoLiFT Prototype – WonderWeb: ontology infrastructure for the Semantic Web.* Technical Report. WonderWeb Deliverable D11

Notes

[1]http://lod-cloud.net/
[2]http://www.w3.org/XML/Datamodel.html
[3]http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/
[4]http://clarkparsia.com/pellet/
[5]http://www.ddialliance.org/
[6]https://github.com/boschthomas/PhD
[7]http://dublincore.org/documents/dces/
[8]http://dublincore.org/documents/dcmi-terms/
[9]http://rdf-vocabulary.ddialliance.org/discovery