Transforming XML Schema to OWL Using Patterns

Ivan Bedini, Christopher Matheus, Peter F. Patel-Schneider[1], Aidan Boran
Alcatel-Lucent Bell Labs,
Ireland (1. New Jersey (US))

Benjamin Nguyen
University of Versailles St-Quentin &
INRIA-Rocquencourt Project SMIS,
France

Abstract — One of the promises of the Semantic Web is to support applications that easily and seamlessly deal with heterogeneous data. Most data on the Web, however, is in the Extensible Markup Language (XML) format, but using XML requires applications to understand the format of each data source that they access. To achieve the benefits of the Semantic Web involves transforming XML into the Semantic Web language, OWL (Ontology Web Language), a process that generally has manual or only semi-automatic components. In this paper we present a set of patterns that enable the direct, automatic transformation from XML Schema into OWL allowing the integration of much XML data in the Semantic Web. We focus on an advanced logical representation of XML Schema components and present an implementation, including a comparison with related work.

Keywords-component; XML Schema, Ontology, transformation patterns, ontology design, automatic ontology generation.

I. INTRODUCTION

In the last decade, the formalism of eXtensible Markup Language (XML) [9] has reached consensus among most standards bodies, becoming the de facto standard format for data exchange. Several reasons motivated this choice, the first of them being that XML provides a format that is at the same time both human readable and machine interpretable. Another reason is its simplicity and suppleness of usage fits well with the most part of application information exchange requirements. Furthermore, the introduction of the Document Type Definition (DTD) and XML Schema (XS) [10] formalisms installed a clean separation between meta-data and instances containing the actual data to be exchanged. Nevertheless, XML still remains, in a certain sense, too open and let to an excess of dialects that tend to overload its basic usage and meanings. The more recent Web Ontology Language (OWL) [12], along with the Resource Description Framework (RDF) [17] on which it is based, has become another popular standard for data representation and exchange. Being able to translate XML Schema models to RDF/OWL ontologies through an automated process offers a significant advantage that can reduce the human work necessary when designing an ontology and the effort required to transform the Web into a Semantic Web.

In this paper we provide a pragmatic view of XML Schema practices based on a detailed analysis of Business to Business (B2B) standard specifications that, as shown in [3], describes a large fraction of the use of this technology. Our goal is to identify practical patterns for demonstrating how XML Schemas can be mined to extract ontological assertions automatically and to provide a concrete and implementable approach that improves existing systems. We show that it is not a simple process, but that this operation requires precise attention on design practices. Moreover we provide some considerations on how to best exploit the semantics given by XML Schema sources to provide labels composed by dictionary word as ontology entities names. After this first step, we present our implementation to validate our approach and we compare the resulting data transformations with those of other systems. Indeed, as we show, some systems can already derive an OWL ontology from XML Schemas. More often the ontology is obtained with ad hoc mapping of XS components either to OWL entities or to an intermediate data model. Rather than providing a closed set of mapping procedures, the approach we provide is based on pattern recognition. The 40 patterns we have defined are capable of mapping the most part of XS constructions by integrating several specific design practices. This behavior ensures a better interpretation of XML schema sources with the possibility of improving the derivation of the conceptual information handling exceptions. Our pattern-based system can also be extended simply by adding new patterns to fit other specific requirements.

This paper is organized as follows. First, we present a brief analysis of XML Schema design practices based on B2B standard specifications seen as XML sources. The next sections present XML components and detail 40 transformation patterns. Then we present the prototype we have developed to validate the approach. Afterwards we provide some elements to evaluate our transformations and compare our system and approach with other systems. Finally, we conclude this chapter with a discussion of future work and research directions.

II. B2B XML SCHEMA STANDARD SPECIFICATIONS

To study and test our approach we have collected a corpus of 25 B2B standard specifications composed by 3432 XS files containing more than 586.000 XML Schema components and among these tags at least 170.000 are named. (More detailed information can be found in [3].) Fig. 1 provides a global view of the use of XML Schema components we have considered. It clearly shows that standard bodies include a considerable amount of documentation. Moreover XS *element* and *attribute* are the most used components, while others like *union*, *all*, *any* and *substitutionGroup* are very seldom employed. Here again, the figure only provides a statistical measure of the component adoption and simply gives us a list of those components that should be included in the extraction of information from XML Schemas. The result of our analysis is a tailoring for the extraction operation to XML sources for the B2B domain. However even though it has not been proved yet we have

defined generic patterns and validated on some well defined schemas and we estimate that our choices can be applied to a wide set of XML Schema sources.

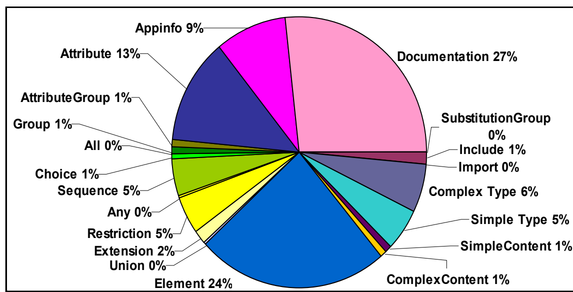

Figure 1. XML Schema components extraction

III. Related Work

With the establishment of XML and RDF/OWL several tools and methods already address the problematic of generating RDF/OWL files from XML based sources. Although many of them have a different main scope, they can be considered as part of the mapping of XML sources to ontology. COMA++ [1] has the main objective to provide several automatic matching algorithms and can produce an RDF output from this mapping. Although COMA++ can be considered as part of the mapping tools, but it does not consider specific XML Schema structures, relies on human intervention and result limited and poorly extensible to this scope. Similarly to COMA++, the approach in [4] has a different focus but permits to generate ontologies from XML sources. It targets the integration of heterogeneus data coming from different source formats. It is based on the Logical Data Model (LDM) ontology as a neutral representational format to represent incoming information from an external schema into their mapping environment. Their implementation is a rule based system that at some extent is comparable to our, but has the drawback to have the resulting ontology tailored on the LDM. Always related to our field, some works provide a tool that allow the automatic transformation of XML sources to existing OWL ontology. The process generally requires an initial task, provided by a user, aiming to define the mapping. Among them JXML2OWL [18] and XSPARQL [7]. These tools have a practical approach that permit to automatically transform XML sources to RDF/OWL format. The inconvenient is that they require the presence of a reference/target ontology and the user provided set of correspondences. Closer to our main scope there are XML2OWL [8] and OWLMAP [11]. However their approach is based on a close table of mapping, generally expressed with simple pairs (XML schema element, OWL entity) and implemented using XSL (XML Style Sheet) technology to perform a partial mapping. These approaches do not allow to extend the mapping. With respect to all cited works our approach and implementation present some innovative advantages. The consideration and improvement of semantics used to name OWL entities, to show a simple way to define specific interpretations of XML constructs to OWL 2, to be easily extensible and to provide a concrete and explicit OWL syntax as resulting mapping.

IV. Derivation of Logical Assertions from XML Schemas

As stated in [14], ontologies and XML schemata serve very different purposes. Ontology languages are a means to specify domain theories based on logical representation and XML schemata are a means to provide integrity constraints for information sources (i.e., documents and/or semi-structured data). It is therefore not surprising to encounter differences when comparing XML schema with ontology languages. However, XML schema and OWL ontologies have one main goal in common: both provide vocabulary and structure for describing information about data. Indeed it is simple to imagine equivalences between OWL classes and XS elements, like *Person* or *Employee* presented below in Listing 3, or even derive hierarchical information such as *rdfs:subClasseOf* between *Someone* and *Employee* and *owl:ObjectProperty* (like *hasLongitude* and *hasLatitude* for *Coordinate* in Listing 1). These simple equivalences between OWL and XS permit the provision of not only basic information for a target ontology, but also interesting properties and restrictions relating entities.

TABLE I. List of abbreviations/variables used in patterns

Abbreviation/Variable	Description
ct_name	Complex type name (e.g. *Person*)
st_name	Simple type name (e.g. *amount*)
nativeDataType	Represents any datatypes as defined in XML Schemas Part 2 [6] (e.g. *xsd:string* and *xsd:Boolean*)
basedDT	Data type on which the restriction/extension is based
has_*ct_name*	Object or datatype property given name adding the prefix 'has_' plus the name of the associated complex type (e.g. *has_coordinate*)
has_*st_name*	Object or datatype property given name adding the prefix 'has_' plus the name of the associated simple type (e.g. *has_monetaryAmount*)
ct_name_dt	Name of a datatype derived by a complex type (e.g. *author*)
elt_name	Name of the element (e.g. *Customer*)
elt_type	Name of the type of the element
Elt_name_Ct_name	Derived name for an OWL class composed by the name of the complex type name plus the element name (if different) (e.g. *Domiciliation_Address*)
has_*elt_name*	Object or datatype property given name adding the prefix 'has_' plus the name of the associated element
has_*elt_name_ct_name*	Object or datatype property given name adding the prefix 'has_' plus the names of associated complex type and element
attr_name	Name of the attribute
has_*attr_name*	Object or datatype property given name adding the prefix 'has_' plus the name of the associated attribute
has_*ct_name_attr_name*	Object or datatype property given name adding the prefix 'has_' plus the names of associated complex type and attribute
group_name	Name of the group
has_*group_name*	Object property given name adding the prefix 'has_' plus the name of the associated group
attr_group_name	Name of the attribute group component
attr_type	Name of the type of the attribute
lang_cd	Code of the language (e.g. *en, fr, it, …*)

A. Transformation Patterns

Patterns are used in many areas as "templates" or abstract descriptions encoding best practices of some field. In this section we provide a set of 40 patterns to transform XS constructs to the OWL2-RL profile [16] that help constructing

axiom-rich, formal ontologies, based on the identification of recurring patterns of knowledge in the XML Schemas, and then stating how those patterns map onto concepts in the ontology. From a modelling perspective, we are not designing specific logic representations; instead we are making an interpretation of XML constructs to obtain the maximum logical expression derivable from that formalism. In the interpretations below, selected XML patterns are written using XML Schema syntax, while RDF/OWL correspondences are expressed using Turtle syntax [5]. Table I presents all generic names used in pattern descriptions.

One must be aware that we work only on XS, thus we target TBox statements and we do not integrate XML instances (that may be better compared to ABoxes). Each sub-section below presents a short introduction of the XML components with their transformation patterns to OWL.

1) Simple and Complex Types

Simple and complex type components are defined and used only within the schema document(s). The XS *complexType* component is normally used to define components with child elements and/or attributes. The *simpleType* is used to create a new datatype that is a refinement of a built-in XS type (e.g., string, date, gYear, etc). In particular, we can derive a new simple type by restricting an existing simple type; in other words, the legal range of values for the new type is a subset of the existing type range of values. Type components can be anonymous (without name) when used locally for an element, but they must be named for a global definition. Listing (1.a) provides as example the definition of a global complex type *CoordinateType*. In addition to the so-called *atomic* types, simple types have also the concept of *list* and *union* types. Atomic types and list types enable an element or an attribute value to be one or more instances of one atomic type. In contrast, a union type enables an element or an attribute value to be one or more instances of one type drawn from the union of multiple atomic and list types. Listing (1.b) illustrates an example of a simple type declaration with *union* definition.

```
<xs:complexType name="GeographicalCoordinateType">      (1.a)
  <xs:sequence>
    <xs:element name="longitude" type="xs:string"/>
    <xs:element name="latitude" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="DispositionType">                  (1.b)
  <xs:union memberTypes="CriminalDispositionTypes xsd:string"/>
</xs:simpleType>
```

As we observed in XML Schema design practices analysed above all simple types are used in defining concrete datatype, i.e. binary predicates relating individuals with values. For this reason simple types are always mapped to rdfs datatype. Conversely, Complex types can be used to define a sort of composed datatype to which is added meta-data information, like author for a comment or detail of a specific code list used to define the data value, or again supplementary details on the data itself like the unit of measure. However, even in OWL there is no clear representation for such complex datatype. For this reason all named complex types here are directly mapped to OWL classes and further assertions and relations to datatypes are exposed in sections below. Table II presents transformation patterns for named simple and complex type.

TABLE II. SIMPLE AND COMPLEX TYPE TRANSFORMATION PATTERNS

#	XS	OWL
1	<simpleType name="st_name">	:st_name rdf:type rdfs:Datatype .
2	<simpleType name="st_name"> <union memberTypes="st_name1 xsd:nativDataType ..."/> </simpleType>	:st_name owl:equivalentClass :st_name1 ; owl:equivalentClass xsd:nativeDataType; owl:equivalentClass ... .
3	<complexType name="ct_name">	:Ct_name rdf:type owl:Class .

TABLE III. DERIVED TYPES TRANSFORMATION PATTERNS

#	XS	OWL
4	<simpleType name="st_name"> <restriction base="xsd:nativDataType"> <enumeration value="value1">…	:st_name owl:equivalentClass [rdf:type rdfs:Datatype; owl:oneOf ("value1"^^xsd:nativDataType ...)] .
5	<simpleType name="st_name"> <restriction base="basedDT"> <minInclusive value="value1"/> <maxInclusive value="value2"/> </restriction> </simpleType>	:st_name owl:equivalentClass [rdf:type rdfs:Datatype; owl:onDatatype :basedDT; owl:withRestrictions ([xsd:minInclusive "value1"^^:basedDT] [xsd:maxInclusive "value2"^^:basedDT])] .
6	<complexType name="ct_name"> <simpleContent> <extension base="xsd:nativeDataType"> …	:Ct_name rdf:type owl:Class . :has_ct_name rdf:type owl:DatatypeProperty ; rdfs:domain :Ct_name ; rdfs:range xsd:nativeDataType .
7	<simpleType name="st_name"> <restriction base="basedDT"> <minExclusive value="value1"/> <maxExclusive value="value2"/> </restriction> </simpleType>	:st_name owl:equivalentClass [rdf:type rdfs:Datatype; owl:onDatatype :basedDT; owl:withRestrictions ([xsd:minExclusive "value1"^^:basedDT] [xsd:maxExclusive "value2"^^:basedDT])] .
8	<complexType name="ct_name"> <simpleContent> <extension base="st_name"> …	:Ct_name rdf:type owl:Class . :has_ct_name rdf:type owl:DatatypeProperty ; rdfs:domain :Ct_name ; rdfs:range :st_name ; rdfs:subPropertyOf :has_st_name .
9	<complexType name="ct_name"> <simpleContent> <extension base="ct_name2"> (see #26, 27, 28)…	Same than #8 + :Ct_name rdfs:subClassOf :Ct_name2 .
10	<complexType name="ct_name"> <simpleContent> <restriction base="xsd:nativDataType"> (cf #4,5,6)…	:Ct_name rdf:type owl:Class . :ct_name_dt owl:equivalentClass [rdf:type rdfs:Datatype; (cf#4,5,6)] . :has_ct_name rdf:type owl:DatatypeProperty ; rdfs:domain :Ct_name ; rdfs:range :ct_name_dt .
11	<complexType name="ct_name"> <simpleContent> <restriction base="st_name"> …	Same than #10 + :has_ct_name rdfs:subPropertyOf :has_st_name .
12	<complexType name="ct_name"> <simpleContent> <restriction base="ct_name2"> …	Same than #11 + :Ct_name rdfs:subClassOf :Ct_name2 .
13	<complexType name="ct_name"> <complexContent> <extension base="ct_name2">…	:Ct_name rdf:type owl:Class ; rdfs:subClassOf :Ct_name2 .
14	<complexType name="ct_name"> <complexContent> <restriction base="ct_name2">…	:Ct_name rdf:type owl:Class ; rdfs:subClassOf :Ct_name2 .

2) Derived Types

XS provides two forms of sub-classing type components,

called **derived types**. The first form derives by **extension** from a parent complex type with more elements (i.e. properties for the ontology), while the second form can be obtained by **restriction** of the base type, creating a type as a subset. The restriction for simple types operates with the application of constraints on predefined simple types or with the help of regular expressions. Restriction of complex types is conceptually the same as restriction of simple types, except that the restriction of complex types involves a type's declarations rather than the acceptable range of values. A complex type derived by restriction is very similar to its base type, except that its instances are more limited than the corresponding declarations in the base type. Moreover, complex types can be constrained using the *complexContent* component that signals we intend to restrict or extend its content. While *simpleContent* component indicates that the content of the new complex type contains only simple data and no element. In other words, *simpleContent* provides a solution for adding attributes to simple types. Listing (2.a) illustrates an extension for a complex type using the simple content component to add more attributes to *DescriptionType*, which is defined as a string (not shown in the example). And (2.b) shows an example of simple type restriction. Although the two derivations are called extension and restriction, conceptually they both represent a possible restriction of the set of individuals of the base ontological entity. Indeed the extension just adds a property to a class, consequently only individuals having all these properties asserted belong to the "extension". As such, derived types generate either sub-classes or sub-properties of the base entity. Table III details most of the possible patterns for derived types.

```
<xs:complexType name="NoteType">                                    (2.a)
  <xs:simpleContent>
    <xs:extension base="DescriptionType">
      <xs:attribute name="author" type="StringType" use="optional"/>
      <xs:attribute name="status" type="StringType" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="CountryCodeType">                               (2.b)
  <xs:restriction base="xsd:token">
    <xs:enumeration value="FR"/>
    <xs:enumeration value="US"/>   …
  </xs:restriction>
</xs:simpleType>
<xs:element name="CountryCode" type="CountryCodeType"/>
```

3) XS Elements

Along with *attribute*, XSD *element* defines the tag syntax for XML documents. The element component allows the description of simple and complex entities. Elements can be declared via several different methods. Listing 3 shows three examples of them. The first one is a global element with a declared type. In the second example, the element is declared with an inline *complexType* and inline sub-elements. The final example illustrates an element declared with inline *simpleType* that refines an XS built-in data type.

```
Global declaration                                                  (3.a)
<xs:element name="MonetaryAmount" type="AmountType"/>
Local declaration                                                   (3.b)
<xs:element name="GeographicalCoordinate">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="longitude" type="xsd:string" minOccurs="1"/>
      <xs:element name="latitude" type="xsd:string" minOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
Inline anonymous simple type declaration                            (3.c)
<xs:element name="Amount">
  <xs:simpleType>
    <xs:restriction base="udt:amountType">...
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:complexType name="Provider">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="ServiceProvider" minOccurs="0"/>
      <xs:element ref="ContentProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<xs:element name="ServiceProvider" type="ServiceProvider"/>
<xs:element name="ContentProvider" type="ContentProvider"/>
```

TABLE IV. ELEMENT TRANSFORMATION PATTERNS

#	XS	OWL
15	<element name="elt_name" type="xsd:nativeDataType">	:elt_name rdf:type rdfs:Datatype ; owl:equivalentClass xsd:nativeDataType .
16	<element name="elt_name" type="st_name"/>	:elt_name rdf:type rdfs:Datatype ; owl:equivalentClass :st_name .
17	<complexType name="ct_name"> <sequence> <element name="elt_name" type="st_name">…	:has_elt_name rdf:type owl:ObjectDatatype ; rdfs:domain :ct_name ; rdfs:range :st_name .
18	<complexType name="ct_name"> <sequence> <element ref="elt_name"/> (referring to a simple type)…	:has_elt_name_ct_name rdf:type owl:ObjectDatatype ; rdfs:domain :ct_name ; rdfs:range :elt_name .
19	<element name="elt_name"> <simpleType> <restriction base="xsd:nativeDataType"> …	:elt_name owl:equivalentClass [rdf:type rdfs:Datatype;(cf #4,5,6)]. :has_elt_name rdf:type owl:DatatypeProperty ; rdfs:range :elt_name .
20	<element name="elt_name"> <simpleType> <restriction base="st_name"> …	:has_st_name rdf:type owl:DatatypeProperty ; rdfs:range :st_name . :elt_name owl:equivalentClass [rdf:type rdfs:Datatype; (cf #4,5,6)]. :has_elt_name rdf:type owl:DatatypeProperty ; rdfs:range :elt_name ; rdfs:subPropertyOf :has_st_name .
21	<element name="elt_name" type="ct_name"/> (global declaration)	:Elt_name rdf:type owl:Class ; rdfs:subClassOf :Ct_name .
22	<complexType name="ct_name"> <sequence> <element name="elt_name" type="ct_name1"/> …	:has_elt_name rdf:type owl:ObjectProperty ; rdfs:domain :ct_name ; rdfs:range :ct_name1 .
23	<complexType name="ct_name"> <sequence> <element ref="elt_name"/> (referring to a complex type)…	:has_elt_name_ct_name rdf:type owl:ObjectProperty ; rdfs:domain :ct_name ; rdfs:range :elt_name .
24	<element name="elt_name" type="xsd:nativeDataType ">	:elt_name rdf:type owl:DatatypeProperty ; rdfs:range xsd:nativeDataType .
25	<element name="elt_name1" type="elt_type"/> <element name="elt_name2" substitutionGroup="elt_name1"/>	:elt_name2 rdfs:subClassOf :elt_name1 .

Global elements and global types are element declarations

that are immediate children of the root <*schema*> element. Local elements, local types, and inline types are declarations that are nested within other elements or types. Although inline and local declarations result in a much more compact schema, they have the disadvantage of not being reusable. XML Schema specifications do not outline preferences to follow, but as general rule global declaration is often preferred to local and inline declarations. As illustrated in Listing 3.c a global element can be reused by other component definition simply using the *ref* declaration. This makes definition of elements and their use clearly separate, which is generally simpler to understand and reuse.

The wide use of XS element means that multiple correspondences with OWL entities exist (i.e. classes, object properties, datatype properties and even datatypes) and patterns must consider also the referred constructs of the element declaration. We first distinguish local and global declarations: using global declarations, we generally derive classes and datatypes while using local and inline declaration, we derive object and datatype properties. Moreover as mentioned above a local declaration cannot be reused by other elements in the XS source. Conversely a global declaration can be reused in different cases and thus can be used to define a more generic entity for the ontology. This brings us to a specific pattern for element *ref* declarations as presented in #18 and #23 of Table IV. In this pattern we distinguish the derived object property name from the element using a compound name composed of the element name itself plus the name of the component enclosing it. This is because in automated transformation the control over the uniqueness of the property cannot always be guaranteed and a double declaration of a property with two different domains can lead to a misleading design, where the resulting property domain would be formed by the intersection of two classes. Table IV details all defined patterns for the element declaration.

TABLE V. ATTRIBUTES TRANSFORMATION PATTERNS

#	XS	OWL
26	<attribute name="attr_name" type="xsd:nativDT"/>	:attr_name rdf:type rdfs:Datatype ; owl:equivalentClass xsd:nativDT.
27	<complexType name="ct_name"> <attribute name="attr_name" type="st_name"/> …	:has_attr_name rdf:type owl:DatatypeProperty ; rdfs:domain :Ct_name ; rdfs:range :st_name .
28	<complexType name="ct_name"> <attribute ref="attr_name"/> …	:has_ct_name_attr_name rdf:type owl:DatatypeProperty ; rdfs:domain :Ct_name ; rdfs:range :attr_name .

4) Attributes

The XML Schema *attribute* component is used to declare simple values for a given complex element (attributes cannot have child elements). Attributes can be declared locally, or by reference to a global declaration. For complete declarations, global or local, the type attribute is used when the declaration can use a built-in or pre-declared simple type definition. Otherwise an anonymous simple type is provided inline. Listing 4 shows an example of inline attributes declaration. We can also observe that at data content level, this definition of *GeographicalCoordinateType* and that one provided in Listing 3.b and 1.a are equivalent. Here again the XML Schema specification does not provide any recommendation about the usage of one declaration rather than another. Generally attributes are used to transmit metadata information, like an internal identifier or a specific detail on the value. For example, with a geographical coordinate it could be the specific coordinate system (e.g. Cartesian or polar). Nevertheless, in the practices we evaluated, attribute declarations are often preferred simply because of their lower verbosity in XML instances, thus reducing the size of large data sets.

```
<xs:complexType name="GeographicalCoordinateType">            (4)
  <xs:attribute name="longitude" type="xsd:string"/>
  <xs:attribute name="latitude" type="xsd:string"/>
</xs:complexType>
```

Table V details transformation patterns for the attribute components. Similarly to elements we distinguish global declaration from local and inline ones. The former produces a datatype while the two latter produce datatype properties.

5) Grouping XML entities

XML Schema enables groups of elements to be defined and named, using the *group* component that assembles several elements together. The same is done with the *attributeGroup* for grouping attributes of an item. Moreover the definitions of complex types are declared using sequences of elements that can appear in the document instance. XML Schema provides three different constructors to allow the definition of sub-elements sequences. *Sequence* corresponds to an order collection of typed sub-elements; *choice* groups element using an exclusive-or, i.e., only one of its children can appear in an instance; and *all* contains at most one of each element specified as sub-elements. As a result, all the elements in a group may appear once or not at all and they may appear in any order. Any element in a group can be restricted with occurrence indicators; or this *minOccurs* and *maxOccurs* are used to define how often an element can occur in an instance. The default value for these indicators is 1, which means that the element is required and can appear only once. Listing 5 illustrates the definition of *TelecomNumberType* complex type, where sub-elements can be either *FormattedNumber* or the ordered sequence of elements grouped by *TelecomNumberGroup*.

```
<xs:complexType name="TelcomNumberType">            (5)
  <xs:choice>
    <xs:element ref="FormattedNumber"/>
    <xs:group ref="TelcomNumberGroup"/>
  </xs:choice>
</xs:complexType>
<xs:group name="TelcomNumberGroup">
  <xs:sequence>
    <xs:element ref="InternationalCountryCode" minOccurs="0"/>
    <xs:element ref="NationalNumber"/>
  </xs:sequence>
</xs:group>
```

Table VI details the defined transformation patterns for grouping components. Pattern #31 tries to translate the exclusive-or carried by the *choice* component. Indeed, differently from XML Schemas, it is not possible to represent such integrity constraints on properties in OWL. Thus we have created a consistency check that arises if concurrent triples appear in the ontology. This has been done by creating fictitious complex class definitions using the *owl:onProperty* property restriction that enables the definition of a class as the set of all individuals that are connected via a particular property

to another individual. Thus we define a subclass for each element of the group with only one property of the complex component and declare all of these classes disjoint. By doing so we assert that we cannot have individuals with more than one property at once at the same time. Particular attention must be paid to pattern #38 to ensure compliancy with the OWL2-RL profile where the value of occurrences can be only 1.

TABLE VI. GROUPING COMPONENTS TRANSFORMATION PATTERNS

#	XS	OWL
29	<complexType name="ct_name"> <sequence> <element name="elt_name1" type="elt_type1"/> <element name="elt_name2" type="elt_type2"/> … </sequence> </complexType>	:has_elt_name1 rdfs:domain :Ct_name; rdfs:range :elt_type1 . :has_elt_name2 rdfs:domain :Ct_name; rdfs:range :elt_type2 .
30	<complexType name="ct_name"> <all> <element name="elt_name1" type="elt_type1"/> ... </all> </complexType>	:has_elt_name1 rdf:type owl:FunctionalProperty ; rdfs:domain :Ct_name ; rdfs:range [rdf:type owl:Restriction ; owl:onProperty :has_elt_name1 ; owl:onClass :Elt_type1 ; owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger] ….
31	<complexType name="ct_name"> <choice> <element name="elt_name1" type="elt_type1"/> <element name="elt_name2" type="elt_type2"/> </choice> </complexType>	:Elt_name_Ct_name1 owl:equivalentClass [rdf:type owl:Restriction ; owl:onProperty :has_elt_name1 ; owl:someValuesFrom :elt_type1] ; rdfs:subClassOf :Ct_name . :Elt_name_Ct_name2 owl:equivalentClass [rdf:type owl:Restriction ; owl:onProperty :has_elt_name2 ; owl:someValuesFrom :elt_type2] ; rdfs:subClassOf :Ct_name . [] rdf:type owl:AllDisjointClasses ; owl:members (:Elt_name_Ct_name1 :Elt_name_Ct_name2) .
32	<group name="group_name"> <sequence> <element name="elt_name1" type="elt_type1"/> <element name="elt_name2" type="elt_type2"/> </sequence> </group>	:Group_name rdf:type owl:Class . :has_elt_name1 rdf:type owl:ObjectProperty; rdfs:domain :group_name ; rdfs:range :elt_type1 . :has_elt_name2 rdf:type owl:ObjectProperty; rdfs:domain :Group_name ; rdfs:range :elt_type2 .
33	<complexType name="ct_name"> <sequence> <group ref="group_name"/> </sequence> </complexType>	:has_group_name rdf:type owl:ObjectProperty ; rdfs:domain :Ct_name ; rdfs:range :Group_name .
34	<attributeGroup name="attr_group_name"> <attribute name="attr_name1" type="attr_type1"/> <attribute name="attr_name2" type="attr_type2"/> </attributeGroup>	:Attr_group_name rdf:type owl:Class . :has_attr_name1 rdf:type owl:ObjectProperty; rdfs:domain :Attr_group_name ; rdfs:range :attr_type1 . :has_attr_name2 rdf:type owl:ObjectProperty; rdfs:domain :Attr_group_name ; rdfs:range :attr_type2 .
35	<complexType name="ct_name"> <attributeGroup ref="attr_group_name"/> </complexType>	:has_attr_group_name rdf:type owl:ObjectProperty ; rdfs:domain :Ct_name ; rdfs:range :attr_group_name .
36	<complexType name="ct_name"> <sequence> <element name="elt_name" minOccurs="value1"/> </sequence> </complexType>	:Ct_name owl:equivalentClass [rdf:type owl:Restriction ; owl:minCardinality "value1"^^xsd:nonNegativeInteger ; owl:onProperty :has_elt_name] .
37	<complexType name="ct_name"> <sequence> <element name="elt_name" maxOccurs="value2"/> </sequence> </complexType>	:Ct_name owl:equivalentClass [rdf:type owl:Restriction ; owl:maxCardinality "value2"^^xsd:nonNegativeInteger ; owl:onProperty :has_elt_name] .
38	<complexType name="ct_name"> <sequence> <element name="elt_name" minOccurs="valueX" maxOccurs="valueX"/> </sequence> </complexType>	:Ct_name owl:equivalentClass [rdf:type owl:Restriction ; owl:cardinality "valueX"^^xsd:nonNegativeInteger ; owl:onProperty :has_elt_name] .

6) *Annotations*

XML Schema provides three elements for annotating schemas for the benefit of both human readers and applications. One is basic schema description information, the *documentation* component, which is the recommended location for human readable material. The second is *appinfo* component that can be used to provide information for tools, style-sheets and other applications. Both *documentation* and *appinfo* appear as sub-elements of *annotation*, which may itself appear at the beginning of most schema constructions. Table VII provides two elementary transformation patterns for XS annotations.

TABLE VII. ANNOTATIONS TRANSFORMATION PATTERNS

#	XS	OWL
39	<xs:element name="elt_name" type="elt_type"> <xs:annotation> <xs:documentation xml:lang="lang_cd" source="anyURI"> Text of the comment </xs:documentation> </xs:annotation> </element>	:Elt_name rdfs:comment "xsd:documentation Text of the comment"@ lang_cd ; rdfs:seeAlso "anyURI"^^xsd:string .
40	<xs:element name="elt_name" type="elt_type"> <xs:annotation> <xs:appinfo> App info annotation </xs:appinfo> </xs:annotation> </element>	:Elt_name :appinfo "xsd:appinfo text of the annotation"^^xsd:string .

B. *Pattern Recognition Limitations*

Because OWL is generally more expressive than XML Schema, it is not possible to derive a direct transformation pattern for each OWL logical construct. As an example it is not possible to derive automatically a pattern from XML Schema for describing binary relations like inverse, transitive and symmetric properties. The same is true for other construct like *owl:differentFrom*, *owl:NegativePropertyAssertion* and *owl:PropertyChainAxiom*. However, neither is it possible to convert all XML Schema integrity constraints into OWL, as there are areas where XML Schema is more expressive than OWL. For example pattern and length constraints on data values, (e.g. *<pattern value="[a-z] [a-z] [0-9]"/>, <xs:length value="8"/>, <xs:minLength value="5"/>, <xs:maxLength value="8"/>*), have no direct mapping into OWL.

C. *Deriving Names for OWL Entities*

Observed sources have different practices concerning naming conventions that are not always straightforward or adequate for naming ontology entities. For example, XML tags are often compound words that can be expressed using the common *Camel Case* convention with known terms (that we also call **dictionary terms**), like *OfficeLocation*, or using abbreviations to reduce XML tags size like *amt_ccy* (which might stand for *amount currency*). In addition, tags can contain compound words (like *cash-flow*), acronyms, bad spelled words, no separator between terms (like *foodservice*), specific terms, words unrelated to the meaning of the element (like *UnitOfMeasureBBIECommonData*), etc…. This is an important feature to analyse in the automatic derivation of an ontology. Indeed it is our strong belief that an ontology must have correct semantics for naming entities. Concerning XML type declarations we observed that almost all of them contain a prefix or a suffix indicating that it is a type. In our analysis we dropped all of them with the following patterns: *_type, _Type, _TYPE, type, Type, TYPE, _tp, _Tp, _t, _T, type_, TYPE_, t_, T_*. Another design practice we observed is that although elements and attributes define the tag syntax for XML documents, the better semantic can often be extracted from related complex and simple types.

1) *Input Sources Semantics and Structures*

"*Garbage In, Garbage Out (GIGO)*" is a well-known concept in computer science. Computers will unquestioningly process the most absurd of input data and produce absurd output. Indeed, in automated processes the quality of the output is directly dependent on the definition of input elements. Therefore we classify sources in **semantically well structured XML documents** to decide on its pertinence. We say that a concept *c* derived from an XML Schema source is **semantically valid** if its label is composed of clearly identifiable words belonging to a standard common dictionary. Along the same line we say that a set of extracted concepts *C* is **well structured** if the ratio between obtained properties (*#Rs*) and the total number of extracted concepts (*#C*) is higher than a predefined threshold (α). This last definition prevents the integration of only flat definitions of XML elements. For example, applying this test we were able to discard some XBRL (eXtensible Business Reporting Language) files. Indeed their specifications are defined with the help of XLink constructs that our system was not able to manage. Finally, following the definitions above we say that a non empty set of concepts *C*, obtained from a given source, is **semantically well structured** if at least a considerable number of its concepts are semantically valid (on the basis of a predefined threshold β).

V. System Implementation

To validate our approach we have developed a prototype called Janus that aims to generate an ontology from a large source corpus of XML Schemas, like the one presented above concerning the B2B domain. This system integrates sources incrementally and produces an intermediary conceptual model compatible with OWL to add flexibility when matching and integrating heterogeneous representations and semantics. However in this paper we only present a simplified version of the prototype that considers only one schema at a time and directly produces an ontology following the most of the defined transformation patterns presented above. The system has been implemented in Java, uses SAX for parsing XS documents and leverages the OWL-API [13] to handle the creation of OWL ontologies. The overall architecture of the system is presented in Fig. 2. The first step of the process parses XS sources and, implementing specific java algorithms, recognises selected patterns and stores candidate ontology entities in an organized internal data model based on multiple hash-tables. The second step aims to normalize extracted labels and produces correct semantics integrating different dictionaries. The English dictionary is based on WordNet [15] version 3.0 using JWNL. Other dictionaries are specific lists of abbreviations and acronyms, stop-words, *compound words* and the so-called *useless words* that we have been tailored for the B2B domain. All these specific dictionaries are based on either Java property files or simple text files that can easily be replaced and adapted. The following filtering step aims to eventually identify more abbreviations not detected previously and to purge sources that do not produce a semantically well structured output, as explained above. The last step simply translates the normalised internal data model into OWL.

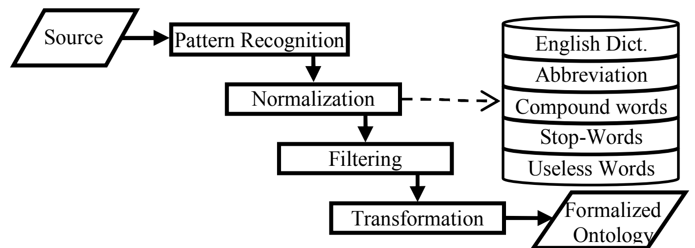

Figure 2. Overall architecture of the implementation

VI. Comparison and Transformation Evaluation

We have compared and evaluated our system with three other similar works. These are XML2OWL [8], OWLMAP [11] and LDM [4]. Mainly our analysis, summed up in Table VIII, highlights the following aspects of the different systems: *Number of XS constructs*, that evaluates the completeness of the map; *XML instances*, which seeks if the resulting ontology contains individuals; *Extensibility* indicates if the system can be extended; *Exception management* specifies if a system is able to look beyond the simple direct mapping and manage exceptions of specific design practices; *Semantic normalisation* evaluates the capacity of the system to resolve linguistic and semantic normalisations; *Concept structures* evaluates the capacity to resolve hierarchical, properties and datatype relations; *Concept relations* provides a quality measure about the richness of semantic relations extracted; *OWL expressivity* is a theoretical interpretation of the retrieved information expressivity using the DL naming convention [1] (the corresponding value is an evaluation we made on the basis of the available documentation). Table IX details the XML Schema components that are considered for the information extraction of each system. As we can see, our system improves existing solutions and only LDM is also able to provide advanced patterns.

TABLE VIII. XML SCHEMA INFORMATION EXTRACTION CONSIDERATIONS

	XML2OWL	OWLMAP	LDM	Janus
N. of XS construct	8	9	18	19
XML instances	✓	✓		
Extensible			✓	✓
Exception management	limited	limited	✓	✓
Semantic normalisation				✓
Concept structures	limited	✓	✓	✓
Concept relations	limited	✓	limited	✓
OWL expressivity	ALUHN	tbd	tbd	ALHONQF(D)

TABLE IX. DETAILS ON THE EXTRACTED XS CONSTRUCTS FOR THE TRANSFORMATION TO ONTOLOGY

[XS construct]	XML2OWL	OWLMAP	LDM	Janus
All	✓	✓	✓	✓
Annotation			✓	
Any			✓	✓
Appinfo				
Attribute		✓	✓	✓
AttributeGroup		✓	✓	✓
Choice	✓	✓	✓	✓
ComplexContent				✓
ComplexType	✓	✓	✓	✓
Documentation				
Element	✓	✓	✓	✓
Extension		✓	✓	✓
Group		✓	✓	✓
Import			✓	✓
Include			✓	✓
Restriction		✓	✓	✓
Sequence	✓	✓	✓	✓
SimpleContent				✓
SimpleType	✓	✓	✓	✓
SubstitutionGroup		✓		✓
Union			✓	✓
List			✓	
Min/Max Occurs	✓	✓	✓	✓
Namespace		✓		

VII. CONCLUSION AND FUTURE WORK

This paper presented our contribution on the transformation of XML schemas into RDF/OWL. The system implemented was shown foremost to be more complete than others, and in particular greatly improves on the number of complex transformation patterns with respect to similar approaches like XML2OWL. It is also much simpler in its design and implementation yet equivalent in transformation capacity to other systems implementing more complex approaches with the integration of intermediary and dedicated models, like LDM. We also provide important elements and guidelines to transform XML Schema sources, based on an initial set of 40 transformation patterns formalised using direct OWL syntax in Turtle format. We show that it is possible to mine XML Schema sources to extract enough knowledge to build semantically correct ontologies with considerable expressivity. Furthermore we have provided a way to define transformation patterns that can be easily implemented by any system and simply extended to augment the number of XML components and construct to be included. In future work we aim to improve the number of patterns to be able to map other specific constructs. With respect to the implementation, the prototype can be improved and built as a reusable API to be easily integrated within other systems. Further research is still needed to improve the capacity to detect well-formed sources and semantics. Furthermore the real challenge of these kinds of applications still remains the extraction from multiple sources and the incremental integration of new sources. Indeed although our approach and implementation are simple to realise, as such, they have limited capacity in integration and merging. However our implementation offers a concrete contribution to the automatic generation of ontologies from XML Schemas. It reduces the effort to convert the Web into a Semantic Web, empower the data integration and can provide a useful input to more complex and ambitious systems.

REFERENCES

[1] Aumueller, D., Do. H., Massmann, S., & Rahm, E. Schema and ontology matching with COMA++. In Proceedings of International Conference on Management of Data 2005. ACM Press, p. 906-8

[2] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. The Description Logic Handbook: Theory, Implementation, Applications. Cambridge University Press, 2003.

[3] Bedini, I. Deriving ontologies automatically from XML Schemas applied to the B2B domain. Doctoral dissertation, University of Versailles, France. January, 2010. Retrieved April 15, 2011 http://bivan.free.fr/Janus/Docs/PhD_Report_IvanBedini.pdf

[4] Beneventano, D., Ed. Semantic and ontology language specification. STASIS Project Deliverable 2.3.2, Version 10. 2008.

[5] Beckett, D., & Berners-Lee, T. Turtle - Terse RDF Triple Language. W3C Team Submission, 14 January 2008.

[6] Biron, P.V., & Malhotra, A. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004.

[7] Bischof, S., Lopes, N., & Polleres, A. Improve Efficiency of Mapping Data between XML and RDF with XSPARQL. In Proceedings of RR 2011.

[8] Bohring H, & Auer S. Mapping XML to OWL Ontologies. Leipziger Informatik-Tage. 2005. Pages: 147-156.

[9] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., & Yergeau, F. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008.

[10] Fallside, D.C., & Walmsley, P. XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004

[11] Ferdinand, M., Zirpins, C., & Trastour, D. Lifting XML Schema to OWL. In Proceedings of ICWE 2004, pages: 354-358.

[12] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., & Rudolph, S. OWL 2 Web Ontology Language Primer. W3C Recommendation 27 October 2009.

[13] Horridge, M., Bechhofer, S. The OWL API: A Java API for Working with OWL 2 Ontologies. 6th OWLED 2009, Chantilly, Virginia.

[14] Klein, M.C.A., Broekstra, J., Fensel, D., Van Harmelen, F., Horrocks, I. Ontologies and Schema Languages on the Web. Spinning the Semantic Web, 2003, pp: 95-139.

[15] Miller, G.A. WORDNET: A lexical database for English. Communications of ACM (11), 39-41. 1995.

[16] Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. OWL 2 Web Ontology Language Profiles. W3C Recommendation 2009.

[17] Manola, F., & Miller, E. RDF Primer. W3C Recommendation 10, 2004

[18] Rodrigues, T. , Rosa, P., & Cardoso, J. Moving from syntactic to semantic organizations using jxml2owl. Computers in Industry, vol. 59, no. 8, pp. 808–819. 2008.