Reverse Engineering of XML Schema to a Conceptual Schema

Ronaldo dos Santos Mello


ronaldo@inf.ufsc.br

Abstract. XML Schema is the actual W3C recommendation for defining the schema of XML documents. Reverse engineering of XML Schema to a conceptual schema is necessary to support several activities, like semantic integration of XML data and XML database re-design. This paper presents a rule-based process for converting a schema in XML Schema to a conceptual schema in the ORM/NIAM model. Different from related work, this process performs a detailed analysis of the XML Schema model to define suitable mappings to the conceptual model. The output conceptual schema is stored in an OWL catalog, which provides the structural basis of a domain ontology for the XML data.

1. Introduction

The XML format has been extensively used for data storage by information systems and people in several application domains. XML documents are kept in many kinds of repositories like XML databases [7] and Web data sources (examples are academic bibliographic repositories [1, 2]). In these contexts, reverse engineering techniques for XML data are required in many activities, like semantic integration of XML data sources, XML-based information extraction and ontology construction, as well as XML database re-design.

There are several approaches related to the problem of reverse engineering of XML schemata. Some of them work on a graph-based data model or a DTD (Document Type Definition) schema for XML documents [10, 11, 12, 17, 19]. In the first case, we have a generic logical model, and the conversion process does not address the particularities of the XML data representation. In the second case, we have a (an already old) W3C recommendation with a small set of resources for XML document structuring [3].

XML Schema is the actual W3C recommendation for schema specification of XML data, providing additional facilities with respect to a DTD, like data types and type extension [3]. Some approaches consider the conceptual abstraction of schemata in XML Schema [9, 16, 21, 22, 23], but no one defines a robust conversion process, e.g., a process that take all features of the XML Schema model into consideration during the conversion.

Based on these limitations, this paper presents a reverse engineering approach that performs a detailed analysis of the XML Schema model in order to convert an schema in XML Schema to a conceptual schema in the ORM/NIAM model. This approach is based on a set of rules that deals with each XML Schema construct, and a conversion
algorithm that executes these rules and produces a corresponding conceptual schema. Mapping expressions are generated during the conversion and associated to entities and relationships in the conceptual schema.

The proposed approach acts as a semi-automatic process that takes as input an XML Schema document and a set of related XML documents. It considers user intervention as well as the support of terminological databases in order to define a semantically precise conceptual schema. The output schema is kept in an OWL catalog, which provides the structural basis of a domain ontology for the related XML data. This conceptual (or ontological) schema may be further used, for example, by a Web query system that execute queries over a set of XML data that respects the same schema.

This paper is organized as follows. Section 2 introduces the considered data models in the reverse engineering approach. Section 3 gives an overview of the reverse engineering process. Section 4 defines the conversion rules and discusses the conversion algorithm and mapping information. Section 5 describes the persistence strategy for the output conceptual model. Section 6 analyses related work and section 7 is dedicated to the conclusion.

2. Input and Output Data Models

2.1. XML Schema Model

The model of XML Schema comprises the following concepts: element, attribute and type. Figure 1 (a) shows an XML Schema document that exemplifies the definition of these concepts. Elements define concrete hierarchical data structures in an XML document. An element is composed by a start-tag, a content model and an end-tag. The content model defines what is enclosed between the start-tag and the end-tag. An attribute describes a property of an element. Its value is specified at the start-tag of the element.

An element may be simple, complex, empty, mixed or anytype. An element is simple if its content model defines an explicit atomic data type or a list or union of atomic data types. In Figure 1 (a), ID (line 4), Street (line 15) and Salary (line 24) are examples of simple elements. An element is complex if its content model is defined by attributes and/or component elements¹. If a complex element has component elements, its content model is defined by one of the following constructs: sequence (child elements order is required); choice (only one child element is allowed in a group of child elements) or all (all child elements must appear once). Address (line 13) and Affiliation (line 32) are examples of complex elements. An element is empty if it has no content model. A mixed element allows a content model composed by elements and unstructured text, and an anytype element has a free content model. Curriculum (line 20), Fiction (line 45) and Body (line 47) are examples of mixed, empty and anytype elements, respectively.

The content model of an attribute behaves exactly like a simple element, e.g., it

¹A component element is an internal element definition or a reference to an external element definition.
may hold a single or a group of atomic values. Examples of attributes are hobbies (line 10) and ISBN (line 49).

A type is an abstract definition of the content model of one or more elements or attributes. A simple type specifies the domain of a simple element or attribute, and a complex type defines the structure of a complex element. Person (line 2) is an example of a complex type.

It is also possible to derive an element or type content model from another type or element definition. In a derivation by extension, an element or type reuses the content model of other type or element and specifies new properties. In a derivation by restriction, one or more reused properties are restricted to more limited domains. The element Author (line 27) is an example of an element derived by extension from Person type. Besides type definition, another reuse mechanism is the group definition. A group speci-
fies the content model of an element or a group of attributes. If a group is a substitution group, it specifies alternative representations for a given element.

Formally, we define a schema in XML Schema as a tuple <\(N, T, E_c, E_s, E_0, E_A, At, P, O, H, D\)>, where \(N\) is the schema name; \(T\) is the set of complex types\(^2\); \(E_c, E_s, E_0\) and \(E_A\) are the sets of complex and simple elements, as well as empty and anytype elements without attributes, respectively; \(At\) is the set of attributes; \(P\) is the set of element-attribute relationships; \(O\) is the set of complex element constructs; \(H\) is the set of ordered hierarchical relationships between elements; and \(D\) is the set of derivation relationships.

A \(t \in T \cup E_0\) is a tuple \(<n, \text{empty}/\text{type}>_\), where \(t.n\) is the type/empty element name. An \(x \in E_c\) is a tuple \(<n, [\text{st}]>_\), where \(x.n\) is the complex element name and \(x.st\) is the optional element category, with \(st \in \{\text{anytype}, \text{mixed}, \text{empty}\}\). An \(y \in E_s \cup At\) is a tuple \(<n, dt, [l], [u], [enum]>_\), where \(y.n\) is the simple element/attribute name; \(y.dt\) is the data type; \(y.l\) is the optional list of data types; \(y.u\) is the optional union of data types; and \(y.enm\) is the optional enumeration of allowed values for the element/attribute.

A relationship \(p \in P\) is a tuple \(<e, a>_\), with \(p.e \in E_c \cup E_s \cup T\) and \(p.a \in At\). A relationship \(h \in H\) is a tuple \(<t, \text{minOccurs}, \text{maxOccurs}, \{\text{ref}\}>_\), where \(h.\text{minOccurs} \in \{0, 1\}\), \(h.\text{maxOccurs} \in \{1, \text{unbounded}\}\) (relationship cardinalities), and with \(h.t, h.\text{ref} \in E_c \cup E_s \cup E_0\), being \(h.t\) the target (component) element, and \(h.\text{ref}\) an optional referenced element name. A construct \(o \in O\) is a tuple \(<s, t, sh>_\), where \(o.s\) is the source (composite) element or type, \(o.t\) is the construct category (sequence, for example), and \(o.sh\) is the construct content, with \(o.s \in E_c \cup T\) and \(o.sh \subseteq H\). A relationship \(d \in D\) is a tuple \(<g, s>_\), where \(g\) and \(s\) denotes, respectively, the generic and the derived relationship members, with \(d.g, d.s \in T \cup E_c\).

2.2. ORM/NIAM Model
We adopt a graphical variant of the ORM/NIAM (Object with Roles Model/Natural language Information Analysis Method) as the output conceptual model \([8]\). Figure 1 (b) shows an example of an ORM/NIAM schema. An ORM/NIAM schema is defined as a tuple \(<N, NL, L, A, I, ME>_\), where \(N\) is the schema name, \(NL\) is the set of non-lexical concepts; \(L\) is the set of lexical concepts; \(A\) and \(I\) are the sets of, respectively, association and inheritance relationships between two concepts; and \(ME\) is the set of mutually exclusive relationship sets.

A lexical concept models information with an associated value. A lexical concept \(l \in L\) is a tuple \(<n, d, [e]>_\), where \(l.n\) is the concept name; \(l.d\) is the concept data type (string or integer, for example); and \(l.e\) is an optional enumeration of allowed values for the concept. City is an example of lexical concept in Figure 1 (b). It is represented by a dotted rectangle\(^3\).

A non-lexical concept models information composed by other information. A

\(^2\)We do not define simple types, because they are not considered by the conversion rules.
\(^3\)Data types are omitted for sake of schema understanding.
concept $nl \in NL$ is a tuple $<n>$, where $n$ is the concept name. Book is an example of non-lexical concept in Figure 1 (b). It is represented by a solid rectangle.

An association relationship $a \in A$ is a tuple $<c_1, c_2, card_d, card_i, [n]>$, where $a.c_1$ and $a.c_2$ are the names of the associated concepts, with $c_1 \in NL$ and $c_2 \in NL \cup L$; $a.card_d$ and $a.card_i$ are, respectively, the direct cardinality (from $a.c_1$ to $a.c_2$) and the inverse cardinality (from $a.c_2$ to $a.c_1$) of the relationship; and $a.n$ is the optional relationship name, which describes the semantic intention (role) of the relationship. An inheritance relationship $i \in I$ is a tuple $<c_1, c_2>$, where $i.c_1$ is the name of generic concept and $i.c_2$ is the name of specialized concept, with $c_1, c_2 \in NL$. A set of mutual exclusion relationships $me \subseteq ME$ is a tuple $<D>$, with $me.D \subseteq A \cup I$.

Examples of association relationships with roles occur between Person and Address (home and office roles). The relationship between Person and Author is an example of inheritance relationship. Mutually exclusive relationships are shown in the relationships that connect Affiliation to University and Company.

The ORM/NIAM model was chosen as the output conceptual model because it has a more straight correspondence with the XML Schema model: non-lexical concepts are suitable to model complex elements, and lexical concepts are suitable to model simple elements and attributes. Besides, simple elements and attributes (valued information) may be associated to several complex elements in XML Schema. This modeling capability is possible in ORM/NIAM, e.g., a lexical concept may have relationships with several non-lexical concepts. Other conceptual models usually do not support this capability. In the EER model [6], for example, valued information can only be modeled as an attribute, which is an exclusive property of an entity or relationship.

3. Reverse Engineering Process Overview

The reverse engineering process comprises three semi-automatic phases, as shown in Figure 2. It extends a previous work dedicated to the conceptual abstraction of DTDs [13]. The input of the process is an XML Schema document and a set of valid XML documents. The output of the process is an ORM/NIAM conceptual schema.

The first phase is called Preprocessing. It modifies the definition of the XML Schema document in order to generate a more well-structured and simplified schema to be further converted. We have basically the following tasks:

1. inclusion of external schema definitions: it inserts schema specifications defined outside that are referenced by the XML Schema document. After that, the complete structure of types, elements and attributes are available for processing;
2. removal of irrelevant data: it excludes schema components that are not semantically relevant in a conceptual abstraction of data. One example could be an Author-list element as a component of an element Book, acting as an intermediate element between Book and Author elements. This task is usually
accomplished by a domain expert user, but the process may suggest some candidate schema structures (like intermediate elements) to be removed;

3. replacement of simple types and groups: it replaces simple types as well as group definitions in the content model of elements and attributes that reference them. The purpose here is to simplify the analysis of the elements and attributes schemata during the conversion;

4. reestructuring of anonymous nested structures: it converts a repeated or optional nested structure of child elements, in the content model of a complex element, into a new complex element called virtual element. Such task also simplifies the definition of concepts and relationships at a conceptual level. One example could be a nested repeated sequence (street, number, city, zipCode) into a complex element Person. It could be better positioned into a virtual element Address, that would be referenced by Person;

5. renaming of schema components: an expert user changes the name of types, elements and attributes in order to clarify their semantic intention.

The second phase is called Conversion. It takes a preprocessed XML schema document and applies a set of conversion rules on it, generating a preliminary conceptual schema and mapping information. This phase is the focus of this paper, being detailed in the next section. Finally, the Reestructuring phase takes a preliminary conceptual schema and performs manual and automatic modifications on it to produce a more semantically correct and simplified conceptual schema (a definitive conceptual schema). An example of manual modification is the validation of default relationship cardinality constraints. An example of automatic modification is the removal of redundant relationships.
4. Conversion Rules

4.1. Rules Definition

The conversion rules give support to the mapping of all XML Schema constructs to ORM/NIAM concepts during the Conversion phase. Basically, all atomic content are mapped to lexical concept. Atomic content holds elements and attributes with atomic data types; list components; and non-structured (textual) parts of mixed elements. Non-lexical concepts correspond to types, elements or attributes that hold a complex structure.

Hierarchical and reference relationships, as well as relationships of elements or types with atomic contents, are mapped to association relationships in ORM/NIAM. The rules that generate association relationships execute auxiliary functions that determine the relationship cardinality constraints. The direct cardinality (composite→component direction) is based on the component cardinality in the composite element definition. The inverse cardinality (composite←component direction) is deduced from an analysis of instances in XML documents. Such analysis tries to discover if a same component instance is associated to more than one different composite instance or not. If a conclusive answer is not obtained, a default ”(1,N)” cardinality is set.

Inheritance relationships are also considered for type-type and type-element extensions. Besides, inheritance relationships may be inferred between elements with the aid of terminological databases. We consider thesauri and domain-specific bases of terms as being terminological databases.

The conversion rules are defined in the following. Figure 1 exemplifies them.

Definition 1: Rule CC (Complex Content Conversion). A $cc \in T \cup E_c$ generates $nl \in NL$, with $nl = <cc.n>$.

Type Person in Figure 1 (a) (line 2) is converted by Rule CC into a non-lexical concept with the same name in Figure 1 (b). Examples of complex element conversion are Author (line 27) and Book (line 38).

Definition 2: Rule SC (Simple Content Conversion). An $sc \in A \cup E_s$ generates:

$$
\begin{align*}
\begin{cases}
  \bullet nl \in NL, & \text{with } nl = <sc.name>, \\
  \quad \text{if } sc.u \text{ is not null;} \\
  \bullet l \in L, & \text{with } l = <sc.n, sc.dt, sc.enum>, \\
  \quad \text{otherwise.}
\end{cases}
\end{align*}
$$

Rule SC usually generate lexical concepts to simple elements and attributes, like ZipCode (line 17) and ISBN (line 49) conversion, respectively. A non-lexical concept $NL_i$ is created only if an union is specified. In this case, lexical concepts are defined additionally to each possible atomic data type, and further connected to $NL_i$ as mutual exclusive association relationships (see Rule U in the following). The conversion of the

---

4We consider an union of atomic data types as a complex structure.
simple element ID (line 4) is an example. The intention here is to consider each possible disjoint atomic value of a property as a concept in the ORM/NIAM schema. No similar treatment is given to a list simple element or attribute since it holds a set of values of a same data type. In this case, maximal direct cardinality is set to $N$ to denote this multivalued association. Hobbies attribute conversion (line 10) is an example.

The next four rules deal with relationship generation in the conceptual schema.

**Definition 3: Rule HC (Hierarchical Relationship Conversion).** Let $e_s \in T \cup E_c$, $e_t \in E_c \cup E_s$, $o \in O$ with $o.s = e_s$. An $h \in H$, with $h.t = e_t$ and $h \subseteq o.sh$ generates:

- $i \in I$, with $i = \langle e_s.n, e_t.n \rangle$,
  - if $h.maxOccurs = 1$ and it is possible to infer that $e_s.n$ is a generic term than $e_t.n$ with the aid of a terminological database;
- $a \in A$, with $a = \langle e_s.n, e_t.n, card_i, card_i, null \rangle$,
  - otherwise.

**Definition 4: Rule HR (Hierarchical-Repeated Relationship Conversion).** Let $e_s \in T \cup E_c$, $e_t \in E_c \cup E_s$, $o \in O$ with $o.s = e_s$. A set $\{h_1, ..., h_n\} \in H$, where $\forall h_i \in \{h_1, ..., h_n\}$

$\Rightarrow h.t = e_t$, and $\{h_1, ..., h_n\} \subseteq o.sh$ generates:

- $a \in A$, with $a = \langle e_s.n, e_t.n, card_i, card_i, null \rangle$,
  - if all $h_i \in h_1, ..., h_n$ have the same semantic intention (role);
- $a_1, ..., a_m \in A$, $m \leq n$, with $a_i = \langle e_s.n, e_t.n, card_i, card_i, role_i \rangle$ and $a_i \in \{a_1, ..., a_m\}$
  - if there are $m$ roles in the $e_s$-$e_t$ hierarchical relationships.

**Definition 5: Rule RH (Referenced Hierarchical Relationship Conversion).** Let $e_s \in T \cup E_c$, $e_t \in E_c \cup E_s$, $o \in O$ with $o.s = e_s$. An $h \in H$, with $h.t = e_t$, $h \subseteq o.sh$ and $h.ref$ is not null generates:

- $i \in I$, with $i = \langle e_s.n, h.ref \rangle$,
  - if $h.maxOccurs = 1$, $h.ref = x.n$, with $x \in E_c$, and it is possible to infer that $e_s.n$ is a generic term than $h.ref$ with the aid of a terminological database;
- $i \in I$, with $i = \langle h.ref, e_s.n \rangle$,
  - if $h.maxOccurs = 1$, $h.ref = x.n$, with $x \in E_c$, and it is possible to infer that $h.ref$ is a generic term than $e_s.n$ with the aid of a terminological database;
- $a \in A$, with $a = \langle e_s.n, h.ref, card_i, card_i, e_t.n \rangle$,
  - otherwise.

Rule HC and Rule RH generate association relationships for hierarchical relationships between a complex element or type and its (referenced or internally defined) child elements. The relationship between Address and City in Figure 1 (b) is generated by Rule HC (lines 13 to 16 in Figure 1 (a)), and the relationship between Curriculum and Author comes from Rule RH application (line 31). Inheritance relationship generation is also considered by these rules if it is possible to infer that the child
element specializes the complex element (Rule HC), or one of the elements involved in the reference is a more generic term than the other (Rule RH). Inheritance is possible only if the child element occurs once in the complex element content model (IS-A notion).

Rule HR is a semi-automatic rule that requires user intervention to determine how many semantic intentions exist between a child element and a complex element. It happens when a child element is repeated in the content model of the complex element. For example, the element Person has two child element Address (lines 7 and 8). Supposing that they have different meanings (home address and office address), two different rolled association relationships are defined in the conceptual schema of Figure 1 (b).

Definition 6: Rule DE (Derivation Conversion). A $d \in D$ generates an $i \in I$, with $i = <g.n, s.n>$.

An example of Rule DE application is the inheritance relationship defined from Person to Author in Figure 1 (b), based on the extension definition on Author element (line 29).

Definition 7: Rule AE (Anytype Element Conversion). A $cc \in E_c$, with $cc.st = \text{"anytype"}$, or a $cc \in E_A$, generates:

\[
\begin{align*}
&l \in L, \text{with } l = <cc.name, \text{string}, \text{null}> , \\
&\text{if } cc \in E_A; \\
&l \in L, \text{with } l = <cc.n+\text{"Text"}, \text{string}, \text{null}> , \\
&(ii) a \in A, \text{with } a = <cc.n, l.n, \text{card}_d, \text{card}_i, \text{null}> , \\
&\text{otherwise}.
\end{align*}
\]

Rule AE treats an anytype element content as an unstructured text that may hold any value. Thus, this kind of element always generates a lexical concept $L_a$ to maintain such text data, even if the element has other attributes that force it to become a non-lexical concept $NL_a$ (Rule CE). If $NL_a$ was previously generated, an association relationship is defined to connect $L_a$ to it. The conversion of the Body element (line 47) is an example.

In a similar way, Rule ME, in the following, generates a lexical concept to hold all textual occurrences in the body of a mixed element. The conversion of the Curriculum element (line 20) is an example.

Definition 8: Rule ME (Mixed Element Conversion). A $cc \in E_c$, with $cc.st = \text{"mixed"}$, generates: (i) $l \in L$, with $l = <cc.n+\text{"Text"}, \text{string}, \text{null}>$; (ii) $a \in A$, with $a = <cc.n, l.n, \text{card}_d, \text{card}_i, \text{null}>$.

Definition 9: Rule E (Empty Element Conversion). Let $e_s \in T \cup E_o$, $o \in O$ and $o.s = e_s$. A set $\{h_1, ..., h_n\} \in H$, with $\{h_1, ..., h_n\} \subseteq o.sh$, where $\forall h_i \in \{h_1, ..., h_n\} \Rightarrow h_i.t \in E_0$, generates: (i) $l \in L$, with $l = <e_s.n+\text{"Type"}, \text{string}, \{h_1.t.n, ..., h_n.t.n\}>$; (ii) $a \in A$, with $a = <e_s.n, l.n, \text{card}_d, \text{card}_i, \text{null}>$.

Empty elements without attributes may be considered a qualification (a "flag" or a
status") of a complex element $e_s$, if they are defined on its content model. Based on such assumption, one or more of these elements are abstracted into a lexical concept by Rule E. Such concept is associated to the concept that corresponds to $e_s$ in order to maintain its qualifications. An application of this rule generates the concept BookType in Figure 1 (b) from the elements Arts, Science and Fiction (lines 43 to 45).

Definition 10: Rule U (Union Conversion). An $sc \in A \cup E_s$ with $sc.u = \{type_1, ..., type_n\}$ generates: (i) $l_1, ..., l_n \in L$, with $l_i = \langle sc.n + \text{"Union"} + i, type_i, \text{null}\rangle$; (ii) $a_1, ..., a_n \in A$, with $a_i = \langle sc.n, l_i.n, \text{card}_d, \text{card}_i, \text{null}\rangle$; (iii) $m \in ME$, with $m.D = \{a_1, ..., a_n\}$.

Definition 11: Rule C (Choice Conversion). Let $o \in O$ and $o.t = \text{"choice"}$. generates $m \in ME$, with $m.D = \{\{r_1, ..., r_n\}\}$, being $r_1.c_1 = o.s.n$ and $r_i.c_2 = h_i.t.n$, with $h_i \in o.sh$.

Rule C defines mutual exclusive relationships at a conceptual level. It is applied to the choice construct defined to the element Affiliation (lines 32 to 37).

4.2. Rules Execution and Mapping Information

The Conversion phase executes the previously defined rules through a conversion algorithm. It basically determines which rule must be applied, depending on the construct that is found in the XML Schema document. It processes on two steps in order to organize the rules execution, considering that some rules depends on the results of other rules:

- **step 1**: generates concepts from the basic XML Schema constructs: types, elements and attributes. It executes rules TE or SC, depending on the discovered construct;
- **step 2**: generates relationships between the concepts created in the previous step, as well as concepts and relationships related to the special kinds of elements (any-type, mixed and empty) or unions. It executes rules HC, HR, RH or DE for defining relationships; rules AE, ME or E, if one of these special kinds of elements is found; and rule U, if an union is defined. Besides, rule C is executed at the end of a complex element analysis, if it defines a choice on its content model.

The conversion algorithm also keeps track of the hierarchical path of found elements in the XML schema document. Such information is needed to define the correspondences between XML Schema and ORM/NIAM schemata. We adopt XPath 1.0 expressions to specify these mappings.

The mapping of a concept is defined as an absolute XPath expression. For example, the mapping of the concepts Job and ISBN in Figure 1 (b) is "/Curriculum/Job" and "/Book/@ISBN", respectively, considering that Author be the root element of the XML documents (denoting by "/").

The mapping of an association relationship is basically defined as a relative XPath expression. Such expression denotes how to navigate between related data in XML documents. Mappings are defined for both relationship directions to allow the translation of any traversal over the conceptual schema graph. In Figure 1 (b), the expressions "Book" and ". . ." denote, respectively, the mapping of the relationship between Author and
Book in the directions Author→Book and Book→Author. Inheritance relationships has no mapping because types do not represent concrete data in XML documents.

5. Conversion Catalog

Information about the output conceptual schema and mappings are kept in a catalog that is stored in an OWL (Ontology Web Language) document [4]. OWL is suitable for ontology and conceptual schema specification and follows an object-oriented paradigm: classes describe conceptual entities, and properties describe attributes and relationships between classes, with optional constraints. Table 1 summarizes the adopted correspondences between ORM/NIAM and OWL constructs for cataloging purposes.

<table>
<thead>
<tr>
<th>ORM/NIAM</th>
<th>OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept</td>
<td>class</td>
</tr>
<tr>
<td>relationship</td>
<td>class</td>
</tr>
<tr>
<td>data type</td>
<td>property constraint</td>
</tr>
<tr>
<td>enumeration</td>
<td>specialized concept class with allowedValues property</td>
</tr>
<tr>
<td>association relationship with role</td>
<td>specialized relationship class with additional role property</td>
</tr>
<tr>
<td>mutual exclusive relationships</td>
<td>OWL predefined disjointWith property</td>
</tr>
<tr>
<td>mapping information</td>
<td>class</td>
</tr>
</tbody>
</table>

In fact, the OWL catalog schema is organized in two levels: metadata and data. Metadata level is fixed and defines classes for all ORM/NIAM constructs, like non-lexical concept and association relationship. Data level is variable and depends on the considered domain. Every OWL class at the data level is a subclass of an OWL metadata class. Figure 3 shows some OWL cataloging at the data level for the ORM/NIAM schema of Figure 1 (b).

Figure 3(a) specifies the non-lexical concept Author. The property RelatedConcepts restricts its relationships to a collection of relationship instances. One of these instances is presented in Figure 3(c). It defines an association relationship between the concepts Author and Book. Four properties (SourceConcept, TargetConcept, DirectCardinality and InverseCardinality) hold the related concepts as well as the direct and inverse relationship cardinalities, respectively. An inheritance relationship definition is similar, but without cardinality properties.

A lexical concept specification is given to the concept City in Figure 3(b). The property DataType is restricted to string.

Each concept and relationship class has mappings to the XML sources which it is defined. Such information is cataloging in the ConceptMapping and
RelationshipMapping classes, respectively. Figure 3(d) shows a mapping to the Author concept. It defines the XML source URL (Source property) as well as the mapping expression to their corresponding element (PathExpression property).

6. Related Work

A schema for XML organizes data in an hierarchical structure, like a hierarchical database schema. Thus, a natural solution should be to consider existing hierarchical database reverse engineering approaches [15, 18, 20] to the context of XML schemata. However, such approaches are limited because the hierarchical model is less rich than the XML model. The hierarchical model defines data types, as well as hierarchical relationships and references between them. The XML model provides additionally other concepts regarding to the semistructured nature of the XML data: mixed, empty and anytype elements, complex elements with alternative representations (choice construct) and element derivation. Besides, the approaches for hierarchical database reverse engineering do not consider data semantics during the conversion process.

These limitations have motivated specific work related to the reverse engineering of XML data, that, in turn, have some drawbacks. Some of them, for example, adopt a
manual approach, acting basically as a tool that aids the user to define conceptual views of XML data [10, 12, 17, 23]. As a consequence, a weak automation level is provided.

Other point is related to the input and output model involved in the conversion. Some approaches consider DTD as the input model, which provides limited resources, if compared to XML Schema [10, 17, 19]. Others work on graph-based logical abstractions of XML data, defining basically mappings between nodes and arcs to entities and relationships of a conceptual schema [11, 12]. Such work do not take into consideration the particularities of the XML Schema model. The remaining ones consider specifically the conceptual abstraction of schemata defined in XML Schema but, in some cases, special element types (empty, for example) are not completely treated [9, 16, 22]; or choices and/or inheritance inference are not properly considered [21, 23]. Our approach differs from these work in terms of generating inheritance relationships for composite-component associations and qualifications for empty elements.

With respect to the output conceptual model, there is a choice among ER-like [9, 12, 16], object- or UML-like [10, 19, 21, 23] or specific graph-based semantic models [11, 17]. However, no one of these models is completely suitable to abstract XML schemata. UML and ER-like models, as pointed out before, are not able to represent simple elements or attributes that are shared among several complex elements. Graph-based models usually does not represent disjoint association relationships or inheritance relationships. Only one related work also adopts ORM/NIAM [22], but its set of conversion rules is less comprehensive then our approach.

Other limitations are related to the considered activities performed by the reverse engineering process. Some work does not execute preprocessing tasks in order to simplify the input XML schema [11, 17, 23]. As a consequence, the conversion algorithm becomes complex because must deal with complex structures and sometimes semantically irrelevant data. Despite of this, some work also do not consider the generation of inheritance relationships [9, 10, 12, 17, 21, 22].

7. Conclusion

Reverse engineering of XML schemata is getting focus due to the widespread use of XML protocols. In the area of XML databases there are several open issues related to this task, including database design and re-design methodologies [5]. Further, XML reverse engineering has been considered in XML schema integration approaches that define a global conceptual schema from a set of heterogeneous XML documents or XML schemata. As a structural basis for knowledge representation, this conceptual schema may also be extended to define a domain ontology. This paper presents a contribution for these issues.

We propose a rule-based process for generating a conceptual schema from a schema in XML Schema. The main contribution of this process is a detailed analysis of the XML Schema constructs, as well as related XML documents. The output conceptual schema models elements’ types (simple, composite, mixed, etc); attributes; data re-
relationships (type-to-type, type-to-element, element-to-element, and element-to-attribute); inferred inheritance relationships on types and elements; and alternative representations for elements (choice construct). No related work provides such coverage.

Besides, we define XPath mapping expressions for the conceptual constructs obtained from XML Schema constructs. Supposing that the reverse engineering is considered for XML schema integration, a query defined over a global conceptual schema can be easily translated to XPath queries to be executed at the XML local sources. No similar detailed mapping strategy using XPath is found in related work.

As user expertise is considered in the process, a good conversion quality is expected. Despite of that, future work include the use of machine learning techniques with the purpose of reducing user intervention as much as possible. Besides, the consideration of semantic integrity constraints of local XML sources is also important. Such information could be discovered and associated to the conceptual schema to improve semantics.

The reverse engineering process is under implementation. It will be part of a larger process called BInXS [14], which aims at building a global conceptual schema from the semantic integration of XML data sources on the Web. The intention is to provide a conceptual abstraction that hides the heterogeneity of the XML sources schemata in order to reduce the complexity of further queries over these sources.

References


