Xere: Towards a Natural Interoperability between XML and ER Diagrams*

Giuseppe Della Penna, Antinisca Di Marco, Benedetto Intrigila, Igor Melatti, and Alfonso Pierantonio

Università degli Studi di L'Aquila, Dipartimento di Informatica
{dellapenna,adimarco,intrigila,melatti,alfonso}@di.univaq.it

Abstract. XML (eXtensible Markup Language) is becoming the standard format for documents on Internet and is widely used to exchange data. Often, the relevant information contained in XML documents needs to be also stored in legacy databases (DB) in order to integrate the new data with the pre–existing ones. In this paper, we introduce a technique for the automatic XML–DB integration, which we call Xere. In particular we present, as the first step of Xere, the mapping algorithm which allows the translation of XML Schemas into Entity-Relationship diagrams.

1 Introduction

Over the last years, Internet related technologies had an exponential growth and motivated a stronger demand for standards in information interchange and treatment. As a consequence, using XML (eXtensible Markup Language) [8] as a standard format for data and documents on Internet is becoming a commonplace.

Moreover, we are facing the shift of XML usage from its original *presentation-centric* nature to a more *information-centric* one. A number of organizations and individuals are using XML to exchange data that need to be stored and managed in some way. Information is often originated directly in XML, and since XML data are contained in files, it is usual to store them "as they are" on a filesystem. Nevertheless, the relevant information contained in such files often needs to be also stored in (legacy) databases. Indeed, applications that manage and manipulate this information usually work on (relational) databases, and companies want to continue to use these "legacy" applications, since adapting them to read directly data from XML files may be too expensive and complex. This results in an almost unavoidable duplication of data, which is often troublesome: the association between the XML data and its image in the database is possibly not preserved, and this may lead to consistency errors.

To face these problems, we propose a general technique, which we call Xere (XML Entity Relationship Exchange), to assist the XML – DB integration, in a way that alleviates the consistency problems and helps in merging the XML data with pre–existing legacy ones. In particular, the database structures that contain the XML data should be naturally related to the XML document structure, so

* Research partially supported by the MURST project SAHARA

M. Pezzè (Ed.): FASE 2003, LNCS 2621, pp. 356–371, 2003.

that it could be possible to transparently store XML data in databases, query the resulting data structures, and automatically regenerate "on the fly" the source documents from the DB content.

In this paper, we present the first step of Xere, consisting of an algorithm that maps XML Schemas to Entity–Relationship models.

Mapping XML Schemas to the Entity–Relationship conceptual model, rather than to the relational structures of a database, has various advantages:

- The ER diagram offers a good documentation to the DB designer and maintainer.
- A *natural* mapping to the relational model is not always possible, since the relational model and XML Schemas are on two very different abstraction layers. In particular, there are many constraints and optimizations that can be only "seen" on the ER model, due to the part of structural information which is lost in the "flat" relational model. Indeed, in the traditional database design process, the relational model is used only in one of the last steps, very close to the physical deployment.
- The integration with pre–existing data structures in a legacy database can be more easily studied on a conceptual model.

We decided to adopt XML Schemas [9] rather than DTDs for a number of reasons. Although, previous approaches (such as [11]) to the XML–DB integration use DTDs to create a general mapping algorithm that is subsequently applied to the XML documents, DTDs are being progressively replaced by XML Schemas.

Moreover, using XML Schemas is convenient since this formalism offers a better expressiveness to describe very complex document structures. It can express advanced concepts like generalization, type derivation and substitution, complex type definitions and a variety of content models (such as sequence, choice, set). XML Schemas are also strongly typed – as databases are: actually, the XML Schema basic types are directly derived from SQL data types (see [9]), and this helps also in the physical deployment of the database structures.

Finally, since a XML Schema is itself an XML-based formalism, it can be processed by a variety of XML–based tools. Indeed, in this paper we also present the prototypical implementation of our Schema to ER mapping using only XML-related technologies and (free) tools. Note that many free tools are also available to translate DTDs in XML Schemas [9], so our approach could indirectly handle also DTDs.

The paper is arranged as follows. Section 2 presents the Xere mapping outlining how XML Schemas are translated into Entity-Relationship diagrams. In Sect. 3 we prove the completeness and soundness of the Xere mapping algorithm. Finally, Sect. 5 compares the work which is presented in this paper with related works and Sect. 6 draws some conclusions and discusses future works.

2 Xere: Mapping XML Schemas into ER Diagrams

This section illustrates the Xere mapping. Since XML Schemas are also XML documents, a running implementation of Xere is given by means of XSLT [10],

as described in Sect. 2.2. In Sect. 2.3 we also sketch a possible optimization of the mapping.

2.1 Mapping Algorithm

The mapping algorithm takes an XML Schema as input and creates the corresponding Entity-Relationship (ER in the following) model as output.

The mapping algorithm illustrated in this section does not support all the features of XML Schemas, because of space limitations, and we focus on a meaningful subset of Schema elements.

Table 1 shows all the XML Schema elements, divided in categories, and some of their attributes. For a complete reference on the XML Schema grammar and its semantics, the reader can refer to [9]. The "Supported" columns in Table 1 show which elements are supported in the current implementation of the Xere mapping. The omitted elements are not fundamental for the schema definitions. Therefore, our current algorithm can be considered effective on most of the XML Schemas. In the following, we describe the features recognized by the mapping algorithm and the resulting ER structures.

Table 1. Schema elements and attributes currently supported by the Xere mapping

Category	Element	Supported
Particles	all	Yes
	element	Yes
	choice	Yes
	sequence	Yes
	group	Yes
	any	No
Attributes	anyAttribute	No
	attribute	Yes
	attributeGroup	Yes
Complex types	complexType	Yes
	complexContent	Yes
	simpleContent	Yes
	restriction	Yes
	extension	Yes
Simple types	simpleType	No
	list	
	union	
Identity constraints	unique	No
	key	Yes
	keyref	Yes
	field	Yes
	selector	Yes
Other elements	schema	Yes
	notation	No
Facets	maxExclusive	No
	maxInclusive	
	minExclusive	
	minInclusive	
	length	
	maxLength	
	minLength	
	fractionDigits	
	totalDigits	
	pattern	
	enumeration	
	whiteSpace	
Comments	annotation	No
	appinfo	
	documentation	
Schema combination	import	No
	include	
	redefine	
Schema attributes	minOccurs	Yes
	maxOccurs	Yes
	mixed	No
	substitutionGroup	No
	default	No
	fixed	No
	nillable	No

```
<element name="A">
 <complexType>
  <sequence>
   <element name="B" type="integer"
      minOccurs="0"
      maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

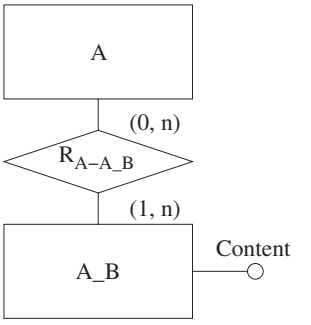

Fig. 1. Mapping of a local element.

```
<element name="A">
 <complexType>
  <sequence>
   <element ref="B"
      minOccurs="b1"
      maxOccurs="b2"/>
   <element ref="C"
      minOccurs="b3"
      maxOccurs="b4"/>
  </sequence>
 </complexType>
</element>
```

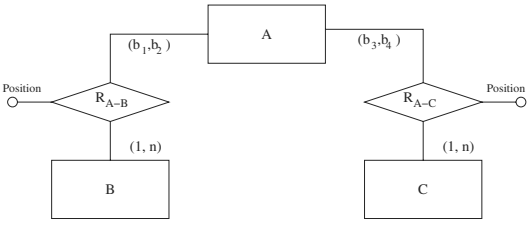

Fig. 2. Mapping of a sequence model.

```
<element name="A">
 <complexType>
  <choice minOccurs="b1"
          maxOccurs="b2">
   <element ref="B"
      minOccurs="b3"
      maxOccurs="b4"/>
   <element ref="C"
      minOccurs="b5"
      maxOccurs="b6"/>
  </choice>
 </complexType>
</element>
```

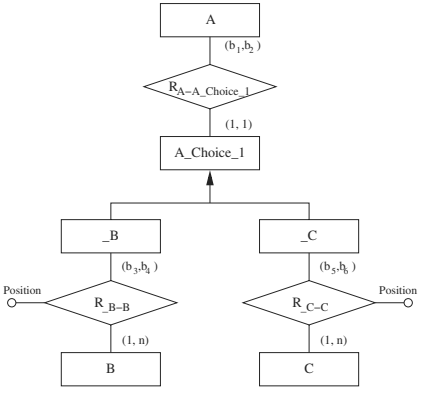

Fig. 3. Mapping of a choice model.

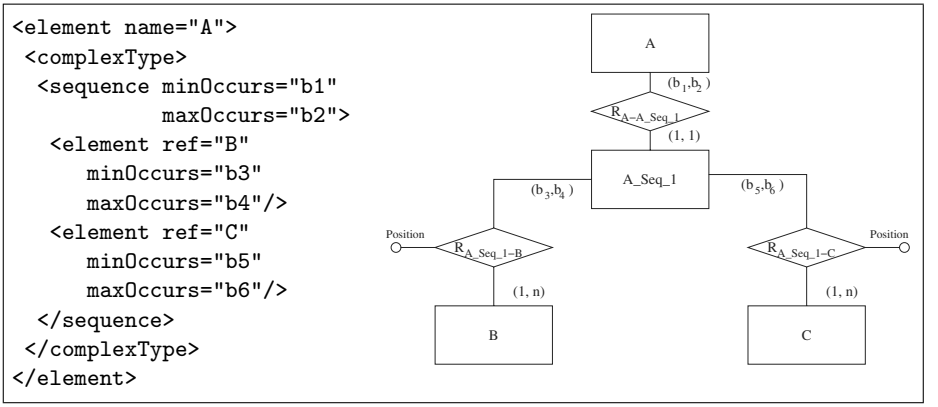

Fig. 4. Mapping of a sequence model with occurrence constraints using an auxiliary entity.

Elements. Each element becomes an entity. The entity name is obtained from the element name considering its nesting (see "Element nesting" below). The element primary key is chosen using the following rules:

1. if the schema explicitly defines a key (using the `xs:key` construct[1]) for the element, that key is used as its primary key. Note that the keys defined using the `xs:key` construct can be composite and make references to attributes belonging to other elements (which are parents of the current one). In this case, we are simply declaring the resulting entity as a weak entity, so its primary key depends on some attributes of other entities that are associated with it via a parent relationship. Our algorithm guarantees that the relation between any element and its parent exists in the resulting ER model.
2. if the element has an attribute of type `xs:ID`, that attribute becomes its primary key.
3. otherwise, the entity has not any key. Entities with no keys are secondary entities that will never be addressed directly.

Simple elements (e.g. without attributes and child elements) have a special "content" attribute, which holds the textual content of the element. Simple elements may be further optimized (see Sect. 2.3).

Attributes. Element attributes are mapped to entity attributes, regardless of their `xs:use` attribute value (e.g. required, implied, etc.).

Element Nesting. Elements in XML Schemas can have a scope, so e.g. we can define elements with the same name in different types. The mapping takes care of this situation by using local names when creating entities derived from nested definitions. The local name is simply obtained by appending the element name to the container's name: for example, an element "B" declared inside another element "A" will result in the creation of an entity named "A_B" (see Fig. 1 for an example).

[1] In this paper, the `xs` prefix always denotes the XML Schema namespace.

Global Declarations and References. Elements, attributes and types can be globally declared and then referenced by name. The algorithm handles both situations.

- global types are inline expanded where referenced.
- global elements are created once (as entities) and referenced when needed.
- global attributes are copied in any entity that references them.

Complex Types. All kind of complex types are recognized. Types derived by extension from other complex or simple types are expanded and the final type is obtained merging all the types in the derivation hierarchy. Types derived by restriction are handled as natural by rewriting the entire complex type with appropriate restrictions.

Content Models. All content models (i.e. `sequence`, `choice`, `all`) are recognized. The mapping is designed to be as natural as possible. Content children can be either elements or other content models. In general, all the content children are created applying recursively the mapping algorithm and then attached to the content owner using relations. Note that, in our ER models, relations are oriented: this means that we can always say that a particular relation *exits* from an entity (the "parent") and *is directed to* another entity (the "child"). Moreover, since these relations represent a parent–child relationship in the XML Schema, their cardinalities are $(1 \text{‹} 1)$ on the parent side, unless different occurrence constraints are explicitly defined, and $(1 \text{‹} n)$ on the child side, since children with the same content from different XML instance documents may be merged and attached to many parents for data optimization purposes. The particular mapping for each content model is described in the following paragraphs.

`Sequence` **Models.** All the content children are linked to the model owner using relations. The sequence relative positions of child elements are expressed in the ER diagram by adding a `position` attribute to these relations (see Fig. 2 for an example). Note that this "extra" attribute actually expresses an implicit attribute of the XML Schema: indeed, the choice of a sequence content model implies an ordering on the information represented by the model children.

`Choice` **Models.** The choice model is translated using the generalization construct of ER diagrams. An auxiliary entity is attached to the model owner with an appropriate relation, and is specialized in as many other auxiliary entities as the content children are. The reason for adding the auxiliary entities is that we need the corresponding relations to store the cardinality constraints. Finally, the actual content model children are linked to the corresponding auxiliary entity using a relation. If a child has occurrence constraints, we add to the corresponding relation a `position` attribute to keep track of the occurrences ordering (see Fig. 3 for an example). Note that the ER generalization construct may not be always semantically consistent with the choice XML Schema content model. However, the generalization, used in the particular way explained

above, is the only construct that allows to maintain at least the "mutual exclusion" semantics of the choice model in the ER diagram.

`All` **Models.** The all model is realized similarly to the sequence model, without keeping track of the children position in the sequence.

Elements `xs:group` are used to import frequently used content models. The algorithm expands them inline anywhere they are referenced. The expanded content models are recursively processed using the previously described rules.

Occurrence Constraints. Occurrence constraints in content model children are mapped to relation cardinalities in the ER model. As already said, a special attribute `position` is added to the relations to keep track of the occurrences ordinal position in choice and sequence content models. If the occurrence constraint is placed on a sequence model, an auxiliary entity and a relation are created to store the corresponding cardinality (see Fig. 4). Note that the role of auxiliary entities is only to provide a more natural way to map the Schema to the ER diagram. Most of the times, these temporary entities are discarded when reducing the ER schema to a relational schema.

Key De nitions and Key References. Key definitions are mapped as entity primary keys. Key references are translated by executing the two following steps:

1. the attribute corresponding to the `xs:field` part of the key reference is removed from its entity.
2. a relation, with the same name of the removed attribute, is created from the referring entity to all the entities containing the attributes that compose the key. The cardinalities of this relation are $(1 \text{‹} 1)$ on the side of the entity with the removed attribute and $(1 \text{‹} n)$ on the other sides.

Finally, the ER model is completed by creating an additional entity, conventionally called *document*, that has a single attribute called *name*. This entity is associated through a relation to the entity representing the XML document root. In this way, we can distinguish the elements of each different document stored in our data base.

2.2 An XSLT Implementation of the Mapping

In this section we briefly introduce a prototype implementation of most of the Xere mapping algorithm described in Sect. 2, with the exception of primary keys, key references, attribute and element groups.

We implemented the algorithm using a XSLT [10] transformation stylesheet that can be used and tested on possibly any platform using an XSLT interpreter.

The output of the transformation is another XML document describing the resulting ER model with a very simple markup containing the following elements:

- `<entity name="foo">` declares an entity of the ER diagram called `foo`.
- `<attribute name="foobar" type="atype">` inside an `<entity>` or `<relation>` element, declares an attribute of that entity or relation with the specified name and type.

- <relation name="foobar" from="foo" to="bar" card="n_1,m_1-n_2,m_2"> declares a relation, called foobar, between the two named entities foo and bar, with the given cardinality (n_1,m_1 and n_2,m_2 are the cardinalities of the foo and bar entities in the relation, respectively).
- <specialization base="foo"> defines a generalization having foo as its base (most general) entity. All the children of the <specialization> element are possible specializations of the base.

The complete implementation of the mapping algorithm outlined here, together with the XML Schema of the language used to represent the ER models, can be found in [3]. Moreover, it is possible to try the algorithm online at the url http://dellapenna.univaq.it/xere/.

2.3 Optimizations on the Algorithm

Many optimizations can be applied to the mapping algorithm described in Sect. 2.1. We decided to remove these optimizations from the first release of the algorithm, since they may make the resulting ER diagram more complex and difficult to understand.

We may suggest two major optimization to the current Xere mapping algorithm:

- Optimizations based on cardinalities and simple types may decrease the number of entities created by the algorithm. For example, an element with a simple type may be safely translated in an attribute (possibly with cardinality) if it is contained in another entity and has "reasonably low" occurrence constraint. However, this optimization should be carefully validated by domain experts not to alter the diagram semantics.
- Element groups are expanded inline in the current mapping. Therefore, the group content becomes a nested content model, and this may lead to the creation of many different instances of the same set of entities. To optimize this process, we may create a temporary entity called, say, *GroupX* and connect the group content model to it. Then, any reference to that group could be translated into a simple relation to the *GroupX* entity, thus saving much space. This transformation is semantically consistent since elements are grouped when they have a fixed semantics that is shared among several different content models.

3 Soundness and Completeness of the Xere Mapping

The Xere mapping algorithm described in Sect. 2 covers the main XML Schema elements, as shown in Table 1. There are some secondary elements and constructs we decided to not include in the first version of the Xere algorithm, since they are not fundamental for the schema definitions. Therefore, our current algorithm can be considered effective on most of the current XML schemas.

In this section we sketch the completeness and soundness proofs for the current Xere algorithm.

Proposition 1 (Xere Mapping Completeness). *The Xere mapping algorithm is complete w.r.t. the given subset of the W3C XML Schema Specification [9].*

Proof. By analyzing the meta–schema given in [9], we can see that the meta–schema is built recursively from the base elements described in Table 1. The Xere algorithm is well-defined on all the base elements shown in Table 1. These base elements are translated in ER entities. The algorithm is in turn recursive, so it can go down through the schema definition, translating the parent–child relations in ER relations. □

To prove the soundness of the Xere mapping we show that, given any XML document valid w.r.t. a particular schema, we can store the document content as an instance of the ER model generated by the Xere algorithm and then rebuild that document (or an equivalent instance) from that ER model instance.

In other words, we can represent XML documents as instances of the ER model created by the Xere mapping for the corresponding schema, so that the *document identity* is preserved.

Definition 1 (Document Identity). *We say that two XML documents D_1 and D_2 with the same schema S are* identical *(or that they have the same* identity*) iff they have*

- *The root elements of D_1 and D_2, namely R_1 and R_2, are the same (i.e. they have the same tag).*
- *The two root elements R_1 and R_2 have the same attributes with the same values. Attribute order is not considered. Attribute values should be expanded w.r.t. their defaults, as described in S.*
- *If R_1 and R_2 have a simple content, then the contents are identical.*
- *If R_1 and R_2 have a complex content, then there exists a bijective mapping between the children of R_1 and R_2, so that the document fragments corresponding to related children are in turn identical. In addition, if R_1 and R_2 have a sequence model (declared in the schema), then the bijective mapping must also preserve the child positions: the first child of R_1 is mapped into the first child of R_2, and so on.*

Proposition 2 (Xere Mapping Soundness). *The Xere mapping algorithm preserves the XML documents identity.*

Proof. Since the Xere algorithm is recursive, the soundness proof will be given by induction on the XML documents structure.

The simplest XML document is composed by a single element, possibly with attributes and a textual content. In an XML Schema, such element would be declared with a schema fragment like that shown in Fig. 5a.

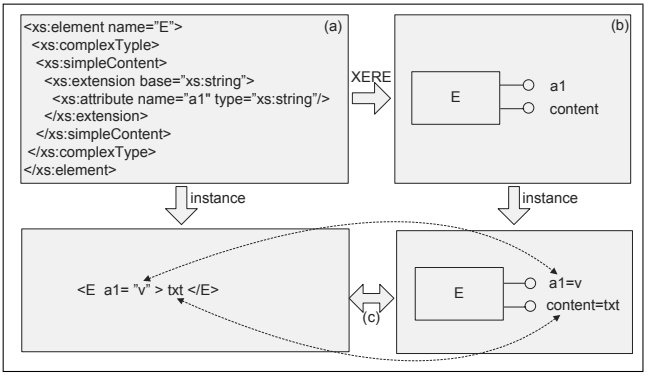

Fig. 5. Mapping of a simple element.

The Xere algorithm would translate this schema fragment creating an entity called E with two attributes, *content* and $a1$, which correspond to the element textual content and to the element attribute $a1$, respectively (see Fig. 5b).

An instance of the given schema is `<E a1=□□v□□>txt</E>`. This instance is mapped on the ER model as an instance of the entity E, where the entity attributes *content* and $a1$ are assigned to the values `txt` and `v`, respectively (see Fig. 5c).

From the other end, given an instance of the ER model described above, we map it to an XML document instance by creating an element with the same name of the entity, i.e. E, and assign it the textual content given by the *content* attribute, if it exists. Then we add an attribute to the element for each other attribute of E, and assign it with the corresponding value (see Fig. 5c).

For the ER model instance generated above, we would rebuild exactly the source XML `<E a1=□□v□□>txt</E>`.

Now we describe how content models are mapped. We inductively suppose that we can store and retrieve any XML subtree, and show how the content models (all, sequence, choice) are mapped.

A basic sequence model is declared by the schema fragment shown in Fig. 6a, where `M1`, `M2`, ..., `Mn` can be element declarations (or references) or nested content models.

The Xere algorithm would translate this schema fragment creating an entity called E, recursively building the ER sub–diagrams corresponding to `M1`, `M2`, ..., `Mn`, and connecting these sub–diagrams to E with n relations. The created relations have a *position* attribute that specifies the sequence position of the corresponding sub–diagram (see Fig. 6b). Note that, by induction hypothesis, we can always build the required sub–diagrams.

An instance of the given schema is `<E><E1>...</E1>...<En>...</En></E>`, where the subtrees `E1`, ..., `En` are valid w.r.t. the respective schema fragments `M1`, ..., `Mn`. This instance is mapped on the ER model instance by

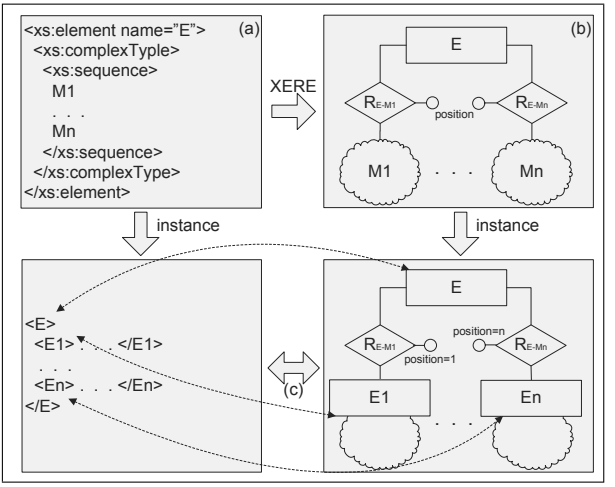

Fig. 6. Mapping of a sequence model.

1. first recursively creating the instances of the ER sub–diagrams corresponding to `E1`, ..., `En`;
2. then, an instance of the ER entity E is created and the relations are set to associate this instance with the sub–diagrams of $E1$, ..., En. Each relation instance has an attribute *position* that is set to the ordinal position of the corresponding sequence child `E1`, ..., `En` (see Fig. 6c).

On the other hand, given an instance of the ER model described above, we map it to an XML document instance by creating an element with the same name of the root entity, i.e. E, and recursively rebuilding the XML subtree corresponding to the n sub–diagrams associated to E with the n relations. Then, we insert these subtrees in the root element E using the order given by the *position* attribute on the relations (see Fig. 6c).

For the model instance generated in the example above, we would rebuild exactly the source XML `<E><E1>...</E1>...<En>...</En></E>`.

A basic choice model is declared by the schema fragment shown in Fig. 7a, where `M1`, `M2`, ..., `Mn` can be element declarations (or references) or nested content models.

The Xere algorithm would translate this schema fragment creating an entity called E and recursively building the ER sub–diagrams corresponding to to `M1`, `M2`, ..., `Mn`. A temporary entity T is created and associated with E using a relation R_{E-T}, and n other temporary entities T_M1, ..., T_Mn are attached as specializations of T. Finally, the roots of the sub–diagrams for `M1`, `M2`, ..., `Mn` are attached to the corresponding temporary entities T_M1, ..., T_Mn using an appropriate relation (see Fig. 7b).

An instance of the given schema is `<E><Ei>...</Ei></E>`, where the subtree `Ei` is valid w.r.t. the respective schema fragment `Mi`. This instance is mapped on

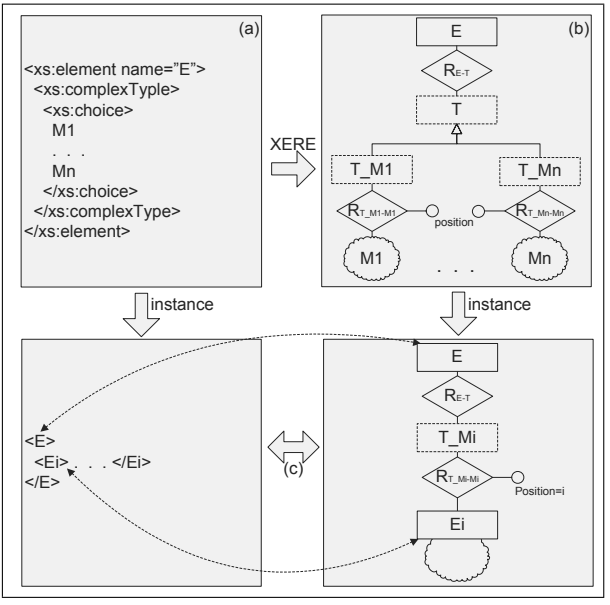

Fig. 7. Mapping of a choice model.

the ER model by first recursively creating the instance of the ER sub–diagram corresponding to `Ei`. Then, an instance of the ER entity E is created and the relation R is set to associate this instance with the sub–diagram of Ei, implicitly choosing the corresponding specialization of T (see Fig. 7c).

From the other end, given an instance of the ER model described above, we map it to an XML document instance by creating an element with the same name of the root entity, i.e. E, and recursively rebuilding the XML subtree corresponding to the specialized sub–diagram associated to E with the relation R. Then, we insert this subtree in the root element E (see Fig. 7c).

For the model instance generated in the example above, we would rebuild exactly the source XML `<E><Ei>...</Ei></E>`.

The all content model is mapped exactly as the sequence model, with the exception of the *position* attribute that is not created and ignored during the document storing and retrieval. Therefore, the all model mapping is even simpler than the sequence model mapping. □

Although we only give here a sketch of the inductive soundness proof, we think it should be enough to ensure the soundness of the Xere approach.

4 An Example

In this section we briefly show an application of our mapping algorithm and the obtained results. The Schema we used, shown in Figure 8, defines the structure

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
 <!--Global Types: we skip the part where the global types "full_name_type",
     "name_type","authors_type","keywords_type","relatedwords_type" and
     where the global elements "family","middle" and "first" were defined-->
 <!--Root Element-->
 <xs:element name="issue">
  <xs:complexType><xs:sequence>
    <xs:element name="editor" type="name_type"/>
    <xs:element name="articles"><xs:complexType><xs:sequence>
      <xs:element ref="article" maxOccurs="unbounded"/>
     </xs:sequence></xs:complexType></xs:element>
  </xs:sequence></xs:complexType>
 </xs:element>
 <xs:element name="article"><xs:complexType><xs:all>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="authors" type="authors_type"/>
  <xs:element name="summary">
   <xs:complexType><xs:choice>
     <xs:element ref="keywords"/>
     <xs:element ref="related_words"/>
   </xs:choice></xs:complexType></xs:element></xs:all>
  <xs:attribute name="category" type="xs:string"/>
  </xs:complexType></xs:element>
 <xs:element name="author" type="full_name_type"/>
 <xs:element name="keywords"><xs:complexType><xs:complexContent>
    <xs:extension base="keywords_type"/>
  </xs:complexContent></xs:complexType></xs:element>
 <!--We skip the definition of "related_words", that is similar to the
     "keywords" definition above-->
</xs:schema>
```

Fig. 8. XML Schema code for the journal issue example.

of a journal issue. We run on this Schema the stylesheet mentioned in Section 2.2 and the ER model resulting is shown in Figure 9.

Let us highlight the two most interesting parts of this Schema:

- the `articles` definition (exemplifying the `sequence` model);
- the `summary` definition (exemplifying the `choice` model)

that determined the generation of the shaded areas in Figure 9. Note that, since the `articles` element is nested in the `issue` element definition, the corresponding entity is labelled `issue_articles` in the generated ER diagram. For the same reason, the `summary` element nested inside the `article` element generates an entity labelled `article_summary`.

5 Related Works

Most of the related works are dealing with some kind of translation from XML to database systems. As stated in Sect. 2, current approaches make use of DTDs and directly map them on relational schemas. Therefore, we were not able to found other techniques directly comparable with ours. In this section we list general

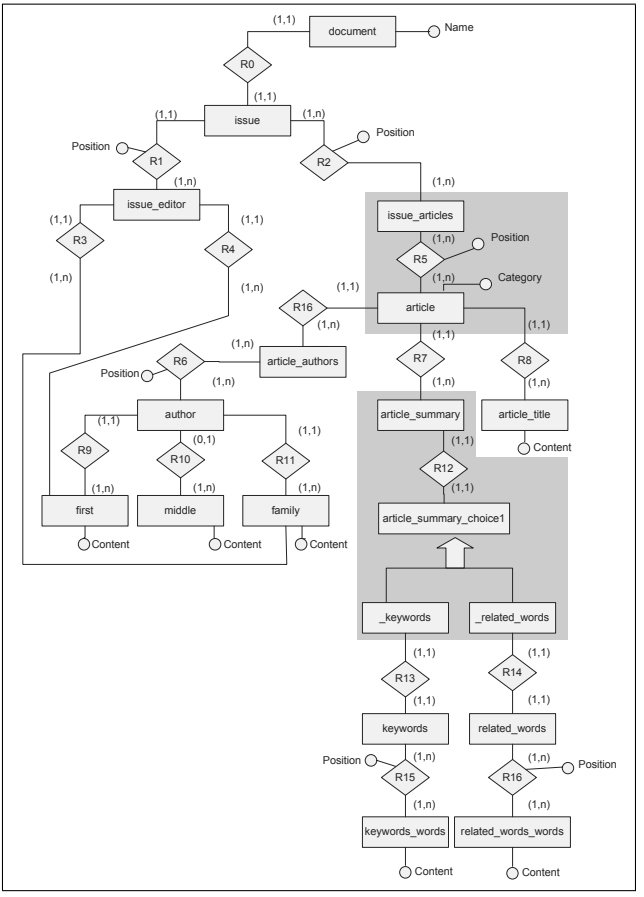

Fig. 9. The ER Model diagram for the journal issue example.

related work about XML–DBMS mapping. Note that many papers in this field still address SGML as the standard formalism for the definition of structured documents, so they actually talk about SGML–DBMS mappings. However, since XML is a subset of SGML, the SGML–DBMS mapping techniques explained in these papers also apply to XML.

There are two main approaches to designing relational database schemas for XML documents. The first approach, namely the *structure-mapping approach*, create relational schemas based on the structure of XML documents (deduced from their DTD, if available). Basically, with this approach a relation is created for each element type in the XML documents, [2,1], and a database schema is defined for each XML document structure or DTD. This is the approach we used in the Xere algorithm. Using ER as the target model, we can optimize the transformation and merge the obtained schemas with legacy DBMS structures.

More sophisticated mapping methods have also been proposed, where database schemas are designed based on detailed analysis of DTDs, [7].

In the *model-mapping approach,* a fixed database schema is used to store the structure of all XML documents. Basically, this kind of relational schema stores element contents and structural rules of XML documents as separate relations. Early proposals of this approach include, [12], or the "edge approach", [4], in which edges in XML document trees are stored as relational tuples. A more recent research using this approach is [11], that also defines an efficient method to query this kind of structure.

In both the approaches above, XML documents are decomposed into fragments and distributed in different relations. Obviously, these decomposition approaches have drawbacks – it takes time to restore the entire or a large subportion of the original XML documents. A simple alternative approach, supported in almost all the XML-enabled RDBMS (e.g. Oracle, SQL Server, etc.), is to store the entire text of XML documents in a single database attribute as a CLOB (Character Large Object). On the other hand, this approach does not allow queries on the document structure using SQL (since all the document is stored in a single field), and the search for a particular document node always implies loading all the XML text and searching using regular expression- or XPath-based engines.

Moreover, since SGML (Standard Generalized Markup Language), [5], was a predecessor of XML, there were several studies on the management of structured documents even before XML emerged, [6]. These methods can roughly be classified into two categories: a *database schema designed for documents with DTD* information and a *storage of documents without any information about DTDs.* The latter approaches are capable of storing well-formed XML documents that do not have DTDs. For both approaches, queries on XML documents are converted into database queries before processing. First, there are simple methods that basically design relational schemas corresponding to every element declaration in a DTD, [2,1]. Other approaches design relational schemas by analyzing DTDs more precisely. An approach to analyze DTD and automatically convert it into relational schemas is proposed in [7]. In this approach, a DTD is simplified by discarding the information on the order of occurrence among elements.

6 Conclusions and Future Work

In this paper, we presented the first step of the Xere methodology, namely a rule-based process to translate an arbitrary XML Schema to an Entity–Relationship diagram in a natural way. This mapping is shown to be sound and complete w.r.t. document identity and XML Schema definition, respectively.

The mapping is denoted as *natural* since it preserves all hierarchical relations defined by a XML Schema. Essentially, the mapping is able to retain all the structure defined by a XML Schema encoding it in the relations and entities of the generated ER diagram. We intend to further investigate this compatibility property as, we believe, it will lead to a homomorphism definition.

In this work, we introduced only the first step of the Xere technique. Next efforts will be devoted to achieve the complete interoperability between XML and relational databases. This requires a translation of ER schemas into relational models and a procedure which allows one to transparently store and retrieve XML documents in the databases created using the Xere technique.

References

1. Abiteboul, S., Cluet, S., Christophides, V., Milo, T., Moerkotte, G., and Siméon, J. 1997. Querying documents in object databases. *Int. J. Dig. Lib.* 1, 1, 5–19.
2. Christophides, V., Abiteboul, S., Cluet, S., and Scholl, M. 1994. From structured documents to novel query facilities. *SIGMOD Rec. 2*3, 2 (June), 313–324.
3. Della Penna, G., Di Marco, A., Intrigila B., Melatti, I., Pierantonio, A. 2002. Towards the expected interoperability between XML and ER diagrams. Technical Report TRCS/G0102, Department of Computer Science, University of L'Aquila.
4. Florescu,D. and Kossmann, D. 1999. Storing and querying XML data using an RDMBS. *IEEE Data Eng. Tech. Bull. 2*2, 3, 27–34.
5. ISO. 1986. Information processing—Text and office systems—Standard General Markup Language (SGML). ISO-8879.
6. Navarro,G. and Baeza-Yates, R. 1997. Proximal nodes: A model to query document databases by content and structure. *ACM Trans. Inf. Syst. 1*5, 4, 400–435.
7. Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., Dewitt,D.J., and Naughton, J.F. 1999. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of the 25th International Conference on Very Large Data Bases* (VLDB, Edinburgh, Scotland, Sept. 7-10). Morgan Kaufmann, San Mateo, CA, 302–314.
8. World Wide Web Consortium. 1998. eXtensible Markup Language (XML) 1.0. `http://www.w3.org/TR/1998/REC-xml-19980210`
9. World Wide Web Consortium. 2001. XML Schema. `http://www.w3.org/XML/Schema`
10. World Wide Web Consortium. 2001. The eXtensible Stylesheet Language (XSL). `http://www.w3.org/Style/XSL/`
11. Yoshikawa, M. and Amagasa, T. 2001. XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. *ACM Transactions on Internet Technology, 1,* 110-141
12. Zhang, J. 1995. Application of OODB and SGML techniques in text database: an electronic dictionary system. *SIGMOD Rec. 2*4, 1 (Mar.), 3–8.