Interoperability Mapping from XML Schemas to ER Diagrams

Giuseppe Della Penna [a,*] , Antinisca Di Marco [a,c] ,
Benedetto Intrigila [b] , Igor Melatti [a] and Alfonso Pierantonio [a]

[a] *Dipartimento di Informatica, Università di L'Aquila, L'Aquila, Italy*

[b] *Dipartimento di Matematica, Università di Roma "Tor Vergata", Rome, Italy*

[c] *Computer Science Department, University College London, U.K*

Abstract

The eXtensible Markup Language (XML) is a *de facto* standard on the Internet and is now being used to exchange a variety of data structures. This leads to the problem of efficiently store, query and retrieve a great amount of data contained in XML documents. Unfortunately, XML data needs often to coexist with *historical* data. At the present, the best solution for storing XML into pre–existing data structures is to *extract* the information from the XML documents and *adapt* it to the data structures' logical model (e.g., the relational model of a DBMS). In this paper, we introduce a technique called Xere (*XML E*ntity *R*elationship *E*xchange) to assist the integration of XML data with other data sources. To this aim, we present an algorithm that maps XML Schemas into Entity-Relationship diagrams, discuss its soundness and completeness and show its implementation in XSLT.

Key words: Data Models, Schema Evolution and Maintenance, Interoperability and Heterogeneity, Web applications/XML, Entity-Relationship diagrams

1 Introduction

A great deal of information is exchanged every day through the Internet. The exponential growth of the Web increases the amount of documents that are

* Corresponding author. Tel: +390862433130 Fax: +390862433057
Email addresses: `dellapenna@di.univaq.it` (Giuseppe Della Penna), `dimarco@di.univaq.it` (Antinisca Di Marco), `intrigil@mat.uniroma2.it` (Benedetto Intrigila), `melatti@di.univaq.it` (Igor Melatti), `alfonso@di.univaq.it` (Alfonso Pierantonio).

shared among a global community which is on the verge of including almost everyone in the next years.

The eXtensible Markup Language (XML) [1], a flexible tagged text format derived from the Standard Generalized Markup Language (SGML) [2], has been proposed in 1996 by the W3C Consortium [3] as a tool to standardize the format of all the documents used on the Internet and meets the challenges of large–scale electronic publishing.

In the last years, XML has become a *de facto* standard and, as the next step of its evolution, it is being adopted also for data description and manipulation. The hierarchical structure of markup documents is very suitable to represent a wide variety of data, especially objects. XML fragments are hence used to contain data structures and make them more portable and open–format. This trend has been further pushed by the fact that applications can easily interface with XML–structured data streams or packets using XML parsers and querying tools (like XPath [4], XSLT [5]) to manipulate their content. Actually, at the present XML is being used much more as a data–definition formalism than as a document–definition language. XML protocols such as SOAP [6] are widely used to transport data on the internet, and a number of organizations are using XML to exchange platform–independent and open–format data.

This scenario points out the problem of efficiently store, query and retrieve a great amount of data exchanged by means of XML documents. To this aim, native XML DBMS such as Tamino [7] have been developed. Such kind of DBMS is a good solution when *all* the data to be managed is in XML format. However, many companies have their own databases filled with *historical* information, and applications that interface with these databases to manipulate the data. Converting all the historical data to XML and/or adapting the applications to work with this new standard may be too expensive and complex and, sometimes, it can make a reliable application instable due to the modifications that have to be made on the software system. This means that, most of the times, it is more convenient to make XML documents coexisting with data in different formats and possibly store them in existing DBMS. This, however, can lead to problems related to data consistency (the same information could be duplicated among different formats) and to the efficiency in retrieving/updating the data. In particular, since all the leading commercial DBMS such as Oracle [8], IBM DB2 [9] and Microsoft SQLServer [10] use the *relational model* to organize the data, the main issue about storing XML is often how to fit it into relational structures.

Unfortunately, the tree structure of XML documents is not easy to store in a relational schema. Moreover, the structure of XML documents is often formalized using the DTD [1] or XML Schema [11] formalisms that contain operators, such as the choice, which are not available in the relational model. To cope with this problem, we may extend database schemas [12, 13], but these extensions would not be compatible with legacy systems, and therefore not a realistic (i.e. commercially useful) solution.

At the present, the best solution for storing XML into pre–existing data structures should be to *extract* the information from the XML documents and to *adapt* it to the data structures' logical model (e.g., the relational model of a DBMS). Typically, this task is performed in a manual way and may require some reverse engineering. Moreover, this task may be further complicated by the structure of the XML documents containing data, that often is not human–readable (i.e., easily understandable to humans). The result is that the integration between the new information (stored in XML documents) and the old one is heavy, error prone and time-consuming. Thus, a tool able to help in this task will be surely helpful.

To assist the integration of XML data with other data sources, especially database systems, we are developing a general technique, called Xere (*XML E*ntity *R*elationship *E*xchange). The main goal of Xere is to design a conceptual model representing the XML documents data that can be easily understood and used as a tool for the design, optimization and integration of XML data. To this aim, in this paper we show an algorithm, called *Xere mapping*, that translates an XML Schema into a conceptual model, the Entity–Relationship (ER) model, that offers a comprehensive and understandable artifact for the analysis and integration of the XML documents data.

We choose the ER (conceptual) model as the target of our algorithm instead of a logical model, such as the relational model that is used by many works in this research field (see Section 7), since we would like to be *DB-neutral*. Indeed, our aim is to build a support useful both to DB designers working with any DB type, as well as to designers that need to integrate XML data in frameworks that are not based on DBMS. To this aim, the ER model offers a very good high–level documentation on the data structures, that cannot be expressed using logical models. Moreover, the generated diagram can be combined with other ER diagrams to integrate the new structures with preexisting data. This integration is much more easy to study at the ER level using the associated graphical view than, e.g., on the flat relational model. Finally, the choice of the ER diagram as the target model for our algorithm also allows to map the information given by the XML Schemas in a *natural* way, i.e., obtaining an ER diagram that is human–readable and that preserves all the information originally stored in the XML Schema. This leads to a mapping process that can be documented and certified as sound and complete.

To automatize the process it is necessary that the XML documents' structure is formally defined by a grammar. We decided to adopt XML Schemas [11] rather than DTDs [1] as the starting point of our mapping since DTDs are being progressively replaced by XML Schemas, especially when XML is used to hold data structures. For example, the payload of a SOAP message must be defined using an XML Schema [6]. Moreover, using XML Schemas is convenient since this formalism offers a better expressiveness to describe complex structures 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: actually, most of the XML Schema basic types correspond to a DB data type (see [11]). Finally, since a XML Schema is itself an XML-based formalism, it is very easy to process and manipulate it using well–known XML–based tools.

A preliminary release of the mapping algorithm have been already presented in [14]. In the present paper we show the complete algorithm and its prototypal implementation obtained using the XSLT technology. We also prove the soundness of the mapping. Namely, we show that we are able to correctly retrieve XML documents when they are stored according to the ER diagram created by Xere mapping algorithm.

The paper is organized as follows. In Section 2 we give some preliminary notions about XML, XML Schemas and the ER model. In Section 3 we describe the XML Schema–to–ER mapping algorithm, and then we show it at work on an example in Section 4. The correctness of our approach is discussed in Section 5, where we give a soundness and completeness proof for the Xere mapping algorithm. In Section 6 we briefly describe the design and the XSLT implementation of the mapping process. Finally, Section 7 offers on overview on the related works and Section 8 gives some final remarks on the presented work.

2 Background

The key concepts used in Xere are XML, XML Schemas and the Entity–Relationship (ER) conceptual model. A full description of these three formalisms is beyond the aims of this paper, so in the following we assume that the reader has a good knowledge of them.

However, for sake of completeness, in this Section we recall some basic notions about XML and the ER model. The information reported here is not complete, so the reader should refer, e.g., to [1], [11] and [15] for an in-depth overview of these concepts and formalisms.

2.1 The eXtensible Markup Language

The basic syntax of XML is quite simple and should be familiar to anyone that knows HTML. Well–formed XML documents are composed by elements, attributes and text, and must obey to some simple rules: for instance, each document must have an unique *root* element.

The XML Schema (XSD) formalism uses the concept of *type* as its basic mechanism to define the structure of XML documents. XSD distinguishes among *simple* and *complex types*: the formers are basic non-structured types and include the most common data types used in DBMS and programming languages. Simple types are used to declare attribute and leaf elements (i.e., elements not containing other elements) of the XML documents. Complex types are used to describe the structure of elements, i.e. their *content model*. A content model specifies which elements can be nested inside other elements, and in which order. Content models are created by freely composing elements through three basic models: sequence, choice, and set. A cardinality constraint can be assigned both to the elements and to the content models. The special *mixed* complex type allows the corresponding elements to contain both freeform text and a content model. Finally, XML Schema allows to define non-hierarchical relations between elements using *identity constraints*. In particular, the key construct is used to declare that a specific element is uniquely identified through a set of its attributes and/or sub–elements. Then, the keyref construct can be used to define a relation between two elements, requiring the matching of a set of attributes and/or sub–elements of an element with those in the key of the other element.

Fig. 1 shows an example of XML Schema, defining the XML language for a simple purchase order. This schema can be read as follows:

> A purchaseOrder has a date attribute and is composed by exactly one shipTo element, an optional billTo element, an items and a shippingDetails element, and an optional comment. Both shipTo and billTo contain a name and an Address, that can be possibly substituted by the more specific USAddress. Items contain a list of item elements, each one specifying a product, its code, the required quantity, etc. Finally, shippingDetails contain comments or handling instructions for particular items, referenced using the corresponding item_key.

2.2 Entity–Relationship Diagrams

Entity Relationship Diagrams (ERD), [15], are a simple but powerful *conceptual model*. Conceptual models are used to describe the fundamental concepts

```
<schema>
 <element name="Address">
  <complexType><sequence>
   <element name="street" type="string"/>
   <element name="city" type="string"/>
  </sequence></complexType>
 </element>
 <element name="USAddress" substitutionGroup="Address">
  <complexType><sequence>
   <element name="street" type="string"/>
   <element name="city" type="string"/>
   <element name="state" type="xs:string"/>
   <element name="zip" type="positiveInteger"/>
  </sequence></complexType>
 </element>
 <simpleType name="SKU">
  <restriction base="string"><pattern value="\d{3}-[A-Z]{2}"/></restriction>
 </simpleType>
 <element name="comment" type="string"/>
 <element name="purchaseOrder" type="PurchaseOrderType"/>
 <complexType name="PurchaseOrderType">
  <sequence>
   <element name="shipTo">
    <complexType><sequence>
     <element name="name" type="string"/>
     <element ref="Address"/>
    </sequence></complexType>
   </element>
   <element name="billTo" minOccurs="0">
    <complexType><sequence>
     <element name="name" type="string"/>
     <element ref="Address"/>
    </sequence></complexType>
   </element>
   <element ref="comment" minOccurs="0"/>
   <element name="items" type="Items"/>
   <element name="shippingDetails">
    <complexType><sequence>
     <element name="item" minOccurs="0" maxOccurs="unbounded">
      <complexType>
       <choice>
         <element name="handlingInstructions" type="string"/>
         <element ref="comment"/>
        </choice>
        <attribute name="partNum" type="SKU" use="required"/>
      </complexType>
      <keyref name="Item_Desc_Keyref" refer="Item_Key">
       <selector xpath="."/><field xpath="@partNum"/>
      </keyref>
     </element>
    </sequence></complexType>
   </element>
  </sequence>
  <attribute name="date" type="date"/>
 </complexType>
 <complexType name="Items">
  <sequence>
   <element name="item" minOccurs="0" maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element name="productName" type="string"/>
      <element name="quantity" type="integer"/>
      <element name="Price" type="decimal"/>
      <element name="shipDate" type="date" minOccurs="0"/>
     </sequence>
     <attribute name="partNum" type="SKU" use="required"/>
    </complexType>
    <key name="Item_Key">
     <selector xpath="."/><field xpath="@partNum"/>
    </key>
   </element>
  </sequence></complexType>
</schema>
```

Fig. 1. A sample XML Schema

of a specific domain, their structure, and the relationships among them. ERD have a very intuitive and widely–known graphical representation, that we will use in this paper to visualize the diagrams generated by our algorithm. The basic ERD elements and their graphical representation are shown in Fig. 2. These are:

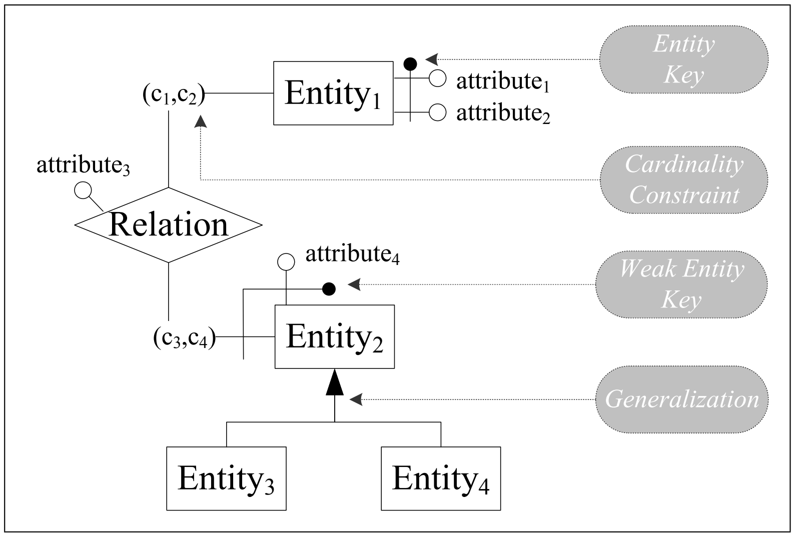


Fig. 2. Entity–Relationship diagrams elements.

- **Entities** represent (collections of) atomic concepts or objects in the considered domain. They are graphically depicted through rectangles that contain the entity name.
- **Relationships** are used to express an association between one or more entities. They are depicted as diamonds connected through lines to the involved entities and containing the association name.
- **Cardinalities** specify the number of relationship instances that an entity can participate in. They are written as (min, max) integer pairs close to the line that connects each entity with each relation. The max cardinality can be unbounded, in this case it is usually written as "n".
- **Attributes** represent the data elements of a particular entity or give information about a specific relationship. They are represented by ellipses that can be attached to any entity or relationship symbol. The attribute name is inside or close to the related ellipse. Attributes can also have a cardinality.

  If one or more attributes constitute the primary key for an entity, i.e. a set of data that uniquely identifies each entity instance, these attributes are linked by a line ending with a black circle as shown in Fig. 2. In many common cases, an entity key definition may also involve one or more of its relationships: in this case, the entity is called *weak*. This means that the entity instances can be unambiguously distinguished only if they are considered within the context given by the associated entities.
- **Generalizations** are used to build generalization (or specialization, if read bottom-up) hierarchies between entities. As depicted in Fig. 2, generalizations are built top–down, with the most general entity on the top. Generalizations can be *total* (i.e. all the parent instances can be classified as one of its children) or *partial* (i.e. there are parent instances that does not correspond to a child's instance).

3 The Xere XML Schema–to-ER mapping algorithm

In this section we describe the Xere mapping algorithm. XML Schemas define patterns of structured content, i.e. data and relations among them. The mapping exploits the information specified in a XML Schema to define a *sound* and *complete* ER diagram, i.e. a data model describing all and only the structural information introduced by the XML Schema. Before proceeding with the technical discussion, let us give some basic considerations about our approach.

3.1 The Xere Mapping Methodology

The Xere mapping aims at providing interoperability between XML Schemas and ER diagrams. The mapping preserves all information provided by an arbitrary XML Schema without including additional information which is not expressly given.

The problem of representing all the information specified by a XML Schema poses several questions. In contrast to ER diagrams, Schemas are a very rich formalism which is able to define recursive types, type hierarchies, usage constraints, etc. Thus, Schemas are located on a much lower abstraction level than ER models, which are usually intended as an intermediate design document. Indeed, not all the constructs used in the Schemas have a precise counterpart in the ER model, which is much simpler. The *choice* content model, for instance, is used in the XML Schema definitions to denote a disjunction between two or more sub–models. Since there is no corresponding construct in the ER model, the generalization is used to gain the same result. Thus, the simulation of the *choice* is obtained by means of auxiliary entities used to represent the root of the generalization hierarchy and the specializations.

However, there are some XML Schema constraints (e.g., attribute types) that cannot be mapped on the ER model. To handle these special cases, we introduce *model annotations*, i.e. simple textual notes attached to the model that contain human-readable descriptions of such constraints. These annotations are intended to document the ER model so that no information is lost from the original XML Schema. Note that using informal annotations is a common practice in ER design, so we are not introducing any extension to the ER formalism.

The mapping algorithm is defined as a set of transformation rules which inductively takes advantage of the syntactical structure of a XML Schema to define the corresponding ER model.

Currently, Xere supports all the core XML Schema features. Only three aspects have been omitted:

- any, anyAttribute. These elements allow to leave underspecified given aspects of the XML Schema and provide it with a limited form of genericity. Roughly speaking, these two elements can be used as placeholders for any element or attribute, respectively. This form of genericity does not have a correspondence in the theory and practice of databases; therefore, it has been not included in the current form of the mapping.
- notation. This is used to specify type information about external entities, typically binary objects. For instance, it could be used to declare which application should handle a particular data format. This kind of detail goes beyond the conceptual model purposes, so it has not been considered.

The overall structure of the Xere algorithm is described in Fig. 3 and 4. The algorithm maps every global XML Schema element to an ER entity. Element attributes become entity attributes and the content model of complex typed elements is mapped using ER relations, generalizations and cardinality constraints. In the following we detail the description of each step of the algorithm shown in Fig. 3 and 4. Note that, for the sake of brevity, we omit from the algorithm description the mapping of two groups of secondary elements:

- the annotation, appinfo, and documentation elements, used to report different kinds of notes on the XML Schema, therefore directly translated to ER annotations,
- the import, include, and redefine elements, used to combine Schemas in various ways, are expanded to obtain a self-contained XML Schema before applying the mapping algorithm.

Mapping XML Schema Elements

Function *MapElement* in Fig. 3 shows the overall element mapping algorithm. In general, each XML Schema element becomes an ER entity. The entity names are based on the corresponding element names, and *local names*, i.e. names prefixed with the parent element name, are used to preserve the scope of visibility of local elements (for example, an element "C" declared inside a global element "A" would result in the creation of an entity labeled "A_C").

```
procedure Xere(Document D)
begin
 foreach global element E in D do
 begin
  MapElement(E)
 end
 MapSubstGroups();
 MapKeyrefs();
 RefineDiagram();
end
procedure MapElement(Element E)
begin
 F = CreateEntity(E);
 if (E has simple type t) then
  AddAnnotation(F,t simple type description);
 else MapContentModel(E,F);
 MapAttributes(E,F);
end
procedure MapAttributes(Element E, Entity F)
begin
 foreach attribute element D in E do begin
  if D is not fixed then begin
   A = CreateAttribute(F,D);
   AddAnnotation(A,D type description);
   if D has a default then AddAnnotation(A,D default value);
   if D is optional then set cardinality of A to (0:1);
  end
  else begin
   create constant value annotation
  end
 end
end
procedure RefineDiagram()
begin
 remove useless auxiliary entities
 merge content-only entities with their parent(s)
end
```

Fig. 3. Xere main steps

Element attributes are then mapped to entity attributes through the *Map-Attributes* function described below. Entities corresponding to elements with simple type also have a special "content" attribute used to store the textual content of the corresponding element.

For each XML Schema element (i.e., entity in the ER) we should provide a primary key that is chosen using the following rules:

```
procedure MapContentModel(Element E, Entity F)
begin
 MapContentParticle(root content model of E, F)
end
procedure MapContentParticle(Particle C, Entity F)
begin
  if C is a sequence then begin
   if C.cardinality <> (1:1) then begin
    F1 = CreateAuxiliaryEntity()
    CreateRelation from F to F1 with cardinality (x:y)
    F=F1
   end
   foreach child D of C do begin
    MapContentParticle(D,F)
   end
  end
  else if C is a choice then begin
   F1 = CreateAuxiliaryEntity()
   CreateRelation from F to F1 with cardinality(C)
   foreach child D of C do begin
    F2 = CreateAuxiliaryEntity()
    CreateSpecialization from F1 to F2
    MapContentParticle(D,F2)
   end
  end
  else if C is an element then begin
   F1 = MapElement(C)
   CreateRelation from F to F1 with cardinality(C)
  end
end
```

Fig. 4. Xere content models mapping algorithm

(1) if the XML Schema explicitly defines a key (using the key construct), that key is used as the primary key for the corresponding ER entity. If the key construct contains values or attributes from other (child) elements, we classify the ER entity as *weak*. In this case a relation is created, if not present, from the weak entity to the entity corresponding to the referenced child element, and it is included in the weak entity primary key.

As an example, in Fig. 8 the key of element A includes its child A_B. The corresponding entity will therefore be weak, and its key will include the relation to the child entity A_B as shown in Fig. 8.

(2) if the element has an attribute of type ID, that attribute becomes the primary key of the corresponding entity.

Remark 1 *Schemas allow the definition of different keys for the same entity, at different* scope levels. *However, the ER theory says that each entity can*

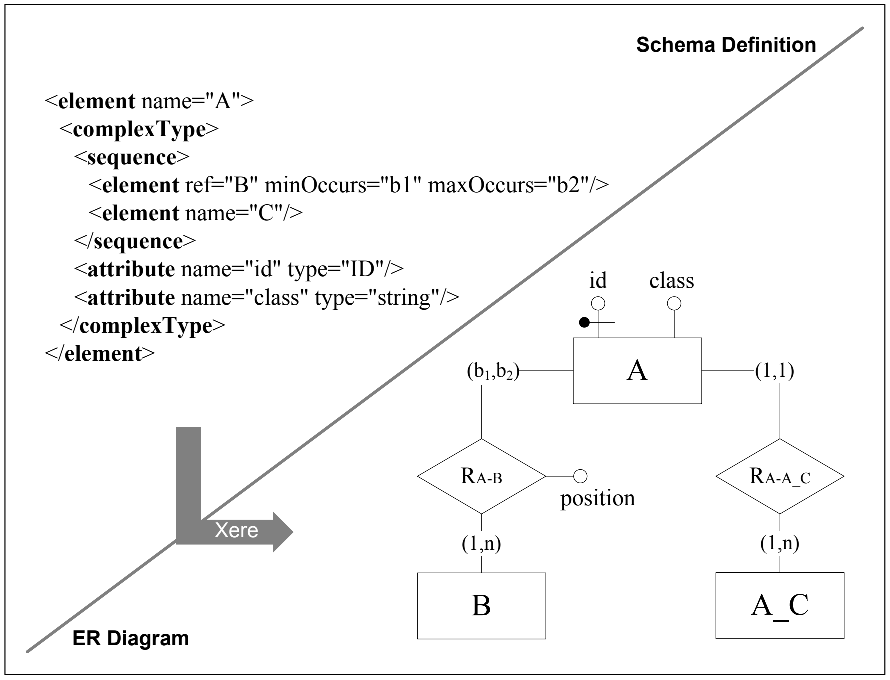


Fig. 5. Mapping of a simple XSD sequence model. The figure also shows the mapping of attributes, entity keys and local elements.

have only one key. Therefore, MapKeyrefs cannot handle multiple keys and the algorithm maps only key *and* keyref *statements having a child like <*selector xpath="."*> (i.e., that give the key and the foreign key references, respectively, for the element they are nested in), whereas it outputs an error message if any other* selector *element is encountered.*

Note that if an element is declared with the nillable attribute set to true, then an appropriate annotation is added to the ER diagram to remark that in the corresponding ER entity all the attributes can be set to null.

The element mapping ends with the most important step, that is the mapping of the element type. If the element has a simple type, the algorithm adds a *content* attribute to the corresponding ER entity and reports the associated type information in an annotation that is added to the ER model and referred to such attribute. Otherwise, the element content model is mapped to appropriate ER structures through the *MapContentModel* function.

Mapping of Element Attributes

The *MapAttributes* function maps each attribute of an element to an attribute of the corresponding entity, using appropriate model annotations to preserve the associated type information. Moreover, the algorithm maps the use, default and fixed properties of the attribute definition in the following way:

- if use is *required* and no fixed values are specified, then the attribute is simply mapped in an entity attribute.
- if use is *optional*, and no fixed values are specified, then the entity attribute is assigned to a $(0, 1)$ cardinality.
- if use is *prohibited* the attribute definition is ignored.
- if a fixed value is specified, then the attribute is actually a constant and should not be included in the entity definition. Therefore, the attribute is not created and an appropriate annotation is added to the model to keep track of the constant value.
- if a default or fixed value is specified, we add an appropriate annotation to the model about this constraint. Moreover, if a fixed value is declared, then the attribute is actually a constant and therefore it is not included in the corresponding entity definition.

Note that we always assume that global attribute references and attribute groups referenced through the attributeGroup element are inline expanded in the element definition. This means that all the attributes are actually defined inside the corresponding element.

Mapping of Content Models

Function *MapContentModel* maps complexType content models by calling function *MapContentParticle* that recursively processes the nested content particles, that is the atomic components of a content model (elements and/or other content models). As a general rule, if a content particle has a cardinality constraint, then *MapContentParticle* reports it on the relation that connects the corresponding entity to its parent entity (that is passed as a parameter to the function). Moreover, if the particle has a multiplicity (maxOccurs greater than one), the instance ordering is preserved in the ER diagram by adding a position attribute to the generated relation.

Each content model particle has its mapping logic. In general, the content models themselves have no corresponding entity in the generated diagram, since they are not objects to be represented, but only structuring elements. However, if a content model has an associated cardinality constraint, then an entity is created for it to allow placing the cardinality as described above. In particular,

- in sequence and all models, all the children particles are recursively mapped and connected through appropriate relations to the parent entity (see Fig. 5 for an example).

  Note that the same mapping technique is used for both sequence and all since these models are identical except for the ordering semantics (i.e., all children can appear in any order in the instance XML documents) that has

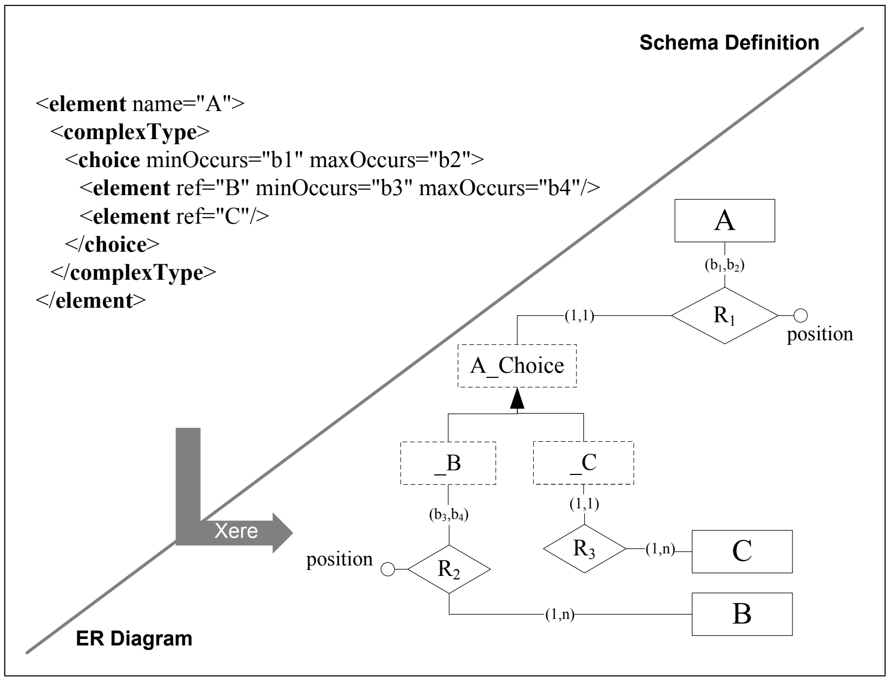

```
<element name="A">
  <complexType>
    <choice minOccurs="b1" maxOccurs="b2">
      <element ref="B" minOccurs="b3" maxOccurs="b4"/>
      <element ref="C"/>
    </choice>
  </complexType>
</element>
```

Fig. 6. Mapping of a XSD choice model

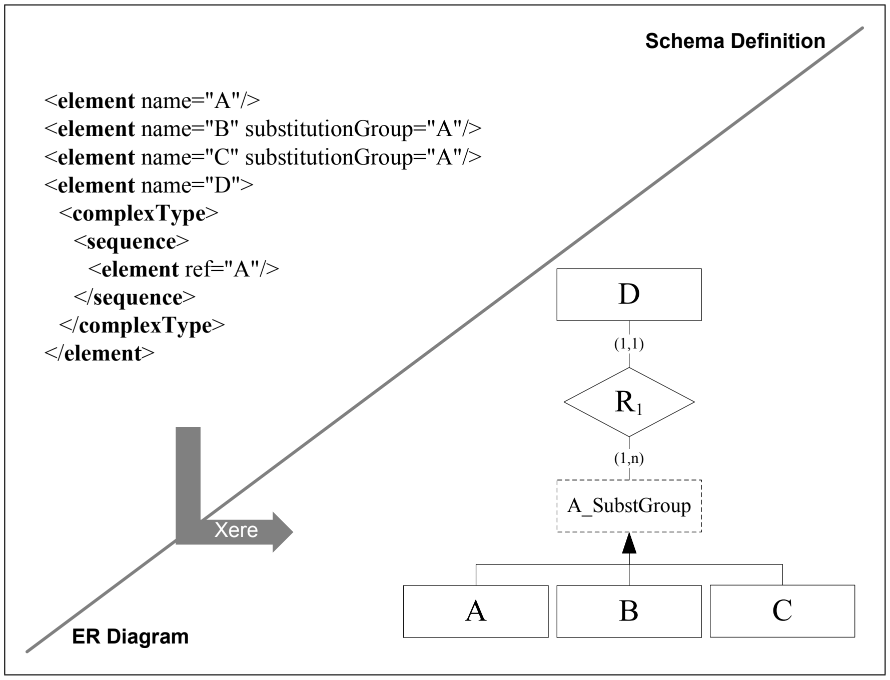
no sense in the ER model.

- to map choice models, the algorithm uses the generalization construct of ER diagrams.

  A new entity is created and attached to the parent entity, and it is specialized in as many other entities as the content children are. Each content model children particle is then recursively mapped and connected through an appropriate relation to the corresponding specialized entity (see Fig. 6 for an example).

  Note that the ER generalization construct may not be always *semantically* consistent with the choice 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.
- element particles defining local element are recursively mapped and connected to their parent entity through an appropriate relation. Global element references are simply translated into a relation from the parent entity to the entity corresponding to the global element (i.e., global elements are created only once, then connected to their references via relations).

As a special case, mixed content models allow freeform text to appear anywhere among the child elements, which must however be consistent with the specified model. In this case, before mapping the model we modify it so that a special distinct "text" element is allowed anywhere plain text can appear. This allows to use the standard mapping rules given above to recreate the mixed semantics in the resulting ER model.



```
<element name="A"/>
<element name="B" substitutionGroup="A"/>
<element name="C" substitutionGroup="A"/>
<element name="D">
  <complexType>
    <sequence>
      <element ref="A"/>
    </sequence>
  </complexType>
</element>
```

Fig. 7. Mapping of a XSD substitution group.

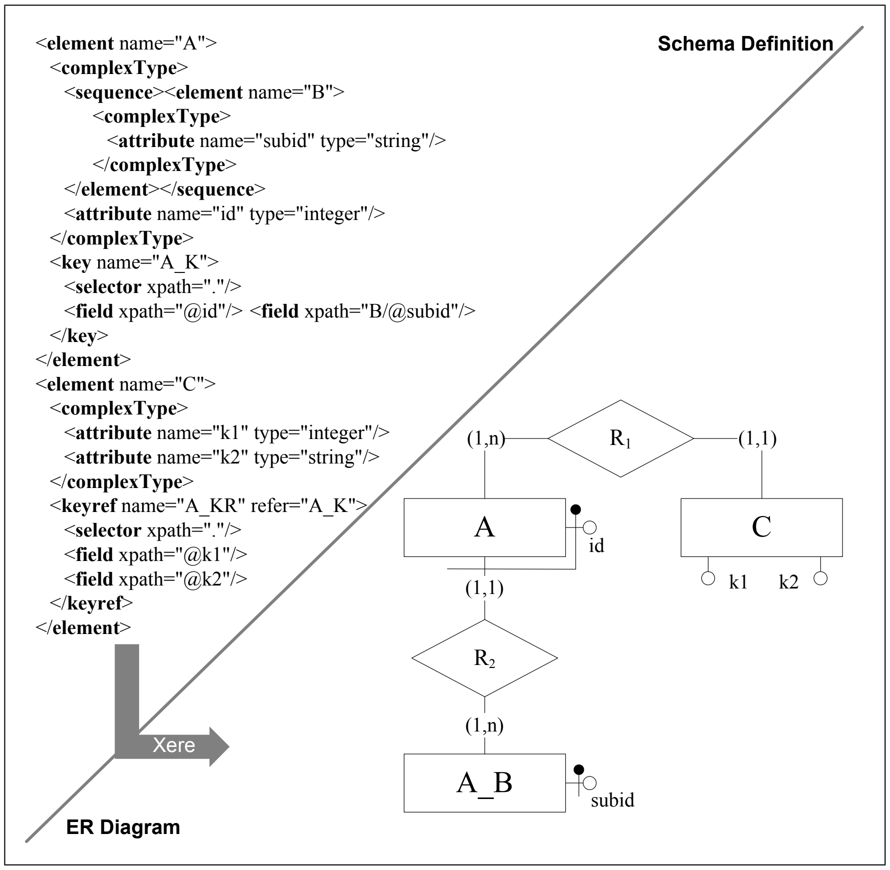
Remark 2 *The algorithm also takes care of complex types derived by extension (that are flattened by merging all the types in the derivation hierarchy) and by restriction (handled as natural by rewriting the entire complex type with appropriate restrictions), and inline expands referenced global complex types. However, these details will not be explained for the sake of simplicity.*

Mapping of Substitution Groups

Function *MapSubstGroups* performs a post-processing on the generated ER diagram to map XSD substitution groups using total generalizations (see Fig. 7). In particular, the elements belonging to a substitution group become specializations of a new general entity introduced to map the substitution group itself. Relations to any of the substitution group elements are then redirected to the generalization root, as shown in Fig. 7.

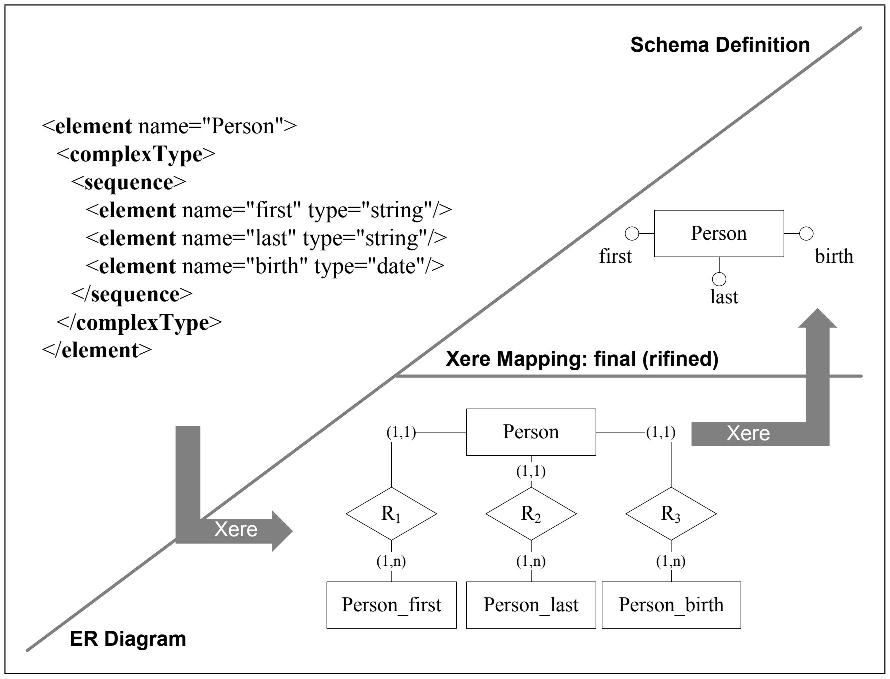
Mapping of Identity Constraints

Function *MapKeyrefs* creates new relations in the generated ER diagrams to map the key references defined by the keyref construct. The new relations are named adequately, as shown in Fig. 8.



```
<element name="A">
  <complexType>
    <sequence><element name="B">
        <complexType>
          <attribute name="subid" type="string"/>
        </complexType>
    </element></sequence>
    <attribute name="id" type="integer"/>
  </complexType>
  <key name="A_K">
    <selector xpath="."/>
    <field xpath="@id"/> <field xpath="B/@subid"/>
  </key>
</element>
<element name="C">
  <complexType>
    <attribute name="k1" type="integer"/>
    <attribute name="k2" type="string"/>
  </complexType>
  <keyref name="A_KR" refer="A_K">
    <selector xpath="."/>
    <field xpath="@k1"/>
    <field xpath="@k2"/>
  </keyref>
</element>
```

Fig. 8. Mapping of the key/keyref XSD constructs



```
<element name="Person">
  <complexType>
    <sequence>
      <element name="first" type="string"/>
      <element name="last" type="string"/>
      <element name="birth" type="date"/>
    </sequence>
  </complexType>
</element>
```

Fig. 9. Refinement of nested elements with simple type.

Finally, function *RefineDiagram* manipulates the generated diagram by applying some simple transformations that are used both to compress the final result (in terms of number of entities and relationships) and to make it appearing more human-made and therefore natural.

The *MapElement* function creates an entity for each element with complex or simple type. However, elements with simple type contain only a content attribute. Actually XML Schema designers often use local, nested elements with simple types instead of simple attributes only to give the XML documents a more modular and better-looking structure. Therefore, *RefineDiagram* compresses such kind of entities (and the corresponding relations) making them attributes of their parent entity and mapping possible cardinality constraints to attribute cardinalities (see Fig. 9).

The *MapParticles* function introduces some auxiliary entities to correctly place the occurrence constraints. However, if these constraints are set to $(1, 1)$, *RefineDiagram* removes the auxiliary entity, since it is not needed to respect the XML Schema definition.

4 An Example

In this section we introduce a complete example of XSD to ER mapping obtained through the Xere algorithm. We first describe the real world objects we want to capture with our XML documents. Then we give the complete XML Schema that describes and validates the documents we need. Finally, we apply the Xere mapping to the XML Schema and show the resulting ER model.

The example presented has been obtained from the W3C "International Purchase Order" schema used to exemplify the XML Schema language [11]. With respect to the original example, we removed some secondary details and added keys, key references, substitution groups, etc.

4.1 The Problem

We want to describe with an XML language a general *purchase order*. A purchase order is a business document used to purchase one or more *products* from a single *supplier*. The document can contain two distinct delivery addresses: a mandatory *shipment address*, which is the address the products will be sent to, and an optional *billing address*, which is where the payment invoice has to be sent, if different from the shipment address. Since the document can be used for both national and international orders, we should take into account the different address formats used in various countries, so that the address parts are always distinct and well characterized in the XML structure. The main part of the document contains the list of the ordered products: each list item includes the *code* of the product and its *textual description*, the *price*, the required *quantity*, and a *shipment date*. Finally, the last section of the document can contain *comments* or special *handling instructions* referred to some of the ordered products.

4.2 The XSD model

The XML Schema that formalizes the purchase order document described above is shown in Fig. 1. We use an element called purchaseOrder as the XML document root. The root contains four sub–elements: shipAddress, billAddress, items and shippingDetails. The shipAddress and billAddress elements contain the shipping and the (optional) billing addresses, respectively. These elements in turn contain an Address element, which can be substituted with a more specialized USAAddress. The items element contains a sequence of item elements that describe the required product details. In particular, each item has a partNum attribute to specify the product code. Finally, the shippingDetails element can contain zero or more item elements, each referring to one of the items listed under the items list. This reference is forced using the key/keyref XML Schema constructs. Each shippingDetails item contains a comment or some handlingInstructions.

4.3 The ER model obtained with Xere

The ER model generated by the Xere algorithm applied to the XML Schema described above is shown in Fig. 10. We may highlight some details of the resulting diagram:

- The substitution group for the Address element has been translated in a generalization of the auxiliary entity Address_substGroup.
- The sequence models have been translated using simple relations, like R1. When one of the sequence children has a multiplicity, as for item in the items content model, the corresponding relation has a pos attribute (R21 in the example).
- The choice model in the shippingDetails element has been mapped to a specialization using the auxiliary root entity purchase-Order_shippingDetails_item.

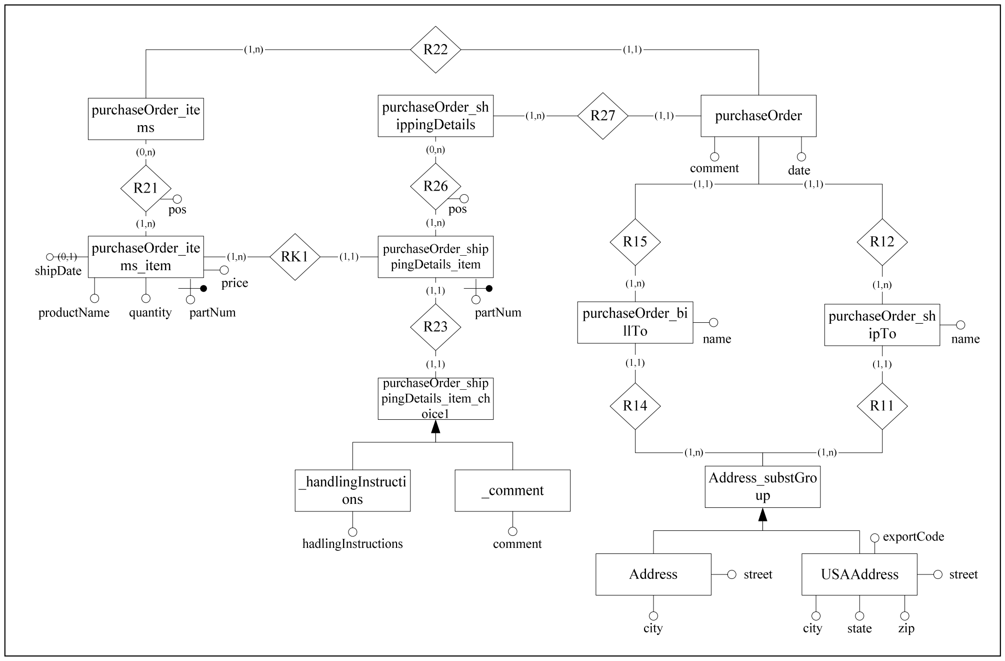


Fig. 10. Xere mapping of the international purchase order XML Schema.

- The key definition for the element items/item has been mapped by setting the partNum attribute as the corresponding entity primary key.
- The keyref construct in the shippingDetails/item element generated a corresponding relation RK1 between the two entities involved.

4.4 Discussion

The case study given in this Section contains almost all of the XML Schema constructs, nesting as well as global definitions, text-only elements and substitutions groups. Therefore, the "purchase order" offers a fairly representative example of how the Xere algorithm works on a complex and general XML Schema definition. The resulting diagram is very simple, and captures all the specifications in a compact and realistic way. We remark that, besides some secondary details, the diagram shown in Fig. 10 is what a professional ER designer would create from the above specifications, whereas in our case the ER diagram has been automatically generated from a XML Schema definition.

It is worth noting that the Xere approach is modular and its behavior does not rely on the overall XML Schema definition, since the algorithm works locally to map and then to refine each element, attribute, key, etc. This makes Xere scalable and suitable for any XML Schema, regardless of its size and complexity. Indeed, the aim of Xere is not to simplify the information given in the XML Schema, since it mostly works at syntactic level, to give a good conceptual view on the XML Schema contents. If the XML Schema is itself complex and large, so it will be the ER diagram. However, we feel that, if the XML Schema definition is good (this is, obviously, a precondition of any schema-transformation algorithm), then the resulting ER will always be readable and semantically clear. To verify this we are currently experimenting Xere on many different Schemas that can be found on various web repositories.

5 Soundness and Completeness

In this section we make two formal statements on the Xere mapping algorithm described in Section 3.2. Namely, we show that the algorithm is *complete*, i.e. it is able to translate in an ER diagram every given XSD (that does not contain the minor XSD aspects listed in Section 3.2), and *sound*, that is it *properly* translates the given XSD. To this end, we will show that the ER diagram $\mathcal{E}$ obtained from the XSD $\mathcal{S}$ allows to store and retrieve the information contained in any XML document which is valid w.r.t. $\mathcal{S}$.

Proposition 3 (Xere Mapping Completeness) *Let $\mathcal{S}$ be a well–formed XSD (w.r.t. the W3C XML Schema Specification [11]). Suppose that $\mathcal{S}$ does not contain elements of type* any, anyAttribute *or* notation *and that all the* key, unique *and* keyref *statements in $\mathcal{S}$ have a only child like* <selector xpath=".">*(the aim of these limitations has been discussed in Section 3.2). Then, the procedure* Xere *of Fig. 3 correctly terminates on input $\mathcal{S}$, and translates all the $\mathcal{S}$ elements.*

PROOF. We only sketch the proof, since taking into account all the XSD aspects would require too much space. We can note that the procedure Xere of Fig. 3 correctly handles all the XSD elements listed, except those we explicitly choose not to deal with (i.e. any, anyAttribute and notation). Moreover, the procedure is recursive, thus all the possible nestings are taken into account; namely, all the base elements of $\mathcal{S}$ are translated into adequate ER constructs, while the parent–child relations of $\mathcal{S}$ result in ER relations. This ends our sketch of proof.

To prove the soundness of the Xere mapping we proceed as follows. We consider the ER $\mathcal{E}$ output by procedure Xere of Fig. 3 when the input is a given XSD $\mathcal{S}$ (with the same limitations of Proposition 3). Then, we show that $\mathcal{E}$ may be used to store the information contained in every XML document $\mathcal{D}$ valid w.r.t. $\mathcal{S}$. Moreover, starting from the information stored in $\mathcal{E}$, we prove that is possible to obtain an XML document $\mathcal{D}'$ which is *equivalent* to $\mathcal{D}$. In this way, we show that $\mathcal{E}$ effectively maintains the structure and the information contained in of $\mathcal{S}$.

Before going on with the proof, we have to clarify when two XML documents are equivalent. Roughly speaking, two XML documents are equivalent if they contain the same information and the same nestings. The attribute order in the documents should not be considered, whereas the element ordering is significant only when it derives from a sequence complex content, or there is an element repetition due to a maxOccurs $\neq 1$ setting. This notion is formalized in the following Definition.

Definition 4 (Document Equivalence) *We say that two XML documents $\mathcal{D}_1$ and $\mathcal{D}_2$, both valid w.r.t. the same XML Schema $\mathcal{S}$, are* equivalent *iff their root elements are* equivalent. *In turn, two XML elements E_1 and E_2 of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, are* equivalent *iff all the following conditions hold.*

- *E_1 and E_2 have the same tag.*
- *E_1 and E_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 defined in $\mathcal{S}$.*
- *If E_1 and E_2 have a simple content, then the contents are identical.*
- *If E_1 and E_2 have a complex content, let $E_1^C = \{E_{11}, \ldots, E_{1n}\}$ and $E_2^C = \{E_{21}, \ldots, E_{2m}\}$ be the children of E_1 and E_2, respectively. Then, $m = n$ and there exists a bijective mapping f between E_1^C and E_2^C s.t., for all $1 \leq k \leq n$*
  - · *E_{1k} and $f(E_{1k})$ are equivalent.*
  - · *if the complex content of E_1 and E_2 has a sequence model, then $f(E_{1k}) = E_{2k}$.*
  - · *if E_1^C and E_2^C contain repetitions of some or all of the complex content elements (due to a maxOccurs $\neq 1$ declared in $\mathcal{S}$), then f has to preserve the relative ordering of the repeated elements (this rule will be better explained in the proof of Proposition 5).*

We are now ready to give our soundness theorem.

Proposition 5 (Xere Mapping Soundness) *Let $\mathcal{S}$ be a XSD satisfying the same hypothesis of Proposition 3 and $\mathcal{E}$ the ER diagram generated by procedure Xere of Fig. 3 applied to $\mathcal{S}$. Then, if $\mathcal{D}$ is a XML document valid w.r.t. $\mathcal{S}$, all the information in $\mathcal{D}$ can be stored in an instance of $\mathcal{E}$, and it is possible to retrieve from that instance of $\mathcal{E}$ a XML document $\mathcal{D}'$ equivalent to $\mathcal{D}$.*

PROOF. The proof is sketched using structural induction on the syntax of $\mathcal{D}$.

As induction basis, we suppose that $\mathcal{D}$ has only one element E, having a simple content (i.e., not containing any nested element) and n attributes. Thus, $\mathcal{S}$ has to be similar to the XML Schema fragment of Fig. 11. Procedure Xere translates $\mathcal{S}$ in the ER diagram $\mathcal{E}$ of Fig. 11, consisting of an entity

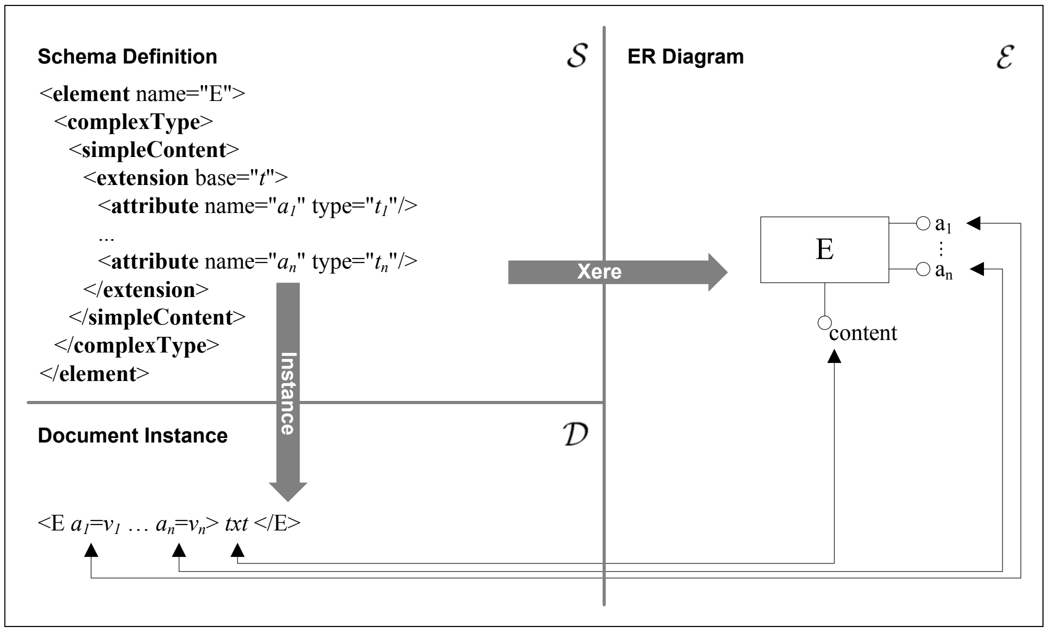


Fig. 11. Mapping of a simple element.

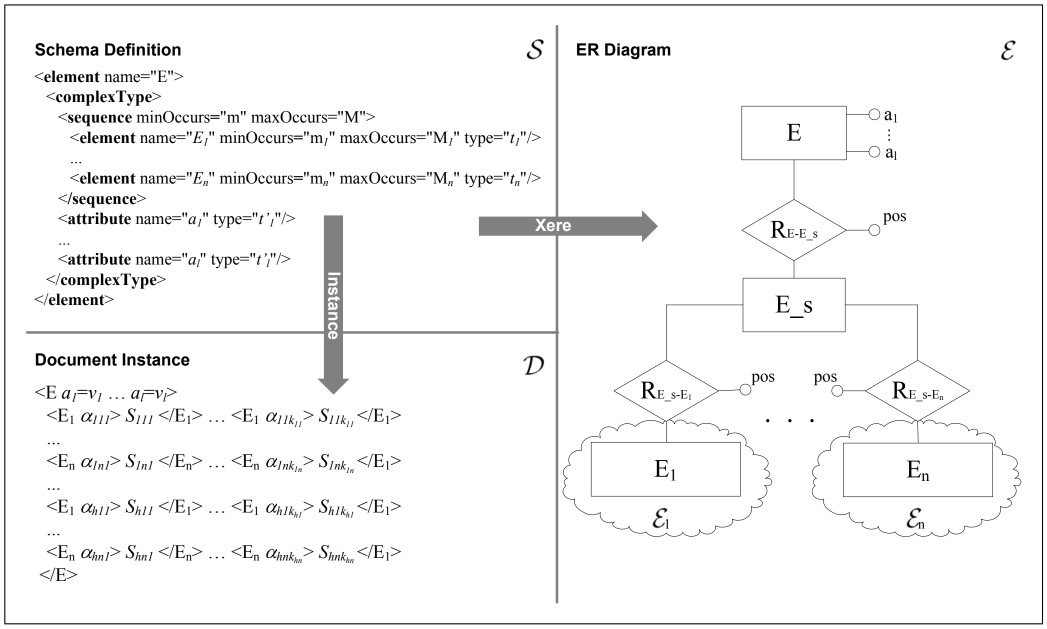
E with $n+1$ attributes, namely *content* and $a_1, \ldots, a_n$, which correspond to the textual content and to the attributes of E, respectively. Thus, all the information contained in E can be properly stored in (an instance of) $\mathcal{E}$. On the other hand, the information stored in an instance of $\mathcal{E}$ can be trivially used to build up an XML document $\mathcal{D}'$ equivalent to $\mathcal{D}$ by properly retrieving the attribute values and the content of E. The only difference between $\mathcal{D}$ and $\mathcal{D}'$ could be in the ordering of $a_1, \ldots, a_n$, which is not taken into account for the XML document equivalence (see Definition 4).

For the induction step, we suppose that the thesis holds for the elements nested in the root element E of $\mathcal{D}$, and we prove that the thesis holds also for E.

We begin by supposing that E has a sequence content model. Thus, the most general XML Schema for $\mathcal{D}$ is like the one named $\mathcal{S}$ in Fig. 12. The corresponding general form for $\mathcal{D}$ is also shown in Fig. 12, where each α_{xyz} is a sequence of attributes, $m \leq h \leq M$ and $m_i \leq k_{xi} \leq M_i, \forall x \in [1, h]$. Note that if $m_i = 0$ then for some $i \in [1, n]$ we can have $k_{xi} = 0$, i.e., element E_i could not appear in some cases. This would not change our proof significantly.

The Xere algorithm would translate this XML Schema fragment in the ER diagram $\mathcal{E}$ shown in Fig. 12. Note that the n elements $E_1, \ldots, E_n$ inside the sequence model generate n ER subdiagrams $\mathcal{E}_1, \ldots, \mathcal{E}_n$. These subdiagrams are rooted in n ER entities $E_1, \ldots, E_n$, linked by n relations $R_{E\_s-E_1}, \ldots, R_{E\_s-E_n}$ to the auxiliary entity E_s. E_s is then directly linked to the root entity E.

Storing and retrieval of the E attributes are correctly performed by the same arguments of the induction basis. As for the E content model, let $1 \leq x \leq h$,



Schema Definition

```
<element name="E">
  <complexType>
    <sequence minOccurs="m" maxOccurs="M">
      <element name="E1" minOccurs="m1" maxOccurs="M1" type="t1"/>
      ...
      <element name="En" minOccurs="mn" maxOccurs="Mn" type="tn"/>
    </sequence>
    <attribute name="a1" type="t'1"/>
    ...
    <attribute name="al" type="t'l"/>
  </complexType>
</element>
```

Document Instance

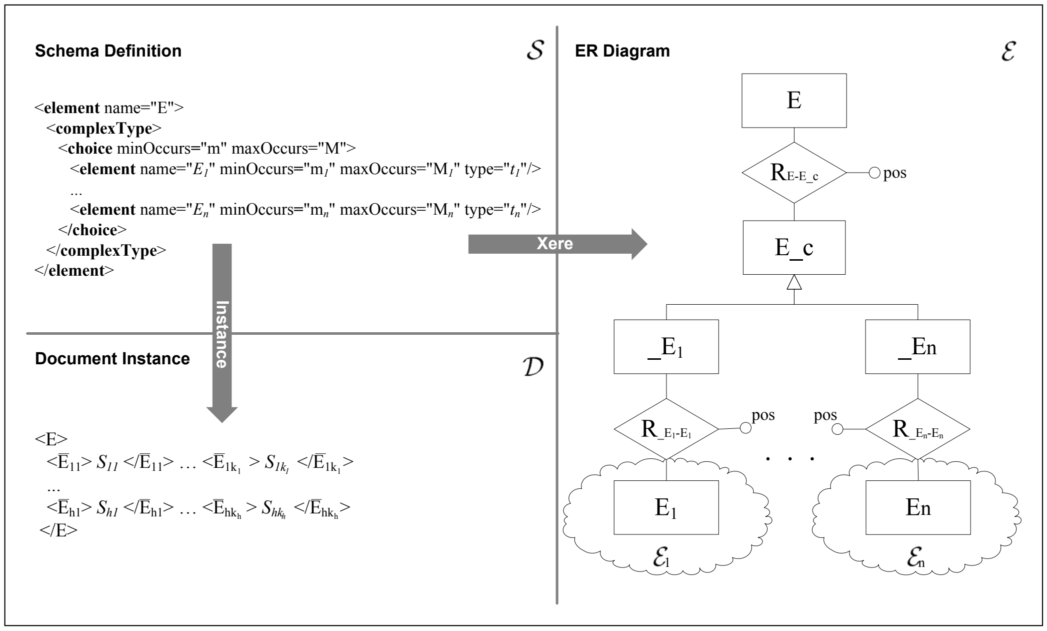
```
<E a1=v1 … al=vl>
  <E1 α111> S111 </E1> … <E1 α11k11> S11k11 </E1>
  ...
  <En α1n1> S1n1 </En> … <En α1nk1n> S1nk1n </E1>
  ...
  <E1 αh11> Sh11 </E1> … <E1 αh1kh1> Sh1kh1 </E1>
  ...
  <En αhn1> Shn1 </En> … <En αhnkhn> Shnkhn </E1>
 </E>
```

Fig. 12. Mapping of a sequence model.

$1 \leq y \leq n$ and $1 \leq z \leq k_{xy}$. Then, the information S_{xyz} as well as the attributes values α_{xyz}, contained in a tag E_y, will be correctly stored and retrieved, by induction hypothesis, in an instance of the corresponding ER subdiagram $\mathcal{E}_y$. In particular, the content S_{xyz} is stored in an instance of $\mathcal{E}_y$, which is connected through an instance of relation $R_{E\_s-E_y}$ having $pos = z$ to an instance of E_s, that is in turn connected to E through a relation instance R_{E-E_s} having $pos = x$. This implies that two different contents S_{xyz} and $S_{x'y'z'}$ with $x \neq x'$ and/or $y \neq y'$ and/or $z \neq z'$ cannot overlap, and their ordering is stored through the pos attributes (the ordering among the elements $E_1, \ldots, E_n$ is not stored, since it is fixed and can be retrieved from the XML Schema $\mathcal{S}$). In the same way, the attributes α_{xyz} are correctly stored and retrieved from the entity E_y. Thus, the document equivalence (see Definition 4) is preserved.

Note that diagram refinements could slightly change the shape of the diagram shown in Fig. 12. Indeed, if $M = 1$, then the generated ER diagram does not contain neither E_s nor R_{E-E_s} and the relations $R_{E-E_1}, \ldots, R_{E-E_n}$ are directly linked to E. Moreover, if E_i has a simple content type, it will be collapsed into an attribute of the entity E_s or E, depending on the value of M. In both cases, showing that the document storage and retrieval continues to satisfy the document equivalence property requires only trivial modifications to the given proof.

Consider now the case in which E has a choice content model. Thus, the corresponding most general XML Schema for $\mathcal{D}$ has to be like the one of Fig. 13, where $m \leq h \leq M$ and $\forall x \in [1, h]\, \exists i \in [1, n]$ s.t. $m_i \leq k_x \leq M_i \wedge \forall y \in [1, k_x]\, \overline{E}_{xy} = E_i$. However, we suppose the content model not to contain



```
<element name="E">
  <complexType>
    <choice minOccurs="m" maxOccurs="M">
      <element name="E_1" minOccurs="m_1" maxOccurs="M_1" type="t_1"/>
      ...
      <element name="E_n" minOccurs="m_n" maxOccurs="M_n" type="t_n"/>
    </choice>
  </complexType>
</element>
```

$$\begin{array}{l} \langle E \rangle \\ \quad \langle \overline{E}_{11} \rangle\, S_{11}\, \langle /\overline{E}_{11} \rangle \ldots \langle \overline{E}_{1k_1} \rangle\, S_{1k_1}\, \langle /\overline{E}_{1k_1} \rangle \\ \quad \ldots \\ \quad \langle \overline{E}_{h1} \rangle\, S_{h1}\, \langle /\overline{E}_{h1} \rangle \ldots \langle \overline{E}_{hk_h} \rangle\, S_{hk_h}\, \langle /\overline{E}_{hk_h} \rangle \\ \langle /E \rangle \end{array}$$

Fig. 13. Mapping of a choice model.

attributes, since their handling is the same as in the sequence case. In Fig. 13 we also report the general form for $\mathcal{D}$.

The Xere algorithm would translate this XML Schema fragment in the ER diagram $\mathcal{E}$ shown in Fig. 13.

To show that $\mathcal{E}$ is fit to store and retrieve the information in $\mathcal{D}$, we fix $1 \leq x \leq h$ and $1 \leq y \leq k_x$. Suppose that $\overline{E}_{xy} = E_i$. Then, the information S_{xy} contained in the tag E_i will be correctly stored and retrieved, by induction hypothesis, in the ER subdiagram $\mathcal{E}_i$. In particular, S_{xy} is stored in an instance of $\mathcal{E}_i$, connected through an instance of the relation $R_{\_E_i - E_i}$, having $pos = y$, to an instance of E_c (we are collapsing the generalization into E_c for sake of simplicity), which is in turn linked to E through an instance of R_{E-E_c} having $pos = x$. Thus, the information contained in two different elements $\overline{E}_{xy}$ and $\overline{E}_{x'y'}$ with $x \neq x'$ and/or $y \neq y'$ and/or $\overline{E}_{xy} = E_i, \overline{E}_{x'y'} = E_j, i \neq j$ cannot overlap and the element ordering is again preserved by the *pos* attributes. Thus, the document equivalence (see Definition 4) is preserved.

Finally, the all content model is mapped exactly as the sequence model, but it is not necessary to maintain the order in the content model elements. Then, the retrieved XML document will be equivalent to the starting one, according to Definition 4.

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

6 Design and Implementation

To test the Xere algorithm, we also implemented a prototypal tool that translates XML Schemas to ER diagrams, described in an appropriate language, and finally displays them. This allowed to debug and refine our approach, as well as to create the examples used in this paper. We designed the Xere mapping tool as a multi–step transformation. The design we propose follows the pipes and filters architectural pattern [16] where each processing step is encapsulated in a filter component and data is passed through pipes between adjacent filters. The components (Filters) read streams of data on input producing streams of data on output, making local incremental transformation to input stream. Instead, connectors (Pipes) transmit outputs from one filter to input of other.

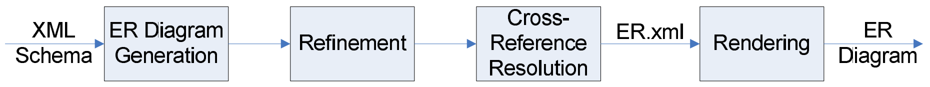


Fig. 14. Software Architecture for the Xere mapping.

Fig. 14 shows the software architecture for the Xere mapping. The system is composed by four different components in sequence (namely pipeline), each implementing a different processing step:

(1) *ER Diagram Generation* takes as input the initial XML Schema and outputs an intermediate file where ER entities, attributes, relationships and specializations (together with their cardinalities) are identified. This step implements the *MapElement* and *MapSubstGroups* functions described in Section 3.2.
(2) *Refinement* creates an ER diagram represented in XML, that is nearly the final output of the algorithm, by applying function *RefineDiagram* shown in Fig. 3 to the output of the *ER Diagram Generation* step.
(3) finally, the *Cross-Reference Resolution* step takes the output of *Refinement* and generates the final ER diagram. Its task is to resolve the cross–references due to the use, in the starting XML Schema, of the keyref construct (function *MapKeyrefs* described in Section 3.2).
(4) Finally, the last step *Rendering* draws the resulting ER diagram.

We implemented each step by means of XSLT [5]. The algorithm is defined inductively over the syntactical structure of the processed documents and the XML transformation stylesheets represent the natural choice to manipulate well–formed XML documents especially when the manipulation is syntax–driven. Also the last step is implemented by a stylesheet using SVG [17]. The data streams a filter passes to its successor in the pipeline is an XML file whose structure describes an ER diagram. Describing such language is out of the scope of this paper, but the reader can find the corresponding XML Schema at the url `http://dellapenna.univaq.it/xere/` where it is also possible to try the algorithm online.

7 Related Works

As we noted in the introduction, most of the work in our field is aimed at translating XML in some other formalism to directly store it in a database. Only few works try to map XML into conceptual structures, like the ER diagrams used by Xere, and use them for more general purposes. For instance, [18] tries to give a support to data integration between XML and UML, and [19] does the opposite by mapping UML class diagrams to XML DTDs.

Since most of the related works on the XML document mapping deal with some kind of translation from XML to database systems, in this section we outline some of the most interesting works and technologies related to the XML–to–DBMS issue. We first give an overview on the current support for XML storage given by the leading commercial DBMS, and then list some remarkable research works in this area.

7.1 XML Support in Relational Databases

The XML support has been introduced in almost all the commercial DBMS in few years. XML documents are basically managed by DBMS using two different storage methodologies: *document–centric* or *data–centric*. In the former case, documents are stored as character entities in CLOB (Character Large OBject) fields. These fields can be manipulated using the common DBMS string handling functions, and possibly queried using XPath. This technique is useful when XML data are often accessed as entire documents, i.e. queries do not address the document structure.

The second technique, data–centric, stores the XML document data in relational structures so that they can be better indexed and searched using the DBMS query language features. This technique is the most suitable for XML documents describing complex data structures. To map XML documents to relational structures, many DBMS (e.g. Oracle [8]) use an intermediate mapping on the object–relational model: this way is very natural since XML elements can be easily seen as objects with fields (attributes, textual content, etc.) and aggregated objects (the child elements).

Oracle 9i [8] with XDB [20] offers an advanced built–in support for XML through the special type *XMLType*. This type can be assigned to any column, and XML data can be converted and assigned to XMLType columns using the `xmltype` constructor. The way XML documents are actually stored in the XMLType fields can be both data or document–centric: if a XML Schema is associated to the XMLType, the DBMS uses it to create an object–relational mapping of the XML data, otherwise the data are stored as a CLOB. To use a particular XML Schema in an XMLType field, Oracle requires it to be first registered through the `dbms_xmlschema` package. The DBMS creates a mapping of the XML Schema structure and datatypes through objects and relational tables and modifies the XML Schema with special attributes that indicate the way each element is mapped. These attributes can be possibly modified by the user to change the mapping and use other tables and/or datatypes, and represent the only documentation that Oracle gives about the created mapping. However, it is very difficult to change the XML Schema used by an XMLType. This "type update" is not common with standard database types, but can be very common with XML Schemas that are subject to evolution and extension. Querying and updating XML documents in Oracle requires the use of complex functions like `decode` and `extractvalue` in support of the standard `select` statement. Moreover, the new keyword `updatexml` must be used to build XML–specific update statements.

Another commercial DBMS that offers a good support to XML is Microsoft SQLServer 2000 [10, 21]. This DBMS can use both annotated XML Schemas or a specifically–designed restriction of Schemas, called *XML–Data Reduced* (*XDR*) [22], to drive data extraction from the DBMS. XML data can be stored in the database in CLOB fields or accessed from external files. The XML documents are transformed "on the fly" in *virtual tables* using the `openxml` rowset provider, and then queried through the standard SQL statements. The mapping of XML documents to these virtual tables is driven by annotated XSD or XDR schemas where the user specifies the target relation and field of each XML element and attribute.

The most remarkable XML–supporting feature in SQLServer is the XML extraction mechanism. All the data stored in a database can be retrieved as XML fragments using the `for xml` keywords in the query statement. The data can be organized in a simple XML structure that directly reflects the selected fields (i.e. each column becomes an element with the same name, etc.) or can be restructured using an explicit mapping included in the query statement. Finally, the user can define special *XML views* on the data by specifying a mapping through an annotated XML Schema.

Other DBMS that offer a more limited support to XML are e.g. Sybase [23] and IBM DB2 [9, 24]. We can make some final considerations on the XML support included in the leading commercial DBMS. The XML data mapping is often driven by an XML Schema and the mapping target is directly the relational or object–relational model of the database. However, the mapping mechanism is not completely transparent and often lacks of documentation.

The mapping can be completely up to the user, or created automatically. In both cases, the resulting relational schema does not preserve or document many of the advanced structural constraints on the data specified by the XML Schema.

Native XML databases are a new generation of DBMS specifically designed to store XML data. Native XML databases have the XML document as their main storage unit (like the rows of relational DBMS), preserve the physical structure of XML documents as well as comments, processing instructions, DTDs, etc. and can also store XML documents without a schema. The main (and in most cases the only) manipulation and querying mechanisms offered by native XML DBMS are the native XML technologies, such as XPath, the DOM, or XML-specific APIs. One of the most known and refined native XML DBMS is Tamino [7]. However, the application of these new DBMS is currently limited, since they cannot efficiently interface with legacy data and relational databases.

7.2 XML Document Mapping Techniques

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. In fact, there were several studies on the management of structured documents even before XML emerged [25]. 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 design relational database schemas for XML documents. The first approach, namely the *structure-mapping approach*, creates 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, [12, 13], and a database schema is defined for each XML document structure or DTD. More sophisticated mapping methods have also been proposed, where database schemas are designed based on detailed analysis of DTDs, [26]. There are also more general approaches to store structured or semi–structured data in relational databases, that also work with XML documents. For instance, [27] uses an intermediate language, STORED, to describe the mapping from semi–structured data to relational structures. The mapping definition is accompanied by an overflow mapping that ensures a lossless storage. The authors show how STORED mappings and overflow mappings can be built using data mining and other techniques applied to instance documents. Overflow mappings can also take advantage of a pre–existing DTDs. To this category of papers also belongs [28], which is based on a different intermediate language (Hypergraph-based Data

Model, HDM) for the translation; however, [28] directly addresses XML documents, without considering neither DTDs nor XSDs.

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, e.g., [29], or the "edge approach", [30], in which edges in XML document trees are stored as relational tuples. A more recent research using this approach is [31], 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. 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.

The current research on XML and database integration also presents some interesting techniques to find the best storage mapping for XML documents. Since there are many ways to map XML trees on flat relations, a variety of heuristic techniques can be applied to find an optimal mapping with respect to a given data access model. For example, [32] introduces an extension of XML schemas that embeds statistical information about the data. The authors describe a set of heuristics that exploit these information to find a suitable mapping for the XML Schema.

Moreover, there are currently a number of papers addressing the opposite problem, that is mapping relational structures to XML documents. These techniques aim at the creation of XML views of relational databases for two main purposes: integrate "legacy" data with XML documents in XML native databases, and allow the use of XML query languages such as XQuery [33] on relational databases. Among other approaches, we may cite SilkRoute [34], that the authors present as a "general, dynamic and efficient tool for viewing and querying relational data in XML". The paper describes a language for defining XML views on relational data and gives an algorithm that composes mapping definitions with XQuery statements to obtain new mappings. Moreover, in [35] the authors describe an efficient translation of such mappings to a set of SQL queries to be applied on the original relational data.

8 Conclusions and Future Works

In this paper, we presented a rule–based recursive algorithm that maps an arbitrary XML Schema to an ER diagram. The given algorithm is able to translate almost all the complex XML Schema constructs using the full expressiveness of the ER model. The resulting diagram is self–explanatory, human–readable, and easily manipulable by database designers to analyze and optimize the XML data representation, and to integrate it with other data sources. All these features are hard or impossible to obtain with the DBMS–storage–oriented mapping techniques that target to the *flat* relational model.

Moreover, the mapping is proved sound and complete w.r.t. the document identity and XML Schema definition, respectively. The document identity defines a congruence relation with classes of isomorphic documents. Indeed, future works include some investigations on the formal property of such congruence relation, since different interoperability styles, possibly a hierarchy of styles, may be given by means of different congruences denoting different XML representations of the same data.

Once the Xere algorithm will be fully implemented, refined and tested, we could apply it in many data analysis, optimization and integration contexts. For instance, in the database context, we plan to show how the Xere–generated diagrams can be used for documentation and data integration, and also as an helper tool to generate the code that allows transparent storage, retrieval and querying of XML documents in relational DBMS.

References

[1] W3C, eXtensible Markup Language (XML), http://www.w3.org/TR/1998/REC-xml-19980210 (1998).

[2] ISO, Information processing—text and office systems—standard general markup language (sgml), iSO-8879 (1986).

[3] W3C, World Wide Web Consortium, http://www.w3.org.

[4] W3C, XML Path Language, http://www.w3.org/TR/xpath (1999).

[5] W3C, eXtensible Stylesheet Language (XSL), http://www.w3.org/Style/XSL/ (2001).

[6] W3C, Simple Object Access Protocol (SOAP), http://www.w3.org/TR/SOAP/ (2001).

[7] SoftwareAG, Tamino XML Server, http://www.softwareag.com/tamino/ (2003).

[8] Oracle Corporation, Oracle9i, http://www.oracle.com/ip/deploy/database/oracle9i/ (2003).

[9] IBM, DB2 universal database 8.1, http://www-3.ibm.com/software/data/db2/udb/ (2003).

[10] Microsoft, SQLServer 2000, http://www.microsoft.com/sql/ (2003).

[11] W3C, XML Schema, http://www.w3.org/XML/Schema (2001).

[12] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, J. Siméon, Querying documents in object databases, Int. J. Dig. Lib. 1 (1) (1997) 5–19.

[13] V. Christophides, S. Abiteboul, S. Cluet, M. Scholl, From structured documents to novel query facilities, SIGMOD Rec. 3 (2) (1994) 313–324.

[14] G. Della Penna, A. Di Marco, B. Intrigila, I. Melatti, A. Pierantonio, Xere: Towards a natural interoperability between xml and er diagrams, in: Proceedings of the 6th Conference on Fundamental Approaches to Software Engineering (FASE 2003), Vol. 2621, LNCS, Springer, 2003, pp. 356–371.

[15] P. Chen, The Entity-Relationship Model: Toward a Unifying View of Data, ACM Transactions on Data Base Systems 1 (1) (1976) 9–36.

[16] D. Garlan, M. Shaw, Software Architecture: Perspectives on an Emerging Discipline, 1996.

[17] W3C, Scalable Vector Graphics (SVG), http://www.w3.org/Graphics/SVG/ (2001).

[18] M. R. Jensen, T. H. M&#248;ller, T. B. Pedersen, Converting xml dtds to uml diagrams for conceptual data integration, Data Knowl. Eng. 44 (3) (2003) 323–346.

[19] R. Conrad, D. Scheffner, J. C. Freytag, Xml conceptual modeling using uml., in: A. H. F. Laender, S. W. Liddle, V. C. Storey (Eds.), ER, Vol. 1920 of Lecture Notes in Computer Science, Springer, 2000, pp. 558–571.

[20] Oracle Corporation, Oracle XMLDB whitepaper, http://otn.oracle.com/tech/xml/xmldb/pdf/XMLDB_Technical_Whitepaper.pdf (2003).

[21] Microsoft, SQLXML and XML mapping technologies, http://msdn.microsoft.com/nhp/Default.asp?contentid=28001300 (2003).

[22] C. Frankston, H. S. Thompson, XML-Data Reduced (XSL), http://www.ltg.ed.ac.uk/ ht/XMLData-Reduced.htm (1998).

[23] Sybase Inc., Sybase database server, http://www.sybase.com/products/databaseservers (2003).

[24] IBM, DB2 XML extender, http://www-3.ibm.com/software/data/db2/extenders/xmlext/ (2003).

[25] G. Navarro, R. Baeza-Yates, Proximal nodes: A model to query document databases by content and structure, ACM Trans. Inf. Syst. 5 (4) (1997) 400–435.

[26] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. Dewitt, J. Naughton, Relational databases for querying xml documents: Limitations and opportunities, in: Proceedings of the 25th International Conference on Very Large Data Bases, 1999, pp. 302–314.

[27] A. Deutsch, M. Fernandez, D. Suciu, Storing semistructured data with STORED, in: Proceedings of the ACM SIGMOD international conference on Management of data, ACM Press, 1999, pp. 431–442.

[28] P. McBrien, A. Poulovassilis, A semantic approach to integrating xml and structured data sources, in: CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag, London, UK, 2001, pp. 330–345.

[29] J. Zhang, Application of OODB and SGML techniques in text database: an electronic dictionary system, ACM SIGMOD Record 4 (1) (1995) 3–8.

[30] D. Florescu, D. Kossmann, Storing and querying xml data using an rdmbs, IEEE Data Eng. Tech. Bull. 2 (3) (1986) 27–34.

[31] M. Yoshikawa, T. Amagasa, A path-based approach to storage and retrieval of XML documents using relational databases, ACM Transactions on Internet Technology 1 (1) (2001) 110–141.

[32] From xml schema to relations: A cost-based approach to xml storage, in: ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02), IEEE Computer Society, Washington, DC, USA, 2002, p. 64.

[33] W3C, XML Query, http://www.w3.org/XML/Query (2001).

[34] M. Fernandez, W.-C. Tan, D. Suciu, Silkroute: trading between relations and xml, Comput. Networks 33 (1-6) (2000) 723–745.

[35] M. Fernandez, A. Morishima, D. Suciu, Efficient evaluation of XML middle–ware queries, in: Proceedings of the ACM SIGMOD international conference on Management of data, ACM Press, 2001, pp. 103–114.