JOHANNES KEPLER UNIVERSITY LINZ
INSTITUTE OF APPLIED COMPUTER SCIENCE
DEPARTMENT OF INFORMATION SYSTEMS
o.Univ.-Prof. Dipl.-Ing. Mag. Dr. Gerti Kappel

Technical Report

X-Ray –
Towards Integrating XML and Relational Database Systems

Gerti Kappel
Elisabeth Kapsammer
Werner Retschitzegger

July 2000

X-Ray -
Towards Integrating XML and Relational Database Systems

Gerti Kappel, Elisabeth Kapsammer, Werner Retschitzegger

Institute of Applied Computer Science, Department of Information Systems (IFS)
University of Linz, Altenbergerstraße 69, A-4040 Linz, Austria
{gk, ek, wr}@ifs.uni-linz.ac.at

Abstract. Relational databases get more and more employed in order to store the content of a web site. At the same time, XML is fast emerging as the dominant standard at the hypertext level of web site management describing pages and links between them. Thus, the integration of XML with relational database systems to enable the storage, retrieval and update of XML documents is of major importance. Data model heterogeneity, however, and schema heterogeneity makes this a challenging task. This paper presents X-Ray, a generic approach for integrating XML with relational database systems. The key idea is that mappings may be defined between XML DTDs and relational schemata while preserving their autonomy. This is made possible by introducing a meta schema and meta knowledge for resolving data model heterogeneity and schema heterogeneity. Since the mapping knowledge is not hard-coded but rather reified within the meta schema, maintainability and changeability is enhanced. The meta schema provides the basis for X-Ray to automatically compose XML documents out of the relational database when requested and decompose them when they have to be stored.

1 Introduction

Web-based information systems no longer aim at purely providing read-only access to their content, which is simply represented in terms of web pages stored in the web server's directory. Nowadays, not least due to new requirements emerging from several application areas such as electronic commerce, the employment of databases to store the content of a web site turns out to be worthwhile [17], [25]. This allows to easily handle both retrieval and update of large amounts of data in a consistent way on a large distributed scale [15]. Besides using databases at the *content level*, the Extensible Markup Language (XML) [33] is fast emerging as the dominant standard for representing the *hypertext level* of a web site, i.e., the logical composition of web pages and the navigation structure [1], [10], [32]. XML is a subset of SGML. As such, XML tags allow to describe the meaning of the content itself. New tags and attribute names can be defined, document structures can be nested to any level of complexity and documents can be associated with a type specification called *document type definition (DTD).*

Because of the increasing importance of XML and database systems (DBS), the integration of them with respect to storage, retrieval, and update is a major need [11], [32]. Regarding the kind of DBS used for the integration, one can distinguish four different approaches [3], [18]. First, *special-purpose DBS* are particularly tailored to store, retrieve, and update XML documents. Examples thereof are research prototypes such as Rufus [30], Lore [20], Strudel [17] and Natix [21] as well as commercial systems such as eXcelon [24] and Tamino [28]. Second, because of the rich data modeling capabilities of *object-oriented DBS*, they are well-suited for storing hypertext documents [5], [33]. Object-oriented DBS and special-purpose DBS, however, are neither in wide-spread use nor mature enough to handle large scale data in an efficient way. *Object-relational DBS* would be also appropriate for mapping to and from XML documents since the nested structure of the object-relational model blends well with XML's nested document model. Similar arguments as above, however, hold against their short-term usage. Thus, the more promising alternative to store XML documents are *relational database systems (RDBS).* Such an integration would provide several advantages such as reusing a mature technology, seamlessly querying data represented in XML documents and relations, and the possibility to make legacy data already stored within an RDBS available for the web.

Concerning the kind of storage within an RDBS, there exist three basic alternatives. The most straightforward approach would be to *store XML documents as a whole* within a *single database attribute*. Another possibility would be *to interpret XML documents as graph structures* and provide a relational schema allowing to store arbitrary graph structures [18], [19], [27], [31]. The third approach is that *the structure of XML documents* in terms of, e.g., a DTD *is mapped to a corresponding relational schema* wherein XML documents are stored according to the mapping [4], [8], [14], [16], [23], [29]. Only the last of these alternatives allows to really exploit the features of RDBS such as querying mechanisms, optimization, concurrency control and the like. Thus, this approach is further investigated in the paper.

Despite of the benefits of the mapping approach, the problem is that when defining the mapping between an XML DTD and a relational schema, one has to cope with *data model heterogeneity* and *schema heterogeneity*. Data model heterogeneity refers to the fact that there are fundamental differences between concepts provided by XML and those provided by RDBS, which have to be considered when defining a certain mapping. These differences concern, e.g., structuring, typing and identification issues, relationships, default declarations, and the order of stored instances (cf. Section 2). Schema heterogeneity in our context means that, even if the DTD and the relational schema to which the DTD should be mapped represent the same part of the universe of discourse, the design of both is likely to be different. This difference can be first of all the result of data model heterogeneity, meaning that one and the same real world entity has to be represented in terms of different concepts. Another reason may be that document design and database design have different goals in mind. For example, to make browsing more effective, documents are very redundant data sources since the same piece of information may occur in several documents and navigated to by several different access paths. In RDBS, on the contrary, redundancy is eliminated by means of normalization techniques to avoid inconsistencies and up-

date problems. A third reason for schema heterogeneity may be that the DTD and the corresponding relational schema have been developed independent of each other without having any integration in mind. Consider for example business to business electronic commerce, where a supplier wants to store the product catalogue of another supplier represented in XML within an already existing relational database. In this scenario, the autonomy of both the DTD and the relational schema should be preserved in that neither of them has to be changed.

Existing approaches deal with these heterogeneity problems in various rather restricted ways. First, to reduce the heterogeneity a priori it is assumed that at least one of the schemata to be mapped to can be adapted to the other one [14]. This, however, contradicts the requirement of autonomy. Second, there is only a certain pre-defined way of mapping provided [29], thereby preventing user-defined mappings which might eventually better resolve a certain heterogeneity with respect to, e.g., space or performance issues. Third, the mapping knowledge is often hard-coded within applications thus making maintenance in case of changes very difficult. With respect to these drawbacks, this paper proposes *X-Ray*, a *generic approach* for integrating XML with RDBS. The key idea is that mappings can be dynamically defined between *DTDs and relational schemata* thus coping with data model heterogeneity and schema heterogeneity. The integration *fully preserves the autonomy of both the DTD and the relational schema,* which in turn ensures the continuity of already existing applications working with the XML documents or the RDBS. This is made possible by introducing a *meta schema* storing information about the DTD, the relational schema and the mapping knowledge itself. The meta schema is responsible for mediation with respect to data model heterogeneity and schema heterogeneity and thus represents the core component of X-Ray. Since the mapping knowledge is not hard-coded but rather reified within the meta schema, maintainability and changeability is enhanced. This meta schema provides the basis for X-Ray to automatically compose XML documents out of the relational database when requested and decompose them when they have to be stored.

XML-DBMS introduced in [4] is closely related to X-Ray. Whereas in X-Ray the mapping knowledge may be specified in terms of tuples of the predefined meta schema, XML-DBMS provides a mapping language DTD. A specific user-defined XML document obeying this mapping language DTD, represents the mapping knowledge for yet another DTD and a relational schema. Based on our previous experience, however, using also a meta schema approach for mapping between objects and relations [22], working with a meta schema is quite intuitive and, thus, also suggested for X-Ray.

The remainder of the paper is organized as follows. Section 2 presents an in-depth analysis of data model heterogeneity between XML and RDBS. Section 3 introduces different mapping possibilities between XML and RDBS. Section 4 defines a set of reasonable mappings to mediate between the different structuring mechanisms supported by XML and RDBS. The design of the meta schema is discussed in Section 5. Finally, Section 6 concludes the paper with a summary and gives an outlook to future work.

2 Comparison of Concepts

This section is dedicated to an in-depth investigation of the similarities and differences between XML concepts and RDB concepts. It provides the basis for analyzing different mapping possibilities as done in Section 3 and Section 4 and represents the prerequisite for developing a meta schema in order to bridge the heterogeneous concepts (cf. Section 5). Note, that whenever necessary, it is distinguished between concepts as defined by the original relational model and their realization by the SQL standard [2].

2.1 Levels of Abstraction

For discussing and comparing the basic concepts of XML from a database point of view, it is important to keep in mind that they belong to different levels of abstraction as illustrated in Fig. 1. These levels comprise the *data model level*, the *schema level*, and the *instance level.*

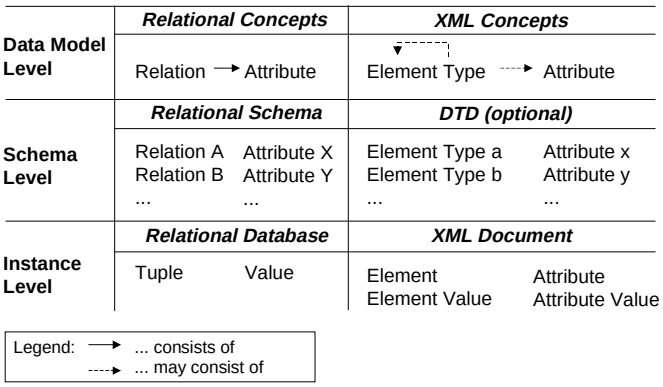

Fig. 1. Concepts at Different Levels of Abstraction

Regarding the data model concepts provided by RDBS and XML, there are fundamental differences leading to data model heterogeneity, which aggravates the integration of both paradigms. The heterogeneity of the data models is mainly due to the different purposes RDBS and XML have been developed for. The aim of RDBS is to store large amounts of data enabling efficient access and ensuring their consistency [2]. In contrast, XML is intended to serve as a format for structuring and exchanging hypertext documents [1], [32]. Data model heterogeneity is discussed in the following sub sections focussing on *typing mechanisms*, *null values and default values*, *identification*, *relationships*, and *order of instances*. Since from a database perspective, many concepts supported by DTDs are inadequate for schema definition, e.g., the typing mechanisms, XML is often referred to as *a data format*, only, having no appropriate data model. Consequently, there are already several efforts to replace DTDs by means of richer XML schema definition languages which are in contrast to DTDs expressed

by means of XML itself [7]. Extensions include a richer set of primitive data types as well as mechanisms to enable inheritance. However, since there is no standard up to now, the rest of the paper builds on DTDs, only.

Analogous to heterogeneity at the data model level, there may also be heterogeneity at the schema level. This is very likely since the *schema*, i.e., type, of an XML document is allowed to be irregular, implicit, partial, incomplete, not always known ahead of time, and may change frequently and without notice, which demonstrates the close resemblance to semi-structured data [34]. In case that XML documents are based on a DTD, applications are able to validate their structure with respect to the DTD by means of an XML parser [14]. The specification of the DTD can be included directly within an XML document or stored within a separate file. Since DTDs are optional, it is not clear at this time if more XML documents will be governed by DTDs or if more documents will exist without such a type description [29]. These are fundamental differences to RDBS where the existence of an a priori schema, which is stored directly within the database, is mandatory, and the validity of tuples with respect to this schema is checked by the system before inserting them into the database.

At the instance level, we consider a certain XML document and a certain relational database, respectively. XML documents are *self-describing*, meaning that parts of the schema definition in terms of *tags* are replicated within each XML document, no matter if the schema is defined explicitly in form of a DTD or not. This is in contrast to RDBS where the schema exists only once for the whole database. Storing the schema along with the data provides flexibility with respect to both integrating heterogeneous sources and changes to the structure. However, the replicated schema information implies space cost for storage, time cost for retrieval, and the danger of inconsistencies in case of updates [14].

2.2 Structuring and Typing Mechanisms

The basic mechanisms used to specify the structure of XML documents and relational schemata are *element types* and *attributes* for XML as well as *relations* and *attributes* for RDBS, respectively (cf. Fig. 1). The name of an XML element type has to be unique throughout the DTD[1], the name of a relation is required to be unique within the whole relational schema. The name of an XML attribute has to be unique within its element type, an RDBS attribute's name has to be unique within its relation. In contrast to these similarities, there are major differences with respect to typing mechanisms.

In RDBS, *atomic domains* can be specified for attributes only, whereas XML allows to specify atomic domains for both attributes and element types. In contrast to RDBS, supporting a large variety of predefined atomic domains, the set of predefined atomic domains in XML is rather restricted. Concerning XML attributes, the predefined domains comprise a *string type* (`CDATA`), an *enumeration type*, and some special types including, e.g., `ID` and `IDREF(S)` (cf. Section 2.4). Concerning XML element

[1] By means of so called *namespaces* [6], XML allows element types within a DTD having the same name. Namespaces, however, are not further considered in this paper.

types, there is only one possible atomic domain namely `#PCDATA`. This domain is different from `CDATA` in that if values contain tags, these tags are interpreted by an XML parser. Element types that contain an atomic domain, only, are furtheron called *atomic element types* (cf. Table 1).

Table 1. Kinds of Element Types

Kind of Element Type (ET)	Atomic Domain	Component ET	Attributes
Atomic ET	✔	✘	~
Composite ET with Element Content	✘	✔	~
Composite ET with Mixed Content	✔	✔	~
Empty ET	✘	✘	~

Legend: ✔ ... contains
✘ ... does not contain
~ ... no influence

Besides atomic domains, element types are allowed to contain other element types furtheron called *component element types* used to build arbitrarily deep part-of hierarchies by means of nesting. For each XML document it is required that all component element types are rooted in a single element type. An element type containing other element types is furtheron called *composite element type*. This is in contrast to RDBS, where part-of hierarchies cannot be realized by means of nesting since relations consist of atomic-valued attributes, only. However, part-of hierarchies can be expressed in RDBS by means of foreign key constraints (cf. Section 2.4). Since composite element types may have an atomic domain in addition to component element types, they are further distinguished into *composite element types with mixed content* and *composite element types with element content* (cf. Table 1). Concerning the latter, it has to be specified whether component element types occur in a *sequence*, or as *choice* meaning that they are mutual exclusive. Element types that neither contain component element types nor have an atomic domain are called *empty element types*. An element type can be also declared to have `ANY` content, meaning that there is no restriction concerning the kind of the element type. Finally, each element type no matter if it is an atomic, composite, or empty element type may contain XML attributes.

Table 1 summarizes the different kinds of element types by denoting their characteristics (concerning examples it is referred to Sections 3 and 4). Note, that the XML standard specification [6] does not provide any terminology for such a distinction.

Concerning the instance level (cf. Fig. 1), XML documents contain *elements* each of them marked by a *start tag* and an *end tag* in terms of the name of a specific element type. The element may contain component elements expressed by nested tags as well as attributes. Both elements and attributes are allowed to contain *values*, therefore we distinguish between *element values* and *attribute values*. Attribute names and their values are placed within the start tag, whereas values of elements occur between start tag and end tag. Consequently, schema information of the DTD is replicated within XML documents in that each element and each attribute value is annotated with the corresponding element type name and attribute name, respectively. The in-

stance level of an RDBS is quite simpler, since values exclusively belong to attributes, which are in turn composed to *tuples*.

2.3 Null Values and Default Values

Similar to RDBS, XML allows to express *null values* as well as *default values*. Whereas in RDBS, however, the concept of null values is defined for attributes only, XML supports null values for both attributes and elements. Default values may be applied to XML attributes, only.

Concerning *XML attributes*, the so-called *default declaration* within a DTD requires to specify for each attribute one of the following constraints:

- `#REQUIRED`, meaning that a value is required in the sense of `NOT NULL` of RDBS
- `#IMPLIED`, denoting the optional nature of an attribute value, expressed by the omission of `NOT NULL` in RDBS. Note, that in case there is no value provided for such an XML attribute at the instance level, the attribute name is omitted within the XML document, too.
- `#FIXED <ConstValue>`, defining a constant value which is not possible in RDBS
- `<DefaultValue>`, specifying a default value analogous to the `DEFAULT` clause in RDBS

Concerning an *element*, whether it may be omitted or not is specified within the DTD by means of *cardinality* constraints. The cardinality specifies how often the element of a certain element type occurs as component element of its composite element. Since element types may be components of more than one composite element type, each of its occurrences as component element type can exhibit another cardinality. The cardinality symbols are '?' (null or 1), '*' (null or more), '+' (1 or more) and no symbol (exactly 1). It is emphasized that there is a semantic difference between start tag being directly followed by an end tag and start and end tag being omitted at all from the XML document. The former matches to one of three different specifications within the DTD:

- An element is specified as an *empty element type*.
- An element is specified as an *atomic element type*, whose *value is an empty string*.
- An element is specified as a *composite element type*, but within the XML document, *no component elements* exist.

In contrast to that, the omission of tags indicates a null value in that a corresponding element does not exist.

2.4 Identification

In RDBS, the unique identification of tuples is done by means of a *primary key*, which may be composed of one or more attributes of the corresponding relation (cf.

Table 2). In XML, only a single attribute of an element type can be designated as identifying attribute by means of the special *attribute type* `ID` which may in turn contain a string value (cf. Fig. 2).

```
DTD:           <!ELEMENT owner (accommodation+)>
               <!ATTLIST owner myIdentifier ID #REQUIRED>

XML document:  <owner myIdentifier="4711"> .... </owner>
```

Fig. 2. Exemplary Identification in XML

The *scope of identification* in RDBS is a single relation, i.e., the value of the primary key uniquely identifies each tuple within a relation. In XML, the scope of identification is broader in the sense that the value of an `ID` attribute is unique within the whole XML document. This allows the unique identification of an element not only with respect to other elements of the same element type but rather across all elements of any element type.

Table 2. Comparison of Concepts: Identification

	RDBS	**XML**
Concept	Primary key	`ID` attribute type
Composite Key	*Yes* - one or more attributes of a relation	*No* - single attribute of an element type
Scope of Identification	Unique identification of tuple *within relation*	Unique identification of element *within document*
Optional Key	Yes	Yes
Equality & Identity	No distinction	No distinction

In XML, element types are not required to contain an `ID` attribute. This is similar to RDBS products, where relations need not contain a primary key. Note, this is in contrast to the theory of the relational model, where primary keys are mandatory for each relation [2]. Even in case that an element type has an attribute of type `ID`, its usage may be optional by defining it as `#IMPLIED`. Since the identification of both tuples in RDBS and elements in XML is *value-based*, it is not possible to distinguish between equality and identity as it is possible in the object-oriented data model [9].

2.5 Relationships

In RDBS, relationships can be expressed *between relations* by means of *foreign keys*, i.e., arbitrary attributes that refer to the primary key of the same relation or of another relation. The number of tuples which may participate in a relationship can be constrained by defining the foreign key as `NOT NULL` and/or `UNIQUE`. With this, different cardinalities can be supported as illustrated in Table 3. XML allows two alternative ways for specifying relationships between element types comprising `IDREF(S)` attributes and component element types (cf. Section 2.2). Attributes of type `IDREF(S)` rep-

resent some kind of foreign key referencing attributes of type `ID`. The distinction between `IDREF` attributes and `IDREFS` attributes concerns their cardinality, in that the former are single-valued and the latter are multi-valued.

Table 3. Comparison of Concepts: Relationships

	RDBS	**XML**	
Concept	Foreign key attribute	`IDREF(S)` attribute	Component ET
Participants	Relations (tuples)	Element types (elements)	
Typing of Participants	Tuples of a certain relation	It is not possible to constrain the type of the participating elements	The participating elements are restricted by the composition structure
Cardinality	(0..1):*, 1:*, (0..1):(0..1), 1:(0..1)	(0..1):*, 1:*, *:*, (1..*):*	1:1, 1:*, 1:(1..*), 1:(0..1)

Legend: 0..1 ... zero or one
1..* ... one or more
* ..... zero or more

In contrast to RDBS, where the participating tuples are constrained to the participating relation, the participating elements cannot be constrained to be of a certain element type.

2.6 Order

In contrast to relations and tuples in RDBS, the element types and elements of an XML document adhere to both an *explicit* and *implicit order*. The order of element types can be explicitly defined within a DTD by using the sequence operator ‘,’. The example shown in Fig. 3 specifies that an element of type village comprises the following three component element types in the specified order, i.e., name has to occur first, then country, and then accommodation.

```
DTD:          <!ELEMENT village (name, country, accommodation*)>

XML document: <village>
                        <name>Innsbruck</name>
                        <country>Tyrol</country>
                        <accommodation>Hotel Post</accommodation>
                        <accommodation>Hotel Admiral</accommodation>
                        <accommodation>Hotel Anker</accommodation>
              </village>
```

Fig. 3. Explicit and Implicit Order

At the instance level, the order of concrete elements is defined implicitly by the position of elements within the XML document (cf. also Fig. 3). Note, that this implicit order may not contradict the explicit order defined by the corresponding DTD. In our example the order of the particular `accommodation` elements is given at the instance level by occurring at a certain position within an element of type `village`. It is important to be aware, that elements occurring as component elements of different composite elements do not always have to exhibit the same order.

```
DTD:           <!ELEMENT village (accommodation|restaurant)*>

XML document:  <village>
                        <restaurant>Einkehr</restaurant>
                        <accommodation>Hotel Post</accommodation>
                        <accommodation>Hotel Admiral</accommodation>
                        <restaurant>Verdi Diele</restaurant>
                        <accommodation>Hotel Anker</accommodation>
               </village>
```

Fig. 4. Implicit Order Between Elements of Different Element Types

For example, elements of type `accommodation` being component elements of type `village` may show a different order than as component elements of type `owner`. An implicit order not only concerns elements of the same element type but also elements of different element types, as is depicted in Fig. 4.

3 Basic Kinds of Mappings Between XML and RDBS

After having analyzed differences between XML concepts and RDBS concepts, let's consider the possibilities for mapping a DTD to a relational schema. A straightforward way would be to map each element type to a relation and each XML attribute to an attribute of the respective relation (cf. Fig. 5). Due to data model heterogeneity and schema heterogeneity, however, such a one to one mapping is neither always possible nor desirable. For example, in the presence of deep element nesting directly mapping elements to tuples of different relations would lead to excessive fragmentation of the document over various relations, thus decreasing performance. This section proposes some basic mapping possibilities representing a prerequisite both for determining which kind of mapping is reasonable in a certain situation (cf. Section 4) and for designing the meta schema (cf. Section 5).

When considering the structuring mechanisms of XML and RDBS as discussed in Section 2.2, three basic kinds of mappings at the data model level may be distinguished (cf. Fig. Fig. 6):

(1) ET_R. An element type(ET) is mapped to a relation (R), furtheron called *base relation*. Note, that several element types can be mapped to one base relation. An example for an ET_R mapping is the element type `accommodation` in Fig. 6.

(2) ET_A. An element type is mapped to a relational attribute (A), whereby the relation of the attribute represents the base relation of the element type. Note, that several element types can be mapped to the attributes of one base relation.

(3) A_A. An XML attribute is mapped to a relational attribute whose relation represents the base relation of the XML attribute. Again, several XML attributes can be mapped to the attributes of one base relation. The XML attributes `name`, `street`, and `village` in Fig. 6 give an example.

It has to be emphasized that both element types and attributes can be mapped to a single base relation and a single attribute, only. Another point is that ET_A and A_A mappings determine also the instance level, in that database values are mapped to

XML values. Thus, it makes sense that ET_R mappings occur together with ET_A and A_A mappings.

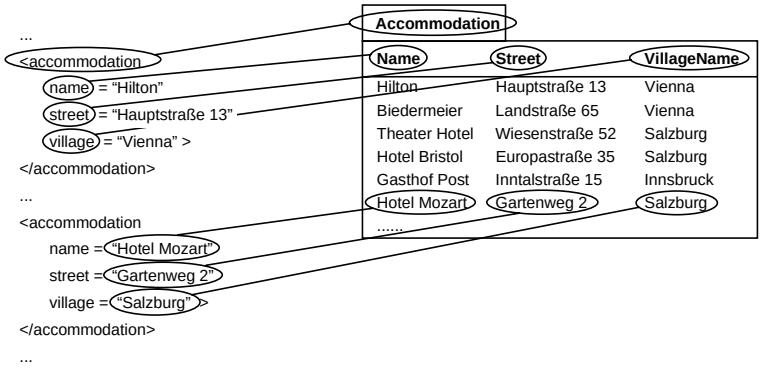

Fig. 5. Straightforward Mapping of XML Concepts to Relational Concepts

Furthermore, it is not mandatory that all element types and attributes of a DTD as well as all relations and attributes of a relational schema have a mapping. An example at the relational side could be a foreign key that serves for establishing a relationship but might not be relevant within the XML document and therefore requires no mapping. An example at the XML side would be an empty element type that occurs exactly once at a certain position within the XML document and does not require any mapping, neither.

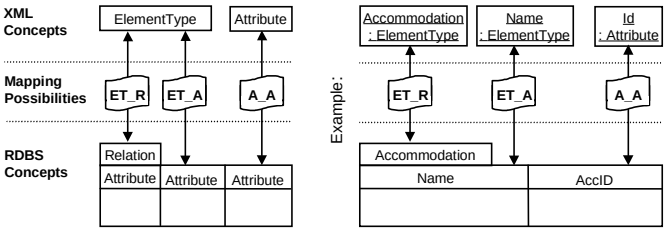

Fig. 6. Basic Kinds of Mappings

The examples demonstrate that the omission of mappings is imaginable not only in case that both DTD and relational schema have been developed independently from each other, but also if one has been derived from the other one. However, in case that the cardinality of a relation or an element type respectively, or the default declaration of an attribute requires the existence of a corresponding instance, a proper mapping is mandatory.

The three basic kinds of mappings introduced above can be further refined with respect to the determination of an element type's base relation. First, if an element type should be mapped, one has to consider the first of its direct or indirect composite element types that is mapped to a relation or an attribute, thus having a base relation. This base relation constitutes the *parent base relation* of the XML element type which should be mapped and is a candidate for being its base relation, too. If none of its composite element types is mapped, an arbitrary relation can be chosen as base

relation. Concerning the example in Fig. 7 (cf. also the more comprehensive example given in Fig. 8), the element types `address`, `street`, and `country` all have the same parent base relation, namely `Accommodation`, which represents the base relation of the composite element type `accommodation`. Note, that aiming at an intuitive presentation, Fig. 7 depicts mappings between XML element types and relations in terms of a UML class diagram. To be able to distinguish between element types and relations, they are depicted as instances of the corresponding 'meta class' `ElementType` and `Relation`, respectively.

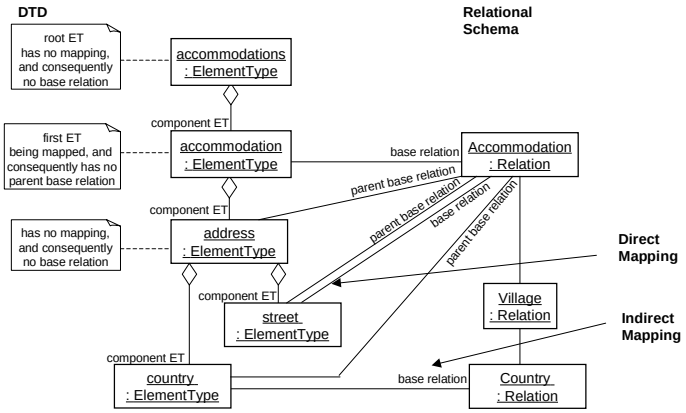

Fig. 7. Exemplary Mappings

Second, if an XML attribute should be mapped, its element type has to be considered first. If the attribute's element type is not mapped, its direct and indirect composite element types have to be considered as done for element types discussed above. Again, the relation which the first of these element types is mapped to represents the parent base relation of the XML attribute, thus being a candidate for being its base relation, too. The parent base relation constitutes also the base relation, if the XML element type or the attribute, respectively, can be mapped to the relation or one of its attributes, which is furtheron called *direct mapping*. For an example, confer to the element type `street` in Fig. 7, which is directly mapped to an attribute of its parent base relation `Accommodation`. Otherwise, a proper base relation may be one of those relations, reachable by the parent base relation via foreign key relationships, which is furtheron called *indirect mapping*. For an example, consider the element type `country`, which is indirectly mapped to an attribute of relation `Country` reachable by its parent base relation `Accommodation`. Indirect mapping is reasonable in case that the relational attribute, which should be the mapping target, is factored out from the parent base relation, e.g., due to normalization reasons or because of vertical partitioning. Note, that element type `address` is used to group address data and thus has no relational counterpart and no base relation at all.

Both direct and indirect mapping is applicable to the three basic mapping possibilities introduced above thus resulting in $ET\_R_{direct/indirect}$, $ET\_A_{direct/indirect}$, and $A\_A_{direct/indirect}$ mappings. Furthermore, the possibility of a direct mapping always implies the possibility of an indirect mapping due to vertical partitioning.

4 Determining Reasonable Mappings Between XML and RDBS

After introducing the basic kinds of mappings, this section discusses reasonable mappings. The determining factors can be categorized into *characteristics of the XML element type* (cf. Section 4.1) and *characteristics of the XML attribute* (cf. Section 4.2). In order to illustrate the subsequent investigations in Fig. 8, we provide a comprehensive running example building on the ones given in the previous section.

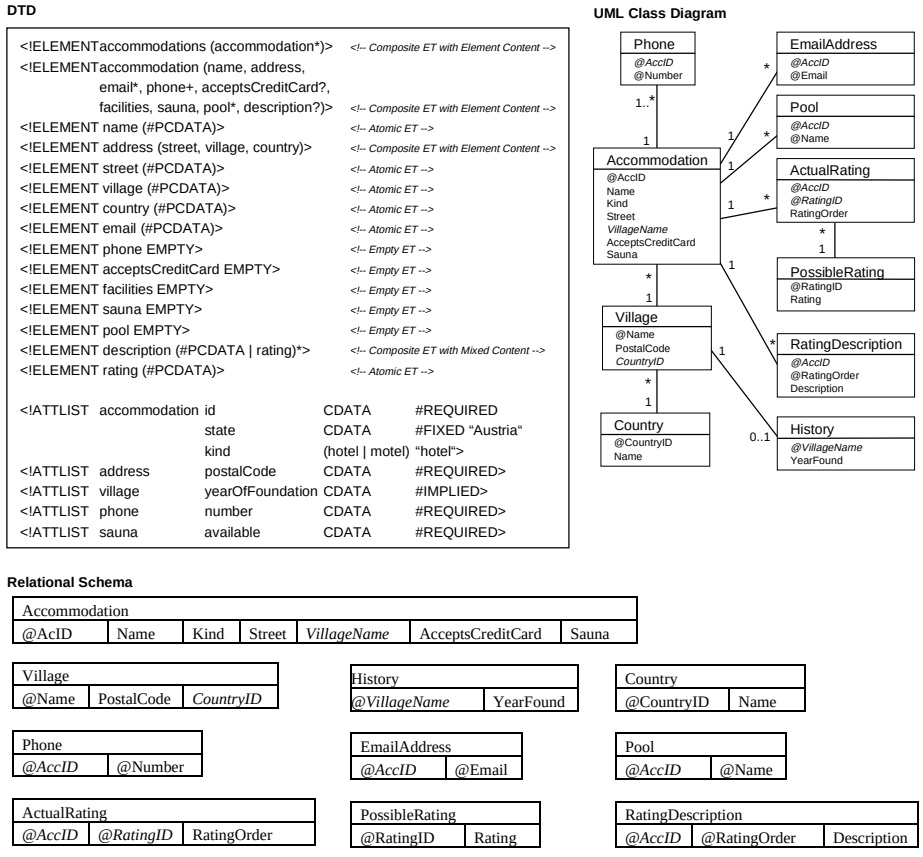

Fig. 8. Exemplary DTD, Relational Schema, and UML Class Diagram

The example is intended to show as many mapping possibilities as possible. Fig. 8 depicts the running example in terms of a DTD and in terms of a relational schema. The latter is depicted with a table structure and as UML class diagram better visualizing relationships. Concerning the relational schema, primary keys are prefixed with '@' and foreign keys are depicted using italic type. Even this small example shows that data model heterogeneity and schema heterogeneity prevent a simple one to one mapping. Regarding the DTD illustrated in Fig. 8, there is a single root element type `accommodations` having no relational counterpart. Its component element type `accom-`

modation contains various element types, which have either relational attributes or relations as counterparts. The different cardinalities specified for each of these element types correspond to those defined at the relational side. Regarding the composite element type address and its atomic component element types street, village, and country it can be seen that the relational schema does not contain a relation Address with attributes Street, Village, and Country. Even more, there does not exist any counterpart for address in the relational schema and its component element types correspond to attributes located in three different relations, connected by '*:1' relationships, namely attribute Street of relation Accommodation, attribute Name of relation Village, and attribute Name of relation Country. Having three relations instead of one is the consequence of the normalization process. accommodation as well as some of its component element types contain attributes. One of these attributes, namely state, has the fixed value 'Austria' and therefore lacks a relational pendant. The attribute kind is restricted to an enumeration of two values. The composite element type description has mixed content, comprising the atomic element type rating meaning that elements of this type may occur several times mixed with atomic data in any order within an XML document. Note, that the attributes RatingOrder of the two classes ActualRating and RatingDescription are not mapped to any XML concept. They express an absolute order over both rating descriptions and actual ratings with respect to a certain accommodation. This is not necessary at the XML side, since the order is implicitly defined by the position of the elements within the XML document.

4.1 Element Type Characteristics

As already mentioned, choosing a certain mapping is based on characteristics of the element type to be mapped. As illustrated in Fig. 9, these decisive characteristics can be categorized into three orthogonal dimensions comprising the *kind of element type*, if it *contains attributes*, and its *cardinality*. Note, that if the element type has been declared to have ANY content (cf. Section 2.2), a reasonable mapping cannot be determined in advance. Depending on the combination of these characteristics, certain reasonable mappings can be determined as shown in Table 4. In the following, these mappings are discussed by means of the running example.

First, we consider *composite element types with element content.* Mapping this kind of element type is neither influenced by cardinality nor whether it contains any attributes. Since there are no values associated with elements of this type, the only reasonable mapping possibility is ET_R. Depending on whether the element type can be mapped to its parent base relation or not, ET_R_{direct} or $ET\_R_{indirect}$ mapping can be used. In fact, the lack of any mapping would not result in a loss of information, since elements of this type contain no values which could be stored in the database.

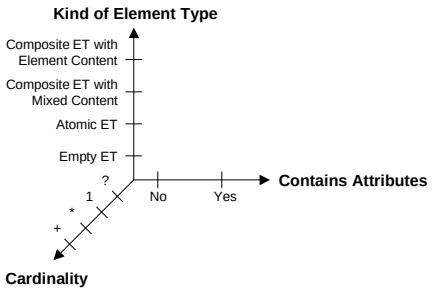

Fig. 9. Orthogonal Dimensions Characterizing XML Element Types

Concerning our running example, whereas the root element type `accommodations` does not require any mapping, the element type `accommodation` is mapped to the relation `Accommodation` (ET_R mapping). Since `accommodation` does not have a parent base relation, we do not distinguish between a direct and an indirect mapping in this case.

Table 4. Reasonable Mappings of XML Element Types

Kind of Element Type	Contains Attributes	Cardinality	Reasonable Mapping
Composite ET with element content	No influence	No influence	ET_$R_{direct/indirect}$; No mapping
Atomic ET	No influence	?, 1	ET_$A_{direct/indirect}$
Atomic ET	No influence	+, *	ET_$A_{indirect}$
Empty ET	No	1	No mapping
Empty ET	Yes	1	ET_$R_{direct/indirect}$; No mapping
Empty ET	No influence	?	ET_A_{direct}
Empty ET	No influence	*, +	ET_$A_{indirect}$
Composite ET with mixed content	No influence	No influence	ET_$A_{indirect}$

Next, let us consider the mapping of an *atomic element type*. The reasonable mappings of such element types depend on the cardinality, only, and are not influenced by the existence of XML attributes. Since atomic element types contain values they always require a mapping to relational attributes, i.e., an ET_A mapping. In case of cardinality '`?`' and '`1`', an ET_A_{direct} mapping is possible, since no more than one element may occur. However, also an ET_$A_{indirect}$ mapping may be necessary, when the relational attribute which the atomic element type should be mapped to is not part of the parent base relation. In case of cardinality '`*`' and '`+`', ET_$A_{indirect}$ mapping is required due to normalization.

Concerning our running example, the most simple case is represented by element type `name` which has cardinality '`1`' and is mapped to attribute `Name` of base relation `Accommodation` representing an ET_A_{direct} mapping. `Accommodation` is mapped to element type `accommodation`, the direct composite element type of element type `name`, i.e., the base relation and the parent base relation are the same. This kind of mapping also applies to element type `street`. In this case the direct composite element type `address` has no mapping and the indirect composite element type `accommodation` is

mapped to the relation that contains the relational counterpart `Street`. The element types `village` and `country` require ET_$A_{indirect}$ mappings, since their relational counterparts are stored in base relations different to the parent base relation `Accommodation` due to normalization reasons. The relational counterparts are attribute `Name` of base relation `Village` and attribute `Name` of base relation `Country`, respectively. This kind of mapping is possible, since `Accommodation` and `Village`, as well as `Village` and `Country` are directly connected via foreign key relationships. Element type `email` has cardinality '`*`' requiring an ET_$A_{indirect}$ mapping and therefore is mapped to attribute `Email` of relation `EmailAddress`. The same holds true for element type `rating` with the difference that the parent base relation `Accommodation` and the base relation `PossibleRating` containing an attribute `Rating` are indirectly connected via the relation `ActualRating`.

Regarding *empty element types* with a cardinality '`1`', no matter if there are attributes or not, no mapping is required since a corresponding element occurs exactly once without carrying any value. However, if there were attributes, it would make sense to employ a direct or indirect ET_R mapping since the base relation could serve as the base relation for the attributes. In case of any other cardinality, the existence of attributes does not influence the reasonable mappings. An ET_A mapping is required in any case. It depends on the particular cardinality whether a direct or indirect mapping is reasonable.

Referring to our example, the empty element types `facilities` without attributes and `sauna` including a single attribute represent the most simple case both having a cardinality of one thus requiring no mapping. The attribute `available` of element type `sauna` is mapped to the relational attribute `Sauna` of the parent base relation of the element type `sauna`, namely `Accommodation`. The optional empty element type `acceptsCreditCard` contains no attributes and is mapped directly to the relational attribute `AcceptsCreditCard` of its parent base relation `Accommodation`. Finally, the empty element types `phone` and `pool` having a cardinality of '`+`' and '`*`', respectively, are mapped via ET_ $A_{indirect}$ to the relational attribute `Number` of the relation `Phone` and the relational attribute `Name` of the relation `Pool`, respectively.

Considering *composite element types with mixed content*, neither the existence of attributes nor the cardinality have any influence on the reasonable mappings. Since at the instance level, several values may occur within a single element, an ET_$A_{indirect}$ mapping is required. Our example contains one composite element type with mixed content, namely `description`, which is mapped to the attribute `Description` of the relation `RatingDescription`. The attributes `RatingOrder` of the two relations `ActualRating` and `RatingDescription` are, as already mentioned, not mapped to any XML concept, since they express an absolute order over both rating descriptions and actual ratings with respect to a certain accommodation.

4.2 XML Attribute Characteristics

The mapping of XML attributes depends on two orthogonal dimensions comprising the *type of the XML attribute* and the *default declaration* (cf. Fig. 10; for the sake of

readability and space restrictions, we do not consider all possible types of XML attributes but rather the more important ones). Considering the different combinations of these two dimensions three reasonable mapping alternatives may be identified as shown in Table 5.

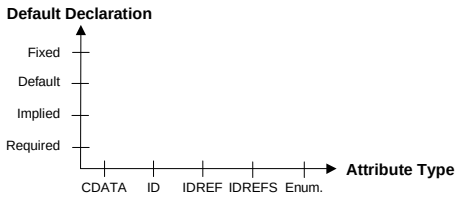

Fig. 10. Orthogonal Dimensions Characterizing XML Attributes

For XML attributes with default declaration being `#FIXED`, no mapping is necessary independent of the type of the XML attribute. In our example, the XML attribute `state` of the element type `accommodation` has the constant value `Austria`. Regarding XML attributes which are not specified to be `#FIXED`, it has to be distinguished whether they are single-valued like `CDATA` or multi-valued defined by `IDREFS`. Single-valued attributes can be directly mapped to relational attributes (A_A_{direct}) or may require indirect mapping due to normalization reasons ($A\_A_{indirect}$), whereas multi-valued attributes may be mapped indirectly ($A\_A_{indirect}$), only. Considering `ID` and `IDREF(S)`, it seems conceivable to map them to primary key attributes and foreign key attributes, respectively, of the relational schema. Due to data model heterogeneity, however, this is not always feasible, since there are differences concerning scope and composite keys (cf. Section 2).

Table 5. Reasonable Mappings of XML Attributes

Attribute Type	Default Declaration	Reasonable Mapping
No influence	#FIXED	No mapping
CDATA, ID, IDREF, enumeration	#REQUIRED, #IMPLIED, Default Value	$A\_A_{direct/indirect}$
IDREFS	#REQUIRED, #IMPLIED, Default Value	$A\_A_{indirect}$

In our example, directly mapped single-valued attributes comprise `id` and `kind` of element type `accommodation`, `number` of element type `phone`, and `available` of element type `sauna`. Single-valued attributes which have to be mapped indirectly are `postalCode` of element type `address`, and `yearOfFoundation` of element type `village`. Multi-valued attributes are not part of our example. It has to be emphasized that with one exception the reasonable mappings of an attribute are independent of the kind of mapping of its element type. In case that the element type of the attribute is not mapped and any of its composite element types that is not mapped depicts a cardinality of '`*`', the attribute can be mapped via $A\_A_{indirect}$, only.

5 The X-Ray Meta Schema

The insights gained in the previous sections concerning data model heterogeneity and mapping possibilities between XML and relational schemata provide the basis for the design of the meta schema of X-Ray. The meta schema is the key mechanism for the genericity of X-Ray allowing to map DTDs and relational schemata. It mediates between heterogeneous concepts and provides the basis for X-Ray to automatically compose XML documents out of the relational database when requested and decompose them when they have to be stored. The mapping knowledge is not hard-coded within an application but rather reified and centrally stored within the meta schema, thus enhancing maintainability and changeability.

5.1 Basic Components of the Meta Schema

The meta schema consists of three components describing the relevant meta knowledge (cf. Fig. 11). The `DBSchema` component is responsible for storing information about relational schemata that shall be mapped to DTDs to make their data available to XML documents or that shall be used to store XML documents. Analogously, the `XMLDTD` component stores schema information about XML documents as specified by means of DTDs. Finally, the `XMLDBSchemaMapping` component stores the mapping knowledge between `DBSchema` and `XMLDTD`. The goal of `XMLDBSchemaMapping` is to bridge both data model heterogeneity and schema heterogeneity in order to support a lossless mapping. This means that if an XML document is stored within the database, it should be possible to reconstruct it by retrieving the corresponding data out of the database and vice versa. It has to be emphasized that although the meta schema is designed on the basis of the concepts provided by DTDs, X-Ray does not require the existence of an explicit DTD. However, there must be at least a common implicit structure of the XML documents, which can be used by an administrator as input for `XMLDTD` and `XMLDBSchemaMapping`.

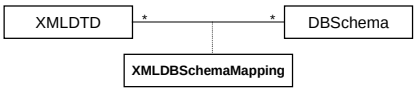

Fig. 11. Components of the X-Ray Meta Schema

In X-Ray, a database schema is not limited to be mapped to a single DTD but may be mapped to several DTDs and vice versa. This is reasonable since, due to presentation requirements, it may be necessary to represent a particular piece of information by several XML documents being based on different DTDs. Likewise, if we assume several relational schemata storing data of the same domain it may be required to represent these data by XML documents based on the same DTD. Concerning the storage of the meta knowledge itself, X-Ray comprises both a relational representation of the meta schema stored within the relational database and an object-oriented representation for main memory mapping. The latter is being initialized with the content of the relational meta schema at the beginning of an X-Ray session, herewith

allowing an efficient composition and decomposition of XML documents at runtime. The object-oriented representation in terms of UML class diagrams is also used throughout this section to concisely and precisely depict the various meta schema components.

5.2 Database Schema Component

Concerning the database schema component, it has to be emphasized that it is not necessary to store meta knowledge about the complete relational schema, but only about those relations and attributes being relevant for the mapping to the DTD. However, not only base relations and their attributes are relevant, but also non-base relations which are the connecting relations between two base relations.

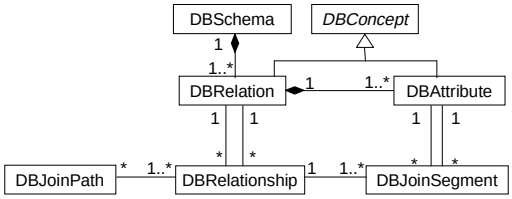

Fig. 12. Meta Schema of the Relational Schema

As illustrated in Fig. 12 `DBSchema` contains at least one `DBRelation`, which consists of at least one `DBAttribute`. `DBAttribute` stores among others its *atomic domain* and whether it represents a *primary key attribute*. `DBRelation` and `DBAttribute` are generalized to `DBConcept`. Relationships (`DBRelationship`) connect two relations and specify one or more *join segments* (`DBJoinSegment`) comprising the join attributes, i.e., primary key and foreign key attributes of two relations that realize the relationship. The relationship comprises more than one join segment in case that the primary key is composed of two or more attributes. In case that parts of an XML document are stored within different relations, information about the proper join paths (`DBJoinPath`) is necessary. A `DBJoinPath` consists of one or more relationships. It comprises more than one relationship if more than two relations have to be joined for composing or decomposing a particular part of an XML document.

5.3 XML DTD Component

Similar to the database schema component, it is not necessary to store meta knowledge about the complete DTD, but only about those parts being relevant for the mapping to the relational schema. The meta knowledge specifies that a DTD (`XMLDTD`, cf. Fig. 13) has a certain element type (`XMLElemType`) that serves as root. For element types with attributes, `XMLAttribute` stores information about their *atomic domains* and their *default declaration*. Similar to the database schema component, `XMLElemType` and `XMLAttribute` are generalized to `XMLConcept`. For enumeration attributes

the possible values are stored within `XMLAttValEnum`. According to the distinction made in Section 2, `XMLElemType` is specialized into `XMLAtomicET`, `XMLEmptyET`, and `XMLCompositeET`. The latter is further specialized into `XMLCompositeETMixedContent` and `XMLCompositeETElemContent`.

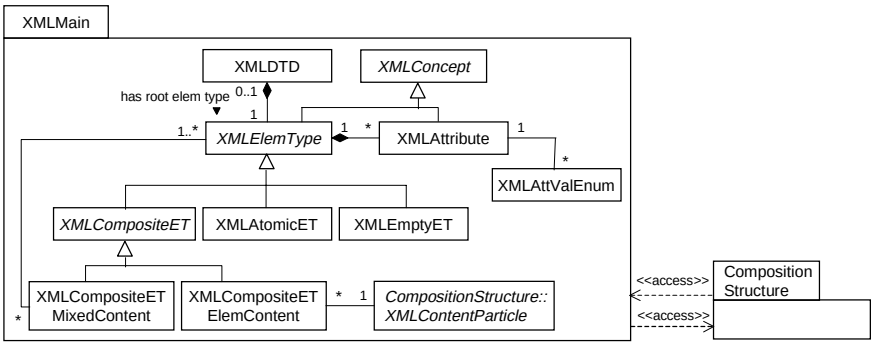

Fig. 13. Meta Schema of the DTD

The nesting structure of an `XMLCompositeETElemContent` is described by the package `CompositionStructure` (cf. Fig. 14). For an `XMLCompositeETMixedContent` the nesting structure needs not to be represented in the meta schema, since, as already mentioned, component element types are allowed to occur in a choice with cardinality '`*`', only.

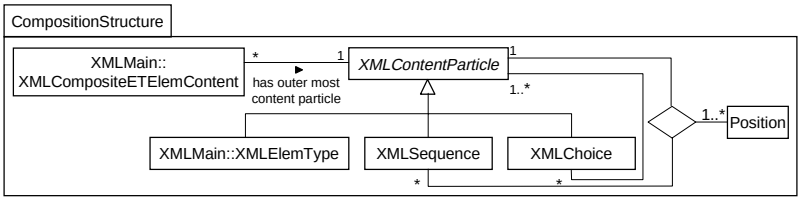

Fig. 14. Meta Schema of the XML Composition Structure

For component element types occurring in an `XMLSequence` or in an `XMLChoice`, the *cardinality* of the element type and in case of a sequence its *position* have to be stored. Furthermore, arbitrary combinations of sequences and choices can be described.

5.4 Mapping Knowledge

The mapping knowledge is expressed by various associations between the object classes of the XML DTD component and the database schema component. Fig. 15 illustrates these mapping relationships denoting them with bold lines. For representation convenience, only those object classes are shown which are part of a mapping relationship. In order to meet the requirement that the meta schema is able to store

mappings between different DTDs (XMLDTD) and different database schemata (DBSchema), the mapping between the class XMLConcept and the class DBConcept takes part in a ternary relationship with the association class XMLDBSchemaMapping. As discussed in Section 4, deciding on the exact kind of element type is a prerequisite for deciding a reasonable mapping to a database concept. Consequently, the leaf classes of the XMLElemType hierarchy are mapped to DBAttribute with two exceptions. The class XMLCompositeETElemContent is mapped to DBRelation, and the mapping of class XMLEmptyET is not further refined, since it inherits the (ternary) association to DBConcept.

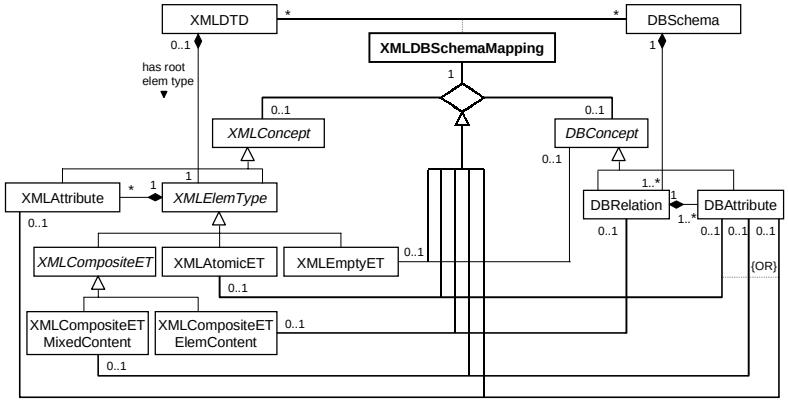

Fig. 15. Meta Schema Describing the Mapping Knowledge

Besides the mapping relationships depicted in Fig. 15, there are also relationships to class DBJoinPath (cf. Fig. 12) which are not illustrated for representation convenience. Due to space restrictions, the attributes of the various object classes are also not shown. An example mapping in terms of the filled-in meta schema is given in the Appendix.

6 Conclusion and Future Work

The main contribution of this paper is to describe X-Ray, an approach for mapping between XML DTDs and relational schemata. The mapping knowledge is not hard-coded but rather reified in terms of instances of a meta schema thus supporting autonomy of the participating DTDs and relational schemata as well as a generic integration thereof. On the basis of the meta schema, XML documents may be automaticallycomposed out of data stored within an RDBS and vice versa decomposed into relational data without any loss of information. The X-Ray prototype builds on former experience in the area of data model heterogeneity and schema heterogeneity [22], and is currently used for case studies to investigate the validity of the developed meta schema.

Future work comprises short-term tasks such as supporting the whole set of XML concepts like implicit ordering and entity definitions, as well as long-term tasks such

as integrating the XML Linking Language (XLink) [13] and the XML Pointer Language (XPointer) [12]. The latter will support the mapping of several XML documents and links between them to relational structures and vice versa. Another important aspect will be the investigation for simplifying the mapping between heterogeneous DTDs and relational schemata by, e.g., simplifying the given DTDs before mapping them [29]. In this respect it will be also analyzed, how far the definition of the mapping knowledge may be automated on the basis of the reasonable mapping patterns described above. Leaving optimization issues aside, an automatically generated default mapping should be possible. If both legacy DTDs and legacy relational schemata are involved, however, schema heterogeneity will impede an automatic mapping.

References

[1] Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, 2000

[2] Atzeni, P., Ceri, S., Paraboschi, S., Torlone, R.: Database Systems – Concepts, Languages and Architectures. Mc Graw Hill, 1999

[3] Bourret, R.: XML and Databases. Technical University of Darmstadt, http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm, June, 2000

[4] Bourret, R., Bornhövd, C., Buchmann, A.P.: A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases. 2nd Int. Workshop on Advanced Issues of EC and Web-based Information Systems (WECWIS), San Jose, California, June, 2000

[5] Böhm, K., Aberer, K.: HyperStorM - Administering Structured Documents Using Object-Oriented Database Technology. Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada, June 1996

[6] Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0, W3C Recommendation. http://www.w3.org/TR/1998/REC-xml-19980210, February, 1998

[7] Bray, T., Frankston, C., Malhotra, A.: Document Content Description for XML, Submission to the World Wide Web Consortium, http://www.w3.org/TR/NOTE-dcd, July 1998.

[8] Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E., Subramanian, S.: XPERANTO: Publishing Object-Relational Data as XML. Int. Workshop on the Web and Databases (WebDB), Dallas, May, 2000

[9] Cattell, R. G. G., Barry, D. K. (eds.): The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publishers, January, 2000

[10] Ceri, S., Fraternali, P., Paraboschi, S.: Design Principles for Data-Intensive Web Sites. ACM SIGMOD Record, Vol. 24, No. 1, March 1999

[11] Ceri, S., Fraternali, P., Paraboschi, S.: XML: Current Developments and Future Challenges for the Database Community. Proc. of the 7th Int. Conf. on Extending Database Technology (EDBT), Springer, LNCS 1777, Konstanz, March, 2000

[12] DeRose, S., Daniel, R., Maler, E.: XML Pointer Language (XPointer). W3C Working Draft, http://www.w3.org/TR/xptr, December, 1999

[13] DeRose, S., Maler, E., Orchard, D., Trafford, B.: XML Linking Language (XLink). W3C Working Draft, http://www.w3.org/TR/xlink, February, 2000

[14] Deutsch, A., Fernandez, M., Suciu, D.: Storing Semistructured Data in Relations. Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, Jerusalem, Jan., 1999

[15] Ehmayer, G., Kappel, G., Reich, S.: Connecting Databases to the Web - A Taxonomy of Gateways. Proc. of the 8th Int. Conf. on Database and Expert Systems Applications (DEXA), Springer LNCS 1308, Toulouse, September, 1997

[16] Fernandez, M., Tan, W-C., Suciu, D.: SilkRoute: Trading between Relations and XML. 9th Int. World Wide Web Conf. (WWW), Amsterdam, May, 2000
[17] Florescu, D., Levy, A., Mendelzon, A.: Database Techniques for the World Wide Web: A Survey. ACM SIGMOD Record, Vol. 27, No. 3, September, 1998
[18] Florescu, D., Kossmann, D.: Storing and Querying XML Data Using an RDBMS. IEEE Data Engineering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999
[19] Gardarin, G., Sha, F., Dang-Ngoc, T.-T.: XML-based Components for Federating Multiple Heterogeneous Data Sources. Proc. of the 18th Int. Conf. on Conceptual Modeling (ER), Paris, Nov., 1999
[20] Goldman, R., McHugh, J., Widom, J.: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Proc. of the 2nd Int. Workshop on the Web and Databases (WebDB), Philadelphia, June, 1999
[21] Kanne, C.-C., Moerkotte, G.: Efficient Storage of XML Data. Proc. Of the 16th Int. Conf. On Data Engineering (ICDE), San Diego, March, 2000
[22] Kappel, G., Preishuber, S., Pröll, E., Rausch-Schott, S., Retschitzegger, W., Wagner, R.R., Gierlinger, Ch.: COMan - Coexistence of Object-Oriented and Relational Technology. Proc. of the 13th Int. Conf. on the Entity-Relationship Approach (ER), Manchester, December, 1994
[23] Klettke, M., Meyer, H.: XML and Object-Relational Database Systems - Enhancing StructuralMappings Based on Statistics. Int. Workshop on the Web and Databases (WebDB), Dallas, May, 2000
[24] Object Design, Inc.: An XML Data Server for Building Enterprise Web Applications. http://www.odi.com/excelon/XMLResource/build_ent_web_apps.pdf, 1999
[25] Pröll, B., Sighart, H., Retschitzegger, W., Starck, H.: Ready for Prime Time - Pre-Generation of Web Pages in TIScover. Proc. of the 8th Int. ACM Conference on Information and Knowledge Management (CIKM), Kansas City, Missouri, November, 1999
[26] Raumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999
[27] Schmidt, A. R., Kersten, M. L., Windhouwer, M. A., Waas, F.: Efficient Relational Storage and Retrieval of XML Documents. Workshop on the Web and Databases (WebDB), Dallas, May, 2000
[28] Schöning, H., Wäsch, J.: Tamino – An Internet Database System. Proc. of the 7th Int. Conf. on Extending Database Technology (EDBT), Springer, LNCS 1777, Konstanz, March, 2000
[29] Shanmugasundaram, J., et al.: Relational Databases for Querying XML Documents: Limitations and Opportunities. Proc. of the 25th Int. Conf. On Very Large Data Bases (VLDB), Edinburgh, 1999
[30] Shoens, K., et al.: The Rufus system: Information organization for semi-structured data. Proc. of the Int. Conf. On Very Large Data Bases (VLDB), Dublin, Ireland, 1993
[31] Surjanto, B., Ritter, N., Loeser, H.: XML Content Management based on Object-Relational Database Technology. Proc. Of the 1st Int. Conf. On Web Information Systems Engineering (WISE), Hongkong, June 2000
[32] Widom, J.: Data Management for XML - Research Directions. IEEE Data Engineering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999
[33] W3C - World-Wide-Web Consortium. http://www.w3.org, 2000
[34] VanZwol, R., Apers, P., Wilschutz, A.: Implementing Semi Structured Data with Moa. Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, Jerusalem, Jan., 1999

Appendix: Exemplary Mapping Knowledge

In the following, an expemplary mapping in terms of the filled-in meta schema is given for the example denoted in Fig. 8. The mapping knowledge is illustrated by listing the relations being part of the relational meta schema.

1. XML DTD

XMLDTD

@XMLDTD	XMLRootElemType
accomXMLDTD	accommodations

XMLContentParticle

@XMLDTD	@XMLCP	XMLKind
accomXMLDTD	accommodations	compositeETWithElemContent
accomXMLDTD	accommodation	compositeETWithElemContent
accomXMLDTD	name	atomicET
accomXMLDTD	address	compositeETWithElemContent
accomXMLDTD	street	atomicET
accomXMLDTD	village	atomicET
accomXMLDTD	country	atomicET
accomXMLDTD	email	atomicET
accomXMLDTD	phone	emptyET
accomXMLDTD	acceptsCreditCard	emptyET
accomXMLDTD	facilities	emptyET
accomXMLDTD	sauna	emptyET
accomXMLDTD	pool	emptyET
accomXMLDTD	description	compositeETWithMixedContent
accomXMLDTD	rating	atomicET
accomXMLDTD	cp1	sequence
accomXMLDTD	cp2	sequence

XMLAttribute

@XMLAttId	XMLDTD	XMLElemType	XMLAttribute	XMLAttType	XMLDefaultDecl	XMLAttDefaultVal
1	accomXMLDTD	accommodation	id	CDATA	#REQUIRED	NULL
2	accomXMLDTD	accommodation	state	CDATA	#FIXED	Austria
3	accomXMLDTD	accommodation	kind	enum	defaultVal	hotel
4	accomXMLDTD	address	postalCode	CDATA	#REQUIRED	NULL
5	accomXMLDTD	village	yearOfFoundation	CDATA	#IMPLIED	NULL
6	accomXMLDTD	phone	number	CDATA	#REQUIRED	NULL
7	accomXMLDTD	sauna	available	CDATA	#REQUIRED	NULL

XMLAttValEnum

@XMLAttId	@XMLAttEnumValue
3	hotel
3	motel

XMLCompositeETElemContent

@XMLDTD	@XMLCompositeET	XMLOuterMostCP	XMLCardinality
accomXMLDTD	accommodations	accommodation	*
accomXMLDTD	accommodation	cp1	1
accomXMLDTD	address	cp2	1

XMLSequence_CP				
@XMLDTD	**@XMLCompositeCP**	**@XMLPosition**	**XMLComponentCP**	**XMLCardinality**
accomXMLDTD	cp1	1	name	1
accomXMLDTD	cp1	2	address	1
accomXMLDTD	cp1	3	email	*
accomXMLDTD	cp1	4	phone	+
accomXMLDTD	cp1	5	acceptsCreditCard	?
accomXMLDTD	cp1	6	facilities	1
accomXMLDTD	cp1	7	sauna	1
accomXMLDTD	cp1	8	pool	*
accomXMLDTD	cp1	9	description	?
accomXMLDTD	cp2	1	street	1
accomXMLDTD	cp2	2	village	1
accomXMLDTD	cp2	3	country	1

XMLChoice_CP			
@XMLDTD	**@XMLCompositeCP**	**@XMLComponentCP**	**XMLCardinality**

XMLCompositeETMixedContent_Content		
@XMLDTD	**@XMLCompositeET**	**@XMLComponentET**
accomXMLDTD	description	rating

2. Database Schema

DBSchema		
@DBSchema	**DBName**	**DBConnectString**
accomDBSchema	oracle816	xray@oradb816

DBAttribute					
@DBAttId	**DBSchema**	**DBRelation**	**DBAttribute**	**DBIsKey**	**DBDataType**
1	accomDBSchema	Accommodation	AccID	y	NUMBER
2	accomDBSchema	Accommodation	Name	n	VARCHAR(50)
3	accomDBSchema	Accommodation	Kind	n	CHAR(10)
4	accomDBSchema	Accommodation	Street	n	VARCHAR(50)
5	accomDBSchema	Accommodation	VillageName	n	VARCHAR(50)
6	accomDBSchema	Accommodation	AcceptsCreditCards	n	CHAR(3)
7	accomDBSchema	Accommodation	Sauna	n	CHAR(3)
8	accomDBSchema	Village	Name	y	VARCHAR(50)
9	accomDBSchema	Village	PostalCode	n	VARCHAR(10)
10	accomDBSchema	Village	CountryID	n	NUMBER
11	accomDBSchema	History	VillageName	y	VARCHAR(50)
12	accomDBSchema	History	YearFound	n	Date
13	accomDBSchema	Country	CountryID	y	NUMBER
14	accomDBSchema	Country	Name	n	VARCHAR(50)
15	accomDBSchema	Phone	AccID	y	NUMBER
16	accomDBSchema	Phone	Number	y	VARCHAR(30)
17	accomDBSchema	EmailAddress	AccID	y	NUMBER
18	accomDBSchema	EmailAddress	Email	y	VARCHAR(30)
19	accomDBSchema	Pool	AccID	y	NUMBER
20	accomDBSchema	Pool	Name	y	VARCHAR(50)
21	accomDBSchema	ActualRating	AccID	y	NUMBER
22	accomDBSchema	ActualRating	RatingID	y	NUMBER
23	accomDBSchema	ActualRating	RatingOrder	n	NUMBER
24	accomDBSchema	PossibleRating	RatingID	y	NUMBER
25	accomDBSchema	PossibleRating	Rating	n	VARCHAR(30)
26	accomDBSchema	RatingDescription	AccID	y	NUMBER
27	accomDBSchema	RatingDescription	RatingOrder	y	NUMBER
28	accomDBSchema	RatingDescription	Description	n	VARCHAR(50)

DBJoinSegment					
@DBRelShipId	**@DBAtt1**	**@DBAtt2**	**DBRelation1**	**DBRelation2**	**DBSchema**
1	VillageName	Name	Accommodation	Village	accomDBSchema
2	Name	VillageName	Village	History	accomDBSchema
3	CountryID	CountryID	Village	Country	accomDBSchema
4	AccID	AccID	Accommodation	Phone	accomDBSchema
5	AccID	AccID	Accommodation	EmailAddress	accomDBSchema
6	AccID	AccID	Accommodation	Pool	accomDBSchema
7	AccID	AccID	Accommodation	ActualRating	accomDBSchema
8	RatingID	RatingID	ActualRating	PossibleRating	accomDBSchema
9	AccID	AccID	Accommodation	RatingDescription	accomDBSchema

DBJoinInfo		
@DBJoinPathId	**@DBRelShipId**	**DBSchema**
1	1	accomDBSchema
2	2	accomDBSchema
3	1	accomDBSchema
3	3	accomDBSchema
4	4	accomDBSchema
5	5	accomDBSchema
6	6	accomDBSchema
7	7	accomDBSchema
7	8	accomDBSchema
8	9	accomDBSchema

3. XML DB Schema Mapping

XMLDBSchemaMapping		
@MappingId	**XMLDTD**	**DBSchema**
1	accomXMLDTD	accomDBSchema

XMLETMapping		
@MappingId	**@XMLElemType**	**KindOfMapping**
1	accommodations	NULL
1	accommodation	ET_R_{direct}
1	name	ET_A_{direct}
1	address	NULL
1	street	ET_A_{direct}
1	village	$ET\_A_{indirect}$
1	country	$ET\_A_{indirect}$
1	email	$ET\_A_{indirect}$
1	phone	$ET\_A_{indirect}$
1	acceptsCreditCard	ET_A_{direct}
1	facilities	NULL
1	sauna	NULL
1	pool	$ET\_A_{indirect}$
1	description	$ET\_A_{indirect}$
1	rating	$ET\_A_{indirect}$

XMLAttributeMapping				
@MappingId	**@XMLAttId**	**KindOfMapping**	**DBAttId**	**DBJoinPathId**
1	1	A_A_{direct}	1	NULL
1	2	NULL	NULL	NULL
1	3	A_A_{direct}	3	NULL
1	4	$A\_A_{indirect}$	9	1
1	5	$A\_A_{indirect}$	12	2
1	6	A_A_{direct}	16	NULL
1	7	A_A_{direct}	7	NULL

XMLElemTypeToRelationMapping		
@MappingId	**@XMLElemType**	**DBRelation**
1	accommodation	Accommodation

XMLElemTypeToAttMapping		
@MappingId	**@XMLElemType**	**DBAttId**
1	name	2
1	street	4
1	village	8
1	country	14
1	email	18
1	phone	16
1	acceptsCreditCard	6
1	pool	20
1	description	28
1	rating	25

XMLCompositeETMixedContent_Content_JoinPath			
@MappingId	**@XMLCompositeET**	**@XMLComponentET**	**DBJoinPathId**
1	description	rating	7

XMLOuterMostCP_JoinPath		
@MappingId	**@XMLCompositeET**	**DBJoinPathId**

XMLSequenceContent_JoinPath			
@MappingId	**@XMLContentParticle**	**@XMLPosition**	**DBJoinPathId**
1	cp1	3	5
1	cp1	4	4
1	cp1	8	6
1	cp1	9	8
1	cp2	2	1
1	cp2	3	3

XMLChoiceContent_JoinPath			
@MappingId	**@XMLContentParticle**	**@XMLComponentCP**	**DBJoinPathId**