X-Ray - Towards Integrating XML and Relational Database Systems

Gerti Kappel, Elisabeth Kapsammer, Stefan Rausch-Schott, Werner Retschitzegger

Institute of Applied Computer Science, Department of Information Systems (IFS)
University of Linz, Altenbergerstraße 69, A-4040 Linz, Austria
{gk, ek, srs, 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. 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 [11], [20]. This allows to easily handle both retrieval and update of large amounts of data in a consistent way on a large distributed scale [9]. Besides using databases at the *content level*, the Extensible Markup Language (XML) [28] 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], [6], [27]. XML is a subset of SGML. As such, an XML document consists of possibly nested *elements* rooted in a single element. Elements, whose boundaries are delimited by *start-tags* and *end-tags*, may comprise *attributes*, whereby both are able to contain *values*. An XML document can be associated with a type specification called *document type definition (DTD)*, containing user-defined *element types* and *attribute specifications* which allow to describe the meaning of the content. Note, that there are already several efforts to replace DTDs by means of richer XML schema definition languages [28]. However, since there is no standard up to now, the rest of the paper builds on DTDs.

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 [7], [27]. Regarding the kind of DBS used for the integration, one can distinguish four different approaches [2], [12]. First, *special-purpose DBS* are particularly tailored to store, retrieve, and update XML documents. Examples thereof are research prototypes such as Rufus [25], Lore [14], Strudel [11] and Natix [15] as well as commercial systems such as eXcelon [19] and Tamino [23]. Second, because of the rich data modeling capabilities of *object-oriented DBS*, they are well-suited for storing hypertext documents [4], [29]. 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 [12], [13], [22], [26]. 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 [3], [5], [8], [10], [18], [24]. 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 [17]. 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 could be because of different goals pursued during design like redundant representation of information versus normalization [17].

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 [8]. This, however, contradicts the requirement of autonomy. Second, there is only a certain pre-defined way of mapping provided [24], 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 [3] 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 [16], 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 introduces different mapping possibilities between XML and RDBS. Based on these mapping possibilities, Section 3 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 4. Finally, Section 5 concludes the paper with a short summary and gives an outlook to future work.

2 Basic Kinds of Mappings Between XML and RDBS

There are several 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. 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 3) and for designing the meta schema (cf. Section 4).

When considering the structuring mechanisms of XML and RDBS, three basic kinds of mappings may be distinguished, which are denoted in Fig. 1 together with an example. Note, that XML elements and attributes are represented in terms of UML objects [21].

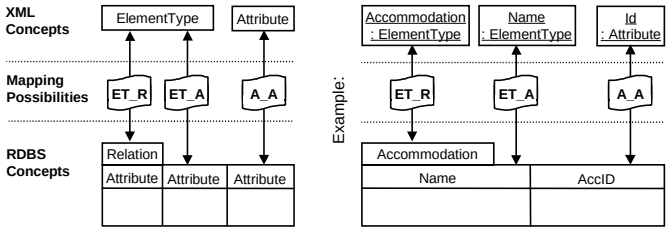

Fig. 1. Basic Kinds of Mappings

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.

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.

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.

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 that values of database attributes are mapped to values of XML elements or attributes. Thus, it makes sense that ET_R mappings occur together with ET_A and A_A mappings. 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. 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, there are cases where a mapping is mandatory, e.g., if a certain constraint requires the existence of a value within the XML document (cf. Section 3.2).

The three basic kinds of mappings introduced above can be further refined with respect to the determination of an element type's base relation. For this, one has to look at the nesting hierarchy built by element types containing other element types. The former are furtheron called *composite element types*, the latter *component element types*. 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. 2 (cf. also the more

comprehensive example given in Fig. 3), 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. 2 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.

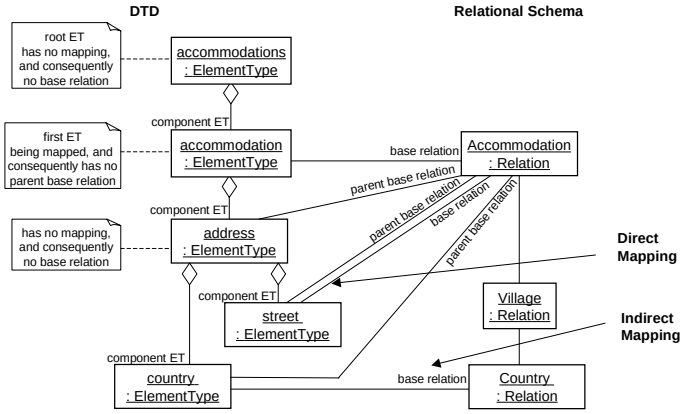

Fig. 2. Exemplary Mappings

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. 2, 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.

3 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 3.1) and *characteristics of the XML attribute* (cf. Section 3.2). In order to illustrate the subsequent investigations we provide a comprehensive running example building on the ones given in the previous section. The example is intended to show as many mapping possibilities as possible. Fig. 3 depicts the running example in terms of a DTD and in terms of a relational schema.

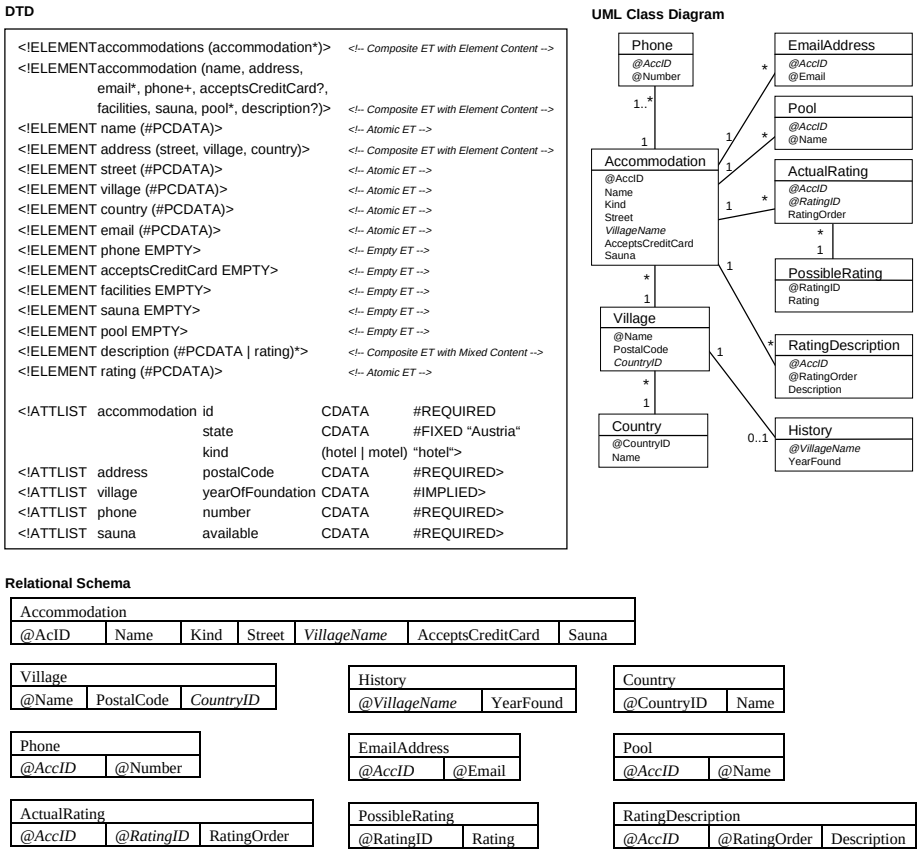

Fig. 3. Exemplary DTD, Relational Schema, and UML Class Diagram

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. The description of this example is given from a mapping viewpoint throughout the forthcoming subsections.

3.1 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. 4, these decisive characteristics can be categorized into three orthogonal dimensions comprising the *kind of element type*, if it *contains attributes*, and its *cardinality*.

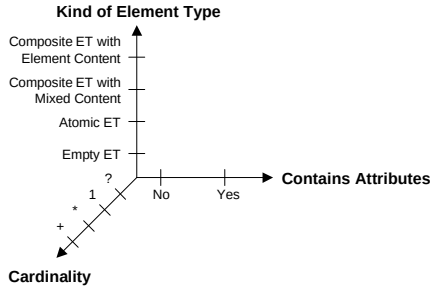

Fig. 4. Orthogonal Dimensions Characterizing XML Element Types

The most simple kind of element type contains an atomic domain (`#PCDATA`), only, and is furtheron called *atomic element type*. Composite element types (cf. Section 2) may have an atomic domain in addition to component element types, and thus are further distinguished into *composite element types with mixed content* and *composite element types with element content.* Concerning the latter, it has to be specified whether component element types occur in a *sequence* (“,”), or in a *choice* (“|”) meaning that they are mutual exclusive. This is not applicable to the former since in this case component element types are allowed to occur in a choice with cardinality ‘*’, only.

Element types that neither contain component element types nor have an atomic domain are called *empty element types.* Each element type no matter if it is an atomic, composite, or empty element type may contain *attributes*. Finally, *cardinality constraints* specify how often elements of a certain element type occur as component elements 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). Depending on the combination of these characteristics, certain reasonable mappings can be determined as shown in Table 1. 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.

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 1. 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 (cf. Section 2) 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`, which are 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.

3.2 XML Attribute Characteristics

The mapping of XML attributes depends on two orthogonal dimensions comprising the *type of the XML attribute* and its *default declaration*.

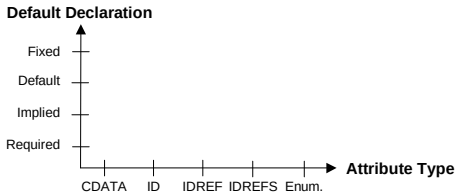

Fig. 5. Orthogonal Dimensions Characterizing XML Attributes

The type of the XML attribute may be a *string type* (CDATA), an *enumeration type*, or some special type including, e.g., ID and IDREF(S) responsible for unique identification of elements within an XML document and for referencing an element (IDREF) or several elements (IDREFS) having an attribute of type ID, respectively. For the sake of readability and space restrictions, we do not consider all possible types of XML attributes but rather the more important ones. The default declaration expresses whether a value is required (#REQUIRED), optional (#IMPLIED), fixed (#FIXED <ConstValue>) or default (<DefaultValue>).

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 [17].

Table 2. 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.

4 The X-Ray Meta Schema

The different kinds of mappings proposed in the previous sections 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 automati-

cally 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.

The meta schema consists of three components describing the relevant meta knowledge, namely `DBSchema`, `XMLDTD` and `XMLDBSchemaMapping` (cf. top part of Fig. 9). 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`.

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.

4.1 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. As illustrated in Fig. 6 `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.

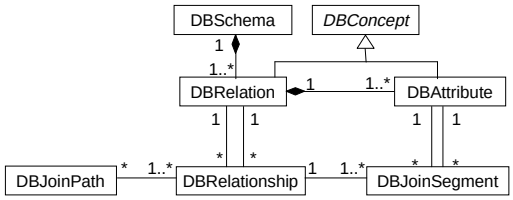

Fig. 6. Meta Schema of the Relational Schema

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.

4.2 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. 7) 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*.

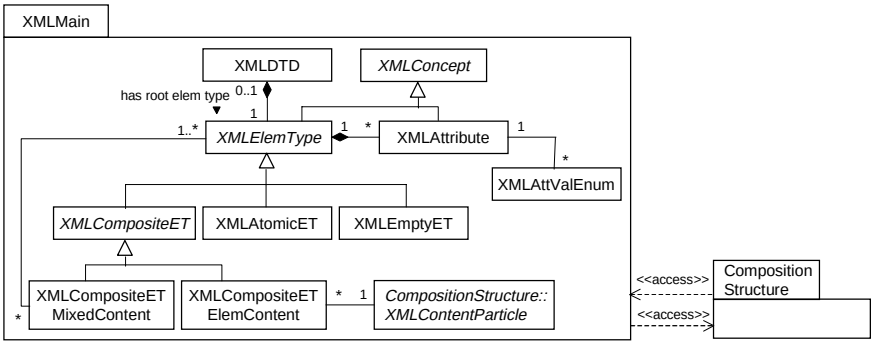

Fig. 7. Meta Schema of the DTD

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 kinds of element types described in Section 2 and 3, `XMLElemType` is specialized into `XMLAtomicET`, `XMLEmptyET`, and `XMLCompositeET`. The latter is further specialized into `XMLCompositeETMixedContent` and `XMLCompositeETElemContent`.

The nesting structure of an `XMLCompositeETElemContent` is described by the package `CompositionStructure` (cf. Fig. 8). 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.

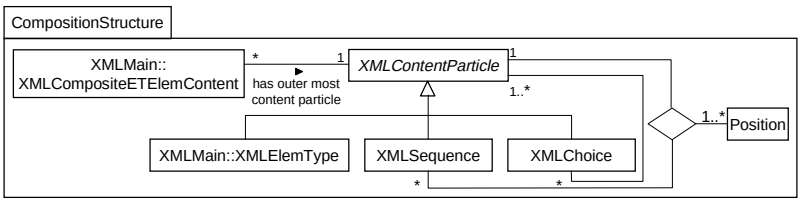

Fig. 8. Meta Schema of the XML Composition Structure

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.

4.3 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. 9 illustrates these mapping relationships denoting them with bold lines.

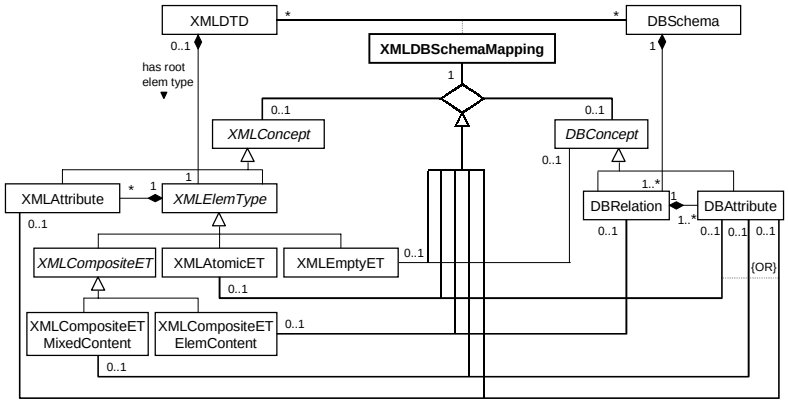

Fig. 9. Meta Schema Describing the Mapping Knowledge

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 3, 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. Besides the mapping relationships depicted in Fig. 9 there are also relationships to class DBJoinPath (cf. Fig. 6) 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 [17].

5 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 automatically composed 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 [16], 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) and the XML Pointer Language (XPointer) [28]. 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 [24]. 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] Bourret, R.: XML and Databases. Technical University of Darmstadt, http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm, June, 2000

[3] 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

[4] 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

[5] 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

[6] Ceri, S., Fraternali, P., Paraboschi, S.: Design Principles for Data-Intensive Web Sites. ACM SIGMOD Record, Vol. 24, No. 1, March 1999
[7] 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
[8] 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
[9] 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
[10] Fernandez, M., Tan, W-C., Suciu, D.: SilkRoute: Trading between Relations and XML. 9th Int. World Wide Web Conf. (WWW), Amsterdam, May, 2000
[11] Florescu, D., Levy, A., Mendelzon, A.: Database Techniques for the World Wide Web: A Survey. ACM SIGMOD Record, Vol. 27, No. 3, September, 1998
[12] 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
[13] 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
[14] 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
[15] Kanne, C.-C., Moerkotte, G.: Efficient Storage of XML Data. Proc. Of the 16th Int. Conf. On Data Engineering (ICDE), San Diego, March, 2000
[16] 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
[17] Kappel, G., Kapsammer, E., Retschitzegger, W.: X-Ray – Towards Integrating XML and Relational Database Systems. Technical Report, Department of Information Systems (IFS), JKU Linz, http://www.ifs.uni-linz.ac.at/ifs/research/publications/papers00.html, July, 2000
[18] 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
[19] Object Design, Inc.: An XML Data Server for Building Enterprise Web Applications. http://www.odi.com/excelon/XMLResource/build_ent_web_apps.pdf, 1999
[20] 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
[21] Raumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999
[22] 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
[23] 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
[24] 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
[25] 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
[26] 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
[27] Widom, J.: Data Management for XML - Research Directions. IEEE Data Engineering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999
[28] W3C - World-Wide-Web Consortium. http://www.w3.org, 2000
[29] 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