Providing Semantically Equivalent, Complete Views for Multilingual Access to Integrated Data

Iryna Kozlova, Norbert Ritter, Martin Husemann

University of Hamburg, Germany

{kozlova, ritter, husemann}@informatik.uni-hamburg.de

Abstract

Today, there are several approaches supporting an integrated processing of XML and (object-)relational data sources. However, each of these approaches favors a single data model for integration purposes and, thus, provides just a single query interface, usually either SQL or XQuery. In order to be more flexibly applicable, our SQXML integration system provides both query languages at its interface. The challenge here is to ensure that the complete information resulting from the data integration process is accessible via both query languages. In this paper we prove that our SQXML integration approach meets this objective. For this purpose we examine the schema transformation, mapping and integration steps performed in the SQXML integration and view generation process in order to show that the resulting SQL and XML views are semantically equivalent in the sense that each encompasses the complete information resulting from schema integration.

Keywords: Information integration, SQL, XML Schema

1 Introduction

The goal of information and database integration research is to provide the necessary foundations for seamless access to a variety of data sources which differ in their formats, data models, and access mechanisms, as well as to manage a plenitude of data spread across these sources. Efficient methods are needed for the integrated processing of data in distributed heterogeneous environments. The most widespread data storage systems in modern enterprises comprise at the first place relational and object-relational databases, while a considerable amount of information resides in native XML databases (like Tamino, X-Hive/DB) or collections of XML documents with a native XML-language access (like XQuery) providing means for storage and management of XML documents in a native format. The parallel existence of these popular types of DBMSs leads to the necessity to integrate data stored in (object-)relational and XML data sources.

In this paper, we focus on the effective integrated processing of data that belongs to the same or to closely related domains and is represented in (object-)relational and XML data models. Integration is achieved by introducing a middleware layer that provides a uniform interface for accessing data in its native data sources. Our approach, the SQXML Integration System or just 'SQXML' (Kozlova 2005), supports a schema integration process that generates a global schema comprising the entire information of all local data sources. In order to be easy to use and flexibly accessible for different kinds of applications, SQXML provides the complete amount of integrated information in both formats, i.e., as an (object-) relational as well as an XML schema[1]. Note that completeness is required in order to entirely disburden users from the heterogeneity of local data sources. Further, offering the integrated schema in several formats supports multilingualism, which in turn allows for flexible exploitation of the integrated information.

Due to the different degrees of expressiveness of the (object-)relational and XML data models it is not a trivial task to ensure that both views as provided by SQXML are semantically equivalent in the sense that they both encompass the complete amount of information stored in all local sources in an integrated way. Note that the proceeding as illustrated in Fig. 1 (a) does generally not lead to complete, semantically equivalent (SQL and XQuery) views. To achieve this goal, a proceeding as illustrated in Fig. 1 (b), which has been chosen in the SQXML approach, is best suited, as we will show in the course of this paper. The basic idea is to develop and exploit a unified metamodel that appropriately integrates all object-relational and XML modeling concepts. As we will see, the use of the unified metamodel as a common metamodel during the schema integration process offers considerable advantages especially regarding what we call semantic equivalence and completeness of the integrated global schemas presented to users and applications.

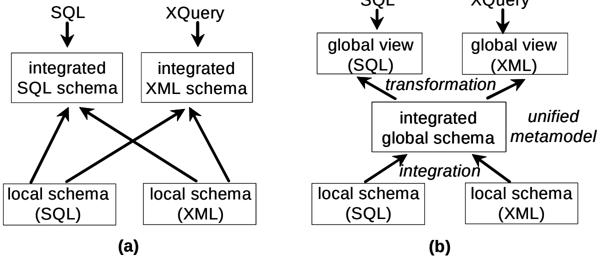


Fig. 1. Schema integration approaches.

Copyright (c) 2007, Australian Computer Society, Inc. This paper appeared at the Twenty-Sixth International Conference on Conceptual Modeling - ER 2007 - Tutorials, Posters, Panels and Industrial Contributions, Auckland, New Zealand. Conferences in Research and Practice in Information Technology, Vol. 83. John Grundy, Sven Hartmann, Alberto H. F. Laender, Leszek Maciaszek and John F. Roddick, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

[1] Throughout this paper we use the term 'XML schema' for both the metamodel as well as a concrete (XML) schema specification.

2 Related work

The goal of investigations in the field of information integration is to provide integrated access to multiple autonomous data sources. Existing approaches on management of heterogeneous data sources vary in their methods and applied methodologies. As related to our work we mention here researches that specially apply the idea of a global mediated schema as a core part of the integration process.

One strategy often applied in integration systems is to use a *predefined* global schema (also called *target* schema) and discover the relationship between the target schema and the local source schemas to access the data. In such systems, the mediated schema is manually designed to satisfy particular needs of the end-user and to capture special aspects of the domain of the user interest. This strategy is realized for example in BRIITY (Haerder, Sauter and Thomas 1999) and Clio (Haas et al. 2005).

The predefined global schema may be incomplete, representing only a special part of the information stored in the local data sources. In contrast, in the SQXML Integration System the constructed global schema fully integrates the entire information from the local sources and represents it to the end-user in a desired format.

In contrast to approaches based on wrapper-mediator architecture (Wiederhold 1992), the SQXML Integration System provides the end-user with a *complete* integrated global schema constructed in an almost automated way. Moreover, access to the entire information is provided via SQL and XQuery so that no special query language has to be learned by the end-user.

One of the central problems that have to be resolved by an integration system that operates on heterogeneous data sources is the choice of a proper *common data model*. The special choice of the SQXML common data model allows for provision of global views in both SQL and XML representations as well as access to the entire information via SQL and XQuery.

To the best of our knowledge, there is no other integration system that provides a specially designed common data model exploiting all features of XML and (object-)relational data models. Moreover, one of the special requirements on our system is to construct both global schema representations, SQL and XML, which is achievable only in case of a special unified data model.

3 Schema integration in SQXML

During the SQXML schema integration process a global schema integrating a given set of local source schemas is constructed. As motivated and illustrated in Fig. 1 (b) we apply the so-called SQXML metamodel (Kozlova et al. 2005) for that purpose. There are several reasons for the development and exploitation of the SQXML metamodel:

- Resolution of structural heterogeneity;
- Automation of schema matching;
- Ensuring completeness of the integrated schema;
- Ensuring equivalence of global views.

Although the former two of the mentioned reasons are very important for the overall SQXML approach (see Kozlova et al. 2005 for details) they are not in the focus of this paper, where we want to emphasize the latter two.

The SQXML metamodel is based on the Common Warehouse Metamodel (CWM) (OMG CWM 2003). We developed the SQXML metamodel by unifying the metamodels of SQL and XML[2]. Thus, each SQL schema and each XML Schema can be unambiguously transformed into an equivalent SQXML schema.

Here we briefly mention some of the SQXML modeling concepts. The top-level container of SQXML is *Schema*, which can contain *TypeDefinitions*, *Entities* and *Procedures*. The *TypeDefinition* class is an abstract class extended by either *SimpleType* or *ComplexType*. The class *Entity* unifies SQL's *Column* with XML Schema's *ElementDeclaration* and *AttributeDeclaration*. The *ComplexType* class integrates structured types: XML Schema's *ComplexTypeDefinition* with SQL's *Table* and *SQLStructuredType*. In order to preserve the origin structure of model classes, the SQXML metamodel was extended by a special *Context* class, intended to store information about the origins of integrated structures (Section 3.1). For example, with the help of the *Context* class each SQXML *Entity* can easily be traced back to an *ElementDeclaration* or an *AttributeDeclaration*.

The SQXML metamodel supports most features of Core SQL:2003 as well as the optional package PKG006 "Basic object support" and some features from the optional package PKG007 "Enhanced object support" (Information technology 2003), thus providing basic object-relational features. Further, it supports the main modeling concepts of XML Schema according to the XML Schema Recommendations and uses the XML Schema data types (XML Schema Part 1, 2 2006). The SQL data types have been converted to XML Schema simple types using the SQL/XML standard (International Organization for Standardization 2005).

The overall SQXML integration and view generation process is illustrated in Fig. 2 as a bottom-up process.

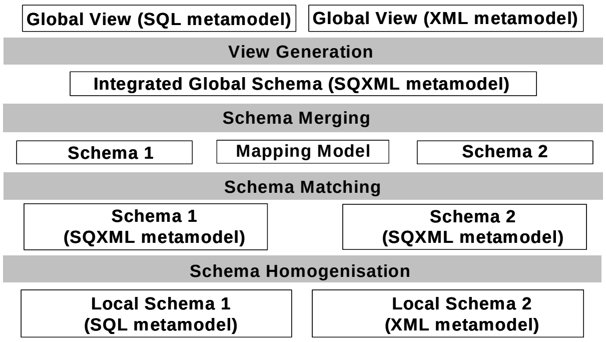

Global View (SQL metamodel)
Global View (XML metamodel)
View Generation
Integrated Global Schema (SQXML metamodel)
Schema Merging
Schema 1
Mapping Model
Schema 2
Schema Matching
Schema 1 (SQXML metamodel)
Schema 2 (SQXML metamodel)
Schema Homogenisation
Local Schema 1 (SQL metamodel)
Local Schema 2 (XML metamodel)

Fig. 2. SQXML integration and view generation process.

[2] Whenever it is clear from the context, we use the terms 'SQL' and 'XML' for both the languages as well as the corresponding data models. With 'SQL' we always refer to the SQL:2003 standard. Further, we use 'SQXML' for our integration approach/system as well as the corresponding data model.

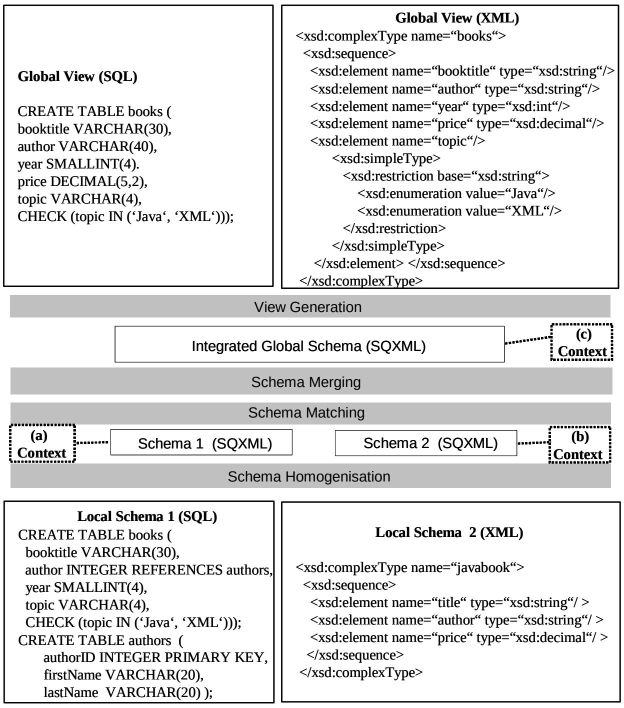
First, the structural heterogeneity between the local SQL and XML schemas is resolved by transforming each of them into an equivalent SQXML schema (*schema homogenization*). The semantic heterogeneity is resolved in two steps *(schema matching* and *schema merging*). Finally, two global views are generated from the integrated global schema resulting from schema merging, one for SQL users/applications and one for XML users/applications, so that the entire integrated information can be accessed via SQL and XQuery correspondingly (Kozlova et al. 2006). Details on the view generation step and the corresponding algorithms can be found in (Kozlova and Ritter 2006). These global views can be assured to be semantically equivalent and complete in the sense that both encompass the whole amount of information as contained in the local data sources.

3.1 Context

The *Context* class has been designed to store (for each *SQXML Entity* and for each *SQXML ComplexType*) information which is needed later on in the integration and view generation process (see Figure 2). For example, in the view generation step it can be assured that local structures which have not been changed during integration will after view generation appear in the same way as in the corresponding local source.

During the schema homogenization step, in which local schemas are transformed to SQXML, the *Context* class gathers information about original names, structures, integrity constraints, and access relating to the transformed schema elements.

Figure 3 illustrates two sample local schemas (at the bottom), the resulting global views (at the top), and the context entries stored during the schema integration process (denoted *(a), (b), (c) Context* boxes).



Global View (SQL)

```
CREATE TABLE books (
booktitle VARCHAR(30),
author VARCHAR(40),
year SMALLINT(4).
price DECIMAL(5,2),
topic VARCHAR(4),
CHECK (topic IN (‘Java‘, ‘XML‘)));
```

Global View (XML)

```
<xsd:complexType name=“books“>
 <xsd:sequence>
  <xsd:element name=“booktitle“ type=“xsd:string“/>
  <xsd:element name=“author“ type=“xsd:string“/>
  <xsd:element name=“year“ type=“xsd:int“/>
  <xsd:element name=“price“ type=“xsd:decimal“/>
  <xsd:element name=“topic“/>
     <xsd:simpleType>
       <xsd:restriction base=“xsd:string“>
         <xsd:enumeration value=“Java“/>
         <xsd:enumeration value=“XML“/>
       </xsd:restriction>
     </xsd:simpleType>
   </xsd:element> </xsd:sequence>
</xsd:complexType>
```

Local Schema 1 (SQL)

```
CREATE TABLE books (
  booktitle VARCHAR(30),
  author INTEGER REFERENCES authors,
  year SMALLINT(4),
  topic VARCHAR(4),
  CHECK (topic IN (‘Java‘, ‘XML‘)));
CREATE TABLE authors  (
     authorID INTEGER PRIMARY KEY,
     firstName VARCHAR(20),
     lastName  VARCHAR(20) );
```

Local Schema 2 (XML)

```
<xsd:complexType name=“javabook“>
 <xsd:sequence>
  <xsd:element name=“title“ type=“xsd:string“/ >
  <xsd:element name=“author“ type=“xsd:string“/ >
  <xsd:element name=“price“ type=“xsd:decimal“/ >
 </xsd:sequence>
</xsd:complexType>
```

Fig. 3. Sample schemas and context examples.

As illustrated by the example in Figure 3, context information is created each time an SQXML schema is created in the SQXML integration process, i.e., in the schema homogenization and in the schema merging steps (Fig.3 (a), (b) and Fig. 3 (c) resp.). Context entries are generated for each SQXML schema element, respectively.

After the schema homogenization, the *Context* class stores the following information for each SQXML entity: the original name of the element (`origName`), its structure (`origStructure`), the local source name (`origSource`), the absolute and relative paths to the element in the local source (`absolutePath`, `relativePath`), the key constraints (`origKeys`), and the internally assigned identifier (`sqxmlIDs`). Information originating from the SQL local source is marked with the prefix ’`sql`’, whereas information concerning the XML local source elements is marked with the prefix ’`xsd`’. The prefix ’`gl`’ indicates relevant information about SQXML elements. Sample context entries with the information described above are represented in Figure 4.

Figure 4 (a) shows the context entry relating to the local (SQL) schema element `booktitle`, which is an attribute of the relation `books`. Figure 4 (b) illustrates the context entry relating to the local (XML) schema element `title`, which is an element within the complex type `javabook`. The prefix ’`gl`’ here indicates that SQXML *Entity* is a new SQXML schema element to which SQL’s *Column* and XML Schema’s *ElementDeclaration* have been transformed during schema homogenization.

After the schema merging step, the local schema elements `booktitle` (local SQL schema) and `title` (local XML schema) have been integrated to the global schema element `booktitle.` The context entry relating to this global schema element is shown in Fig. 4 (c). Information stored with the prefix ’`gl`’ here indicates complete information related to the global schema element, that is, its global name, access paths in the global schema, integrity constraints, and internally assigned ID.

(c) Context of global element `booktitle`

-origName	gl:booktitle, sql:booktitle, xsd:title
-origStructure	gl:Entity, sql:Column, xsd:ElementDeclaration
-origSource	gl:globdb, sql:dbsql, xsd:dbxml
-absolutePath	gl:globschema.books.booktitle, sql:schema1.books.booktitle, xsd:root/javabook/title
-relativePath	gl:booktitle, sql:booktitle, xsd:title
-origKeys	gl:booktitle, sql:booktitle, xsd:none
-sqxmlIDs	gl:ID_3, sql:ID_1, xsd:ID_2

(a) Context of local element `booktitle`

-origName	sql:booktitle
-origStructure	gl:Entity, sql:Column
-origSource	sql:dbsql
-absolutePath	sql:schema1.books.booktitle
-relativePath	sql:booktitle
-origKeys	sql:booktitle
-sqxmlIDs	sql:ID_1

(b) Context of local element `title`

-origName	xsd:title
-origStructure	gl:Entity, xsd:ElementDeclaration
-origSource	xsd:dbxml
-absolutePath	xsd:root/javabook/title
-relativePath	xsd:title
-origKeys	xsd:none
-sqxmlIDs	xsd:ID_2

Fig. 4. Sample context entries.

4 Completeness and semantic equivalence of global views

In this section we consider the different kinds of schema transformations as performed in the overall SQXML integration process. Principally, there are four kinds of transformations; SQL-to-SQXML, XML-to-SQXML, SQXML-to-SQL, and SQXML-to-XML, where the former two are performed during the schema homogenization step and the latter two during the view generation step (cf. Figure 2).

In order to argue the completeness of the global views in the sense that there is no loss of information along all the transformations from the local schemas to the global views, we have to consider all possible transformation paths of schema elements, which are depicted in Figure 5. We address the illustrated paths in a bottom-up manner.

First of all, each local schema element, whether it belongs to a local SQL source or to a local XML source, is transformed into a unique corresponding SQXML schema element during the schema homogenization step. Since the SQXML metamodel has been designed to contain an (at least) equally expressive correspondent for each modeling concept of SQL and XML, there is no loss of information during schema homogenization.

Moving further upward along the transformation paths, we now have to consider the integration step. Regarding a single (SQXML) schema element, there are two possibilities how this schema element can be affected by the integration: It can either be merged with a corresponding schema element of the other (SQXML) schema (dotted paths), or it can be transferred unchanged into the integrated schema (solid paths, see XOR-branches in Figure 5). Transferring schema elements unchanged into the integrated schema obviously fulfills the completeness requirement w.r.t. to these schema elements. We also consider element merges as (semantically) complete since independently of the concrete merge, the information content of the merged schema elements remains available.

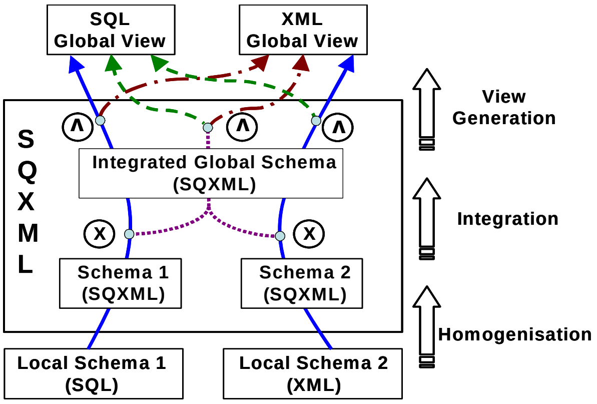


Fig. 5. Transformation paths for local schema elements.

Yet further upward along the transformation paths, each schema element of the integrated schema, whether it is an unchanged element or a merged element, needs to be transformed to be represented in both destination models (see AND-branches in Figure 5) in order to obtain the global views. We consider these transformations in three groups as indicated by the different kinds of lines (solid, dashed-dotted, and dashed) in the upper part of Figure 5.

Although the solid transformation paths are uncritical w.r.t. the completeness property, we want to briefly highlight them in order to point out the unambiguousness of schema round-trips in SQXML. As can be seen from Figure 5, the solid transformation paths represent schema round-trips of the forms SQL-to-SQXML-to-SQL and XML-to-SQXML-to-XML. SQXML ensures that such round-trips unambiguously lead to exactly the original schema through the following two facts: Firstly, the SQXML metamodel has been designed to contain exactly one modeling concept for each SQL concept and for each XML concept. Secondly, since the unambiguousness of the reverse transformation cannot be assured by the SQXML metamodel alone, the context information (see Section 3.1) gathered during the earlier transformation steps is leveraged to map back schema elements to exactly the original structures. For example, in the case of an SQL round-trip, an SQXML *ComplexType* can uniquely be converted into its original structure, either *Table* or *SQLStructuredType*, as this information is stored in the *Context* class. Similarly, in the case of an XML round-trip, an SQXML *Entity* can be uniquely converted into the original structure, i.e., either *ElementDeclaration* or *AttributeDeclaration*. This unambiguousness property of round-trips is especially relevant for local schema elements which are not merged with other schema elements during the integration step, as illustrated by the solid paths in Figure 5.

Even in the case of schema elements which have been merged during the integration step, the context information can be exploited for view generation in order to ensure unambiguousness. Reconsider the example illustrated in Figure 4. Here the global element `booktitle` results from the integration of the local SQL element `booktitle` and the local XML element `title`. From the context information the process in charge of generating the global SQL (XML) view can detect that the modeling concept of the original SQL (XML) source is *Column* (*Element Declaration*) so that the same concept can be used in the global SQL (XML) view to be generated.

The transformations marked as dashed-dotted lines can be performed without any loss of information for the following reasons: Firstly, as already mentioned earlier, the SQXML metamodel has been designed to have exactly one correspondent for each XML modeling concept and for each SQL modeling concept. Secondly, XML has a higher expressiveness than SQL. From these two points it can be concluded that the expressiveness of SQXML conforms to the expressiveness of XML and that each SQXML schema element can be transformed to XML without any loss of information.

The most critical transformations concern the completeness of the global SQL view are those marked as dashed lines in Figure 5. Transforming from SQXML to SQL is not straightforward. In the following we will discuss the restrictions and how we cope with them in SQXML.

4.1 Critical transformations from SQXML to SQL

As mentioned in Section 3, not all XML concepts do have direct correspondents in SQL. Here we discuss those specific modeling concepts that originate from XML Schema and are supported by SQXML, but have no direct correspondents in SQL. First, we consider how the SQXML data types can be converted to SQL types. Then, we discuss the XML Schema concepts that cannot be directly transformed to SQL and propose solutions for those problems.

The SQXML metamodel uses XML Schema types. SQL data types are converted to XML Schema simple types using the SQL/XML standard (Section 3). SQL/XML does not support the inverse mapping of XML Schema simple types to SQL types since some XML Schema data types such as Gregorian Calendar types are not available in the SQL type system. XML Schema types with unbounded cardinality, such as integer and string, cannot be mapped directly either.

Table 1 illustrates the correspondences between XML Schema simple types and SQL types as they are used in SQXML. Below we discuss the problems that may occur in general when performing such mappings and the solutions proposed in SQXML. The Gregorian Calendar types mentioned above (not shown in Table 1) must be mapped to user-defined structured types.

Table 1. Mapping between XML Schema simple types and SQL types.

XML Schema Type	SQL:2003 Type	Notes
string	CHAR,VARCHAR, CLOB	maxLength or Length facets
hexBinary, base64Binary	BLOB	maxLength or Length facets
gDay, date, time, dateTime	DATE, TIME, TIMESTAMP	direct mapping
decimal	DECIMAL	direct mapping
integer	INTEGER, SMALLINT	maxInclusive, maxExclusive
float; double	FLOAT; REAL, DOUBLE PRECISION	direct mapping
anyURI, NOTATION, QName	VARCHAR(100)	limited in length

Regarding *simple types with unbounded cardinality*, the following problems can be identified:

- The conversion of XML string into SQL CLOB, VARCHAR, or CHAR is lossless only in the presence of maxLength or Length facets.
- The conversion of XML base64Binary and hexBinary into SQL BINARY LARGE OBJECT is lossless only in the presence of maxLength or Length facets.
- XML integer can be exactly mapped to SQL INTEGER or SMALLINT only in the presence of maxInclusive or maxExclusive facets.
- Limitations in length may occur when converting XML anyURI, NOTATION, QName into SQL VARCHAR(100).

These problems are resolved in SQXML by using reasonable cardinalities in corresponding SQL types (such as INTEGER, SMALLINT, CLOB, VARCHAR, or CHAR). That is, if corresponding facets are specified in the first three cases, the conversion of XML types to SQL types is lossless. In case no facets are defined, as well as in the last conversion case, limitations in length may occur.

Now we focus on the XML Schema modeling concepts that are not available in SQL and therefore cannot be directly transformed from SQXML to SQL.

While the XML concept *derivation by extension*, which represents the commonly known inheritance concept, has an equivalent in SQL, the concept *derivation by restriction* is not available in SQL. It defines the derived data type as a logical restriction of the base type. SQXML copes with this problem by cutting the inheritance hierarchy. As the concept derivation by restriction is not often used in real-world examples and the consequences of breaking the inheritance hierarchy in these cases depend on the application, we can cope with this limitation without loss of information.

XML elements *with mixed content* are mapped to SQL by adding VARCHAR columns to the affected tables. As the length of the character strings in XML mixed content is not restricted, the SQL columns created for the mixed content are set to the maximum length of VARCHAR (depending on the implementation).

Regarding the conversion of *ModelGroups* from SQXML to SQL, the SQXML *ModelGroup* is a unification of XML Schema's *ModelGroups* that allow for defining 'sequence', 'choice', and 'all' groups of XML elements and of *Particles* that can specify the minimum or maximum occurrence of an element or a *ModelGroup*.

As there is no support for attributes with unbounded maximum number of occurrences in SQL, a new table is created for them during view generation. I.e., whenever an SQXML *Entity* may occur more than once, an additional table connected by a *unique key/foreign key* relationship is created.

In order to reflect the minimum and maximum occurrences as well as 'choice' and 'all' model groups, special constraints are added.

Below we consider the relevant SQL constraints that ensure exact reproduction of occurrence restrictions.

- If the minimum occurrence number is one, then NOT NULL constraints are added to the respective column.
- If the minimum number of occurrences is *n*, a table-level constraint is added:

```
CONSTRAINT minOccurs CHECK (NOT EXISTS
(SELECT * FROM table GROUP BY column
HAVING COUNT(*)<n))
```

- If the maximum constraint is one, no constraint is needed in SQL.
- If the maximum number of occurrences is *n*, a table-level constraint is added:

```
CONSTRAINT maxOccurs CHECK (NOT EXISTS
(SELECT * FROM table GROUP BY column
HAVING COUNT(*)>n))
```

To ensure correct representation of a 'choice' group in SQL, a choice constraint is added specifying that exactly one of the entities of the model group must be not-null. For two columns this constraint is as follows:

```
CONSTRAINT choice CHECK (
    ( column1 IS NULL AND column2 IS NOT NULL)
OR ( column1 IS NOT NULL AND column2 IS NULL))
```

In case of an 'all' group, the entities can either be all present or none of them. This is achieved by the following constraint, again shown for two columns:

```
CONSTRAINT all CHECK (
    column1 IS NOT NULL AND column2 IS NOT NULL)
        OR (column1 IS NULL AND column2 IS NULL) )
```

Thus the conversion of *ModelGroups* from SQXML to SQL proceeds without loss of information as well.

Summing up, we can learn from the discussions of this section that all transformations performed in SQXML's homogenization, integration and view generation steps are lossless in the sense that each local schema element will be represented in both global views which therefore are complete. Furthermore, the special property of the SQXML metamodel that each SQL as well as each XML concept relates to exactly one at least equally expressive SQXML concept together with the described usage of context information ensure unambiguousness of global views. Especially based on the observed completeness property, we can claim that the global views created by SQXML for a given set of local SQL and XML sources are semantically equivalent in the sense that they represent the same information content, which is the entire amount of information contained in all local data sources. We consider this semantic equivalence property crucial since it ensures equal information access opportunities for both XML and SQL applications.

5 Conclusions

In contrast to diverse approaches supporting storage and processing of relational and XML data under a single data model and thus providing only a single interface (either SQL or XML), our research focuses on the development of a new approach aimed at integrated processing of these data formats and providing access to the entire integrated information via both query languages, SQL and XQuery. To achieve this, our SQXML Integration System performs a schema integration process that generates a global schema and finally represents the complete amount of integrated information in both formats, as an (object-) relational and an XML schema.

We have shown that the global views constructed during the SQXML integration process – in both SQL and XML representations – are complete. That is, each local schema element is represented in both global views. Moreover, semantic equivalence of the global views ensures both for global SQL and XML applications equal access to the same entire information stored in the local data sources.

As the primary challenge for our future research we see a dynamic environment, where data sources can leave and enter the system as well as local source schemas can be changed.

6 References

Haas, L.M., Hernandez, M.A., Ho, H., Popa, L., Roth, M. (2005): Clio Grows Up: From Prototype to Industrial Tool. *SIGMOD, Baltimore, USA.*

Haerder, T., Sauter, G., Thomas, J. (1999): The Intrinsic Problems of Structural Heterogeneity and an Approach to their Solution. *The VLDB Journal, (8).*

Information technology-Database languages-SQL-Part 1: Framework (SQL/Framework), ANSI/ISO/IEC 9075-1:2003.

International Organization for Standardization. ISO/IEC 9075-14:2005: Information Technology, Database Language SQL; Part 14: XML Related Specifications (SQL/XML).

Kozlova, I. (2005): SQXML: Integrated Processing of Information Stored in Object-Relational and Native XML Databases. *Proc. of International Conference on Information Integration and Web-based Applications & Services (iiWAS).*

Kozlova, I., Husemann, M., Ritter, N., Witt, S., Haenikel, N. (2005): CWM-based Integration of XML Documents and object-relational Data. *Proc. of International Conference on Enterprise Information Systems (ICEIS), USA.*

Kozlova, I., Ritter, N., Reimer, O. (2006): Towards Integrated Query Processing for Object-Relational and XML Data Sources. *Proc. of International Database Engineering & Applications Symposium (IDEAS).*

Kozlova, I. and Ritter, N. (2006): An approach to unification of XML and object-relational data models. *Proc. of International Conference on Information Integration and Web-based Applications & Services (iiWAS).*

OMG: Common Warehouse Metamodel (CWM) Specification, Vers. 1.1 Vol. 1. Object Management Group (OMG) Specification (2003).

Wiederhold, G. (1992): Mediators in the architecture of future information systems. *IEEE Computer, 25(3).*

W3C, XML Schema Part 1: Structures. W3C Recommendation (2006).

W3C, XML Schema Part 2: Datatypes. W3C Recommendation (2006).