CWM-BASED INTEGRATION OF XML DOCUMENTS AND OBJECT-RELATIONAL DATA

Iryna Kozlova, Martin Husemann, Norbert Ritter, Stefan Witt, Natalia Haenikel
Distributed Systems and Information Systems, University of Hamburg, Germany

Keywords: Schema integration, XML Schema, SQL:1999, metamodel, CWM

Abstract: In today's networked world, a plenitude of data is spread across a variety of data sources with different data models and structures. In order to leverage the potential of distributed data, effective methods for the integrated utilization of heterogeneous data sources are required. In this paper, we propose a model for the integration of the two predominant types of data sources, (object-)relational and XML databases. It employs the Object Management Group's Common Warehouse Metamodel to resolve structural heterogeneity and aims at an extensively automatic integration process. Users are presented with an SQL view and an XML view on the global schema and can thus access the integrated data sources via both native query languages, SQL and XQuery.

1 INTRODUCTION

Nowadays, relational and object-relational database management systems are the most widespread ones in practical use while so-called native XML DBMSs (e.g., Tamino (Tamino, 2004)) gain popularity since they provide means for storage and management of XML documents in a native format. The looming parallel existence of two popular types of database management systems calls for means to effectively integrate (object-)relational and XML data sources.

This can be achieved by introducing a middleware layer that provides a uniform interface for querying and updating both XML and (object-)relational data in their local data sources. The key idea here is a *schema integration* process that generates a single global schema comprising the entire information about the local data sources. During this process, conflicts arising from structural and semantic heterogeneity must be resolved.

We introduce an integration middleware called *SQXML*. The name represents the focus on the two data models and their unification into a common metamodel. SQXML provides a number of unique features for efficient handling of information stored in (object-)relational and XML data sources, most prominently the provision of two views on the integrated data sources, an SQL view and an XML view, allowing users to view the entire information in their preferred format. The integrated data can be accessed via both native query languages, SQL and XQuery, while the local data sources remain unchanged.

In this paper, we focus on the SQXML approach to generating the global schema. This novel approach is based on the *Common Warehouse Metamodel* (CWM) (OMG, 2003), which has been used to create a CWM-based SQXML metamodel unifying the object-relational (SQL:1999) and XML Schema metamodels. The process of creating the global schema is conducted almost automatically, requiring user interaction only during the schema matching phase in order to improve the resolution of semantic conflicts.

The paper is organized as follows: Section 2 presents a sample integration scenario. An overview of the SQXML system is given in Section 3. Section 4 delineates our solution to the structural heterogeneity problem, introducing the SQXML metamodel that unifies the concepts of SQL and XML. Section 5 describes the approach to resolving the semantic heterogeneity between the local SQXML schemas and to creating the global schema. In Section 6, the conversion process that transforms the global SQXML schema into the global SQL:1999 and XML Schema representations is proposed. Section 7 gives a survey of existing related integration approaches, and section 8 concludes the paper.

Kozlova I., Husemann M., Ritter N., Witt S. and Haenikel N. (2005).
CWM-BASED INTEGRATION OF XML DOCUMENTS AND OBJECT-RELATIONAL DATA.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 35-43
DOI: 10.5220/0002523300350043

2 MOTIVATING EXAMPLE

In this section, a simple example is given to illustrate the SQXML integration approach. It will be used as a running example throughout the paper. For the sake of simplicity, we consider only one local object-relational data source and one local XML data source – in general, the goal of SQXML is to operate on more than two local sources.

Suppose there are two data sources, one (object-) relational and the other XML-based, both storing information about books (see local schemas in Figure 1). The local relational schema contains the tables `books` and `authors`, which hold titles, publishers and author names of books on the topics 'Java' and 'XML'. In contrast, the local XML source stores information on Java books only, and its XML Schema definition contains a complex type `javabook` that consists of elements `title`, `author`, `price`, and `publisher`.

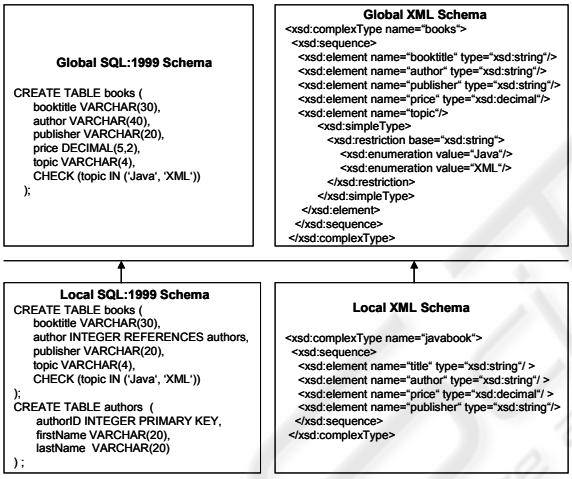

Figure 1: Sample local and global schemas

Beside the metamodel conflicts between SQL:1999 and XML Schema, the following representation conflicts between the local schemas are obvious: First, there is a conflict of *missing attributes* because `price` is missing in the relational schema and `topic` in the XML Schema definition. Second, there are an *entity-versus-attribute* conflict and an *attribute concatenation* conflict regarding the information about authors.

Such structural and semantic conflicts are resolved by the SQXML integration process resulting in two semantically equivalent global schemas, as those shown in Figure 1. The SQXML run-time component accepts SQL as well as XQuery queries and processes them by query splitting and result synthesis, as illustrated in Figure 2.

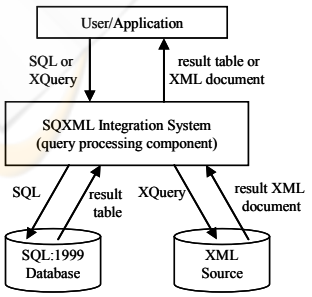

Figure 2: Query processing

3 SQXML OVERVIEW

The major components of the SQXML Integration System are illustrated in Figure 3. At the *Data Level* we consider SQL:1999 databases and XML data sources (providing XML schemas and XQuery support). The *User/Application Level* at the top represents two views on the integrated data, an SQL-view available to SQL users/applications and an XML-view available to XML users/applications.

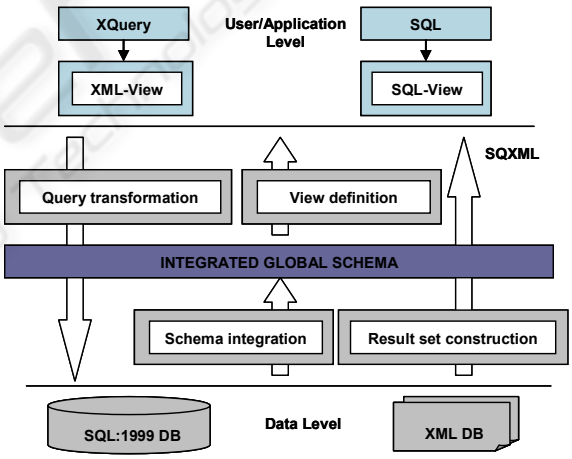

Figure 3: SQXML System

The layer between the User/Application Level and the Data Level contains the actual SQXML Integration System, which is an integration middleware layer. It comprises the schema integration component (in charge of creating the integrated global schema), the view definition component (in charge of creating XML and SQL views), and the query transformation and result set construction components (in charge of evaluating global user queries). In this paper, we focus on the process of creating the global schema.

4 RESOLVING STRUCTURAL HETEROGENEITY

SQL:1999 and XML Schema use different constructs to represent schemas. The main idea to resolve this structural heterogeneity is to unify SQL and XML concepts into a single set of concepts. This idea has been implemented in a new metamodel called SQXML, representing a superset of both models. The SQXML metamodel is used as a common data model to achieve a structurally homogeneous representation of the local SQL and XML schemas. Represented in terms of the SQXML metamodel, the local schemas can then be used in the actual process of creating the integrated global schema. The uniform representation of the local schemas facilitates the automation of the integration process as automatic schema-matching and schema-merging techniques can be used. The resulting global SQXML schema comprises the complete information stored in the local data sources. In the final phase of the process, the global schema is converted into the data sources' native representations, creating two semantically equivalent global schemas in terms of SQL:1999 and XML Schema that are presented to the user. This allows the user to formulate queries in SQL or XQuery.

For building the SQXML metamodel we used the *Common Warehouse Metamodel* (*CWM*) (OMG, 2003). CWM is a metamodel of a generic data warehouse architecture. Its original purpose is to standardize the exchange of models between data warehousing applications and repositories in distributed, heterogeneous environments. Thus, CWM offers a way to represent schemas, i.e., the models, of heterogeneous information sources. CWM consists of multiple components. The *Resource* component includes packages for metamodels of object-oriented, relational, record, multidimensional, and XML data resources. The *Relational* and *XML* packages are of special interest for integration in our case.

There are a number of reasons to use CWM for the unification of SQL:1999 and XML Schema. Most importantly, CWM defines a standardized way to represent heterogeneous models and a standardized terminology for modelling. With the Relational and the XML package it already provides a metamodel for SQL:1999 and XML DTD, which can be used as a starting point for the unification. Note that there is no package for XML Schema in CWM 1.2. Therefore we propose (Section 4.2) a prototype of a CWM XML Schema Definition (CWM XSD).

The model hierarchies of SQL:1999, XML Schema, and SQXML, according to the OMG metadata architecture (OMG; 2003), are shown in Table 1.

Table 1: Metamodel hierarchy

Meta-Level	SQL:1999	XML Schema	SQXML
M3: Meta-Metamodel	MOF	MOF	MOF
M2: Metamodel	CWM:Relational Package	CWM:XML Schema	SQXML
M1: Model (Schema)	Database Schema	XML Schema Definition	SQXML Schema
M0: Instances (Data)	Database	XML Documents	Integrated Data

As illustrated in Figure 4, SQL:1999 database schemas and XML Schema definitions are located at level M1 and their metamodels at level M2. The SQXML metamodel was constructed through a *unification process* conducted at level M2. For each modelling concept of SQL:1999 and XML Schema there is a corresponding concept in the SQXML metamodel. For example, SQL's *Table* and *SQLStructuredType* and XML Schema's *ComplexTypeDefinition* have been unified into SQXML's *Entity* concept (see detailed discussion in Section 4.3). SQXML uses the data types from XML Schema; the SQL:1999 data types have been converted to XML Schema simple types using the

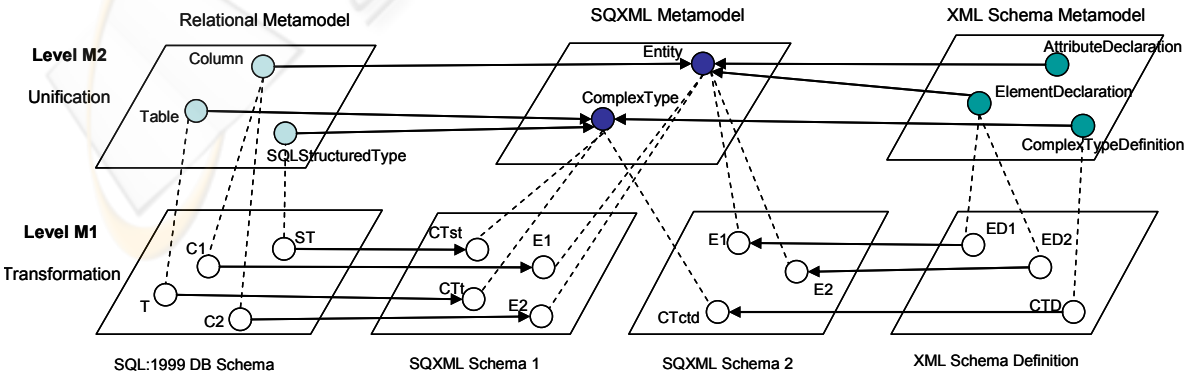

Figure 4: Metamodel unification and schema transformation

SQL/XML standard (ISO/IEC, 2003).

The *transformation process* conducted at level M1 proceeds as follows: Given an SQL database schema and an XML Schema definition, each of them is transformed into an equivalent SQXML schema, depicted in Figure 4 as Schema 1 and Schema 2, correspondingly.

The overall process of creating the global schemas is illustrated in Figure 5. The first step in this process is the transformation step, as discussed above. The further steps, namely matching and merging, are discussed in Section 5, and the concluding conversion step is discussed later in Section 6.

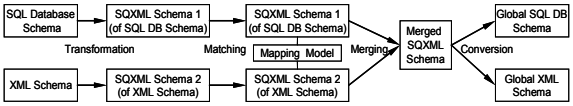

Figure 5: Process of creating the global schema

4.1 CWM Relational Metamodel

The CWM standard includes the package *Relational,* which is a metamodel for SQL:1999 database schemas (OMG, 2003). The complete UML diagram of the metamodel as well as all details on the package dependencies can be found in (OMG, 2003).

CWM Relational does not contain support for some of the advanced features of SQL:1999, like privileges and access control. SQXML supports most features of Core SQL:1999 and offers basic object-relational support, which is not a part of Core SQL. Since not all features of SQL are needed for data source integration and some simplifications facilitate the integration, the following simplifications were applied to the CWM Relational package for our studies:

- *Catalogs* are not supported, as we consider one SQL and one XML schema.
- *SQLIndices* have been left out because they are not part of SQL:1999.
- *Views* are not supported. The inheritance hierarchy of *ColumnSet*, *NamedColumnSet*, *QueryColumnSets* and *Table* collapses into a single class *Table*.
- *Triggers* have been omitted as they are not a part of Core SQL.

Figure 6a illustrates (simplified for the sake of readability) the central part of the CWM Relational metamodel as it is used in the SQXML integration system.

4.2 CWM XML Metamodel

We developed a CWM XML Schema metamodel (CWM XSD) based on the XML Schema recommendation (W3C, 2001a, 2001b). Due to space restrictions, here we only describe its simplified version as it is used in the SQXML integration system. In comparison to the complete version, XML namespaces and the following classes have been omitted: *Annotation* and *Notation-Declaration*, *WildCard* and *AttributeWildcard*.

Figure 6b shows that the top-level container of an XML Schema definition is the *Schema* class. It contains global type definitions, global attribute and element declarations as well as global modelling

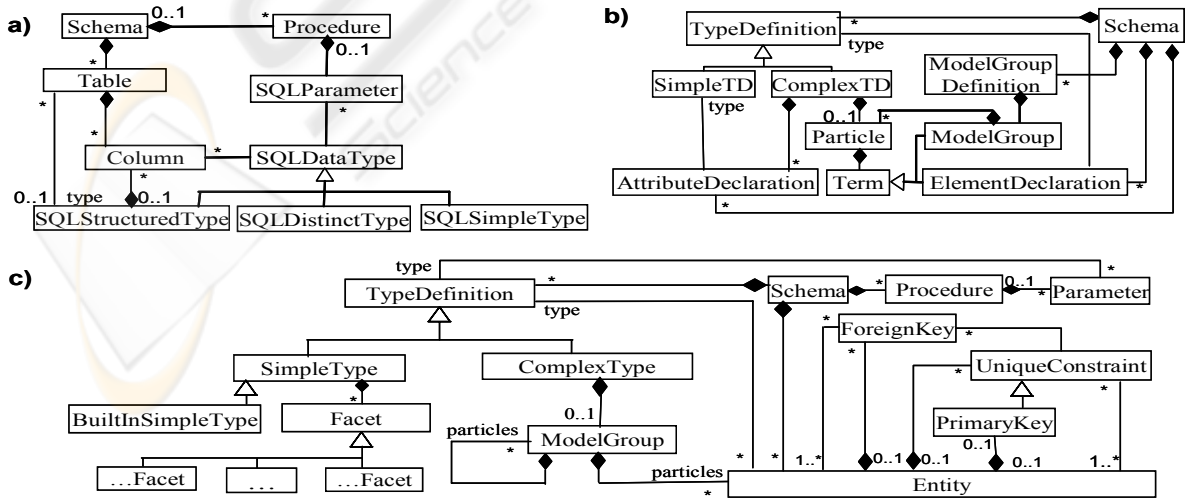

Figure 6: Simplified UML diagrams of:
a) CWM:Relational metamodel, b) XML Schema metamodel, c) SQXML metamodel

groups for elements and attributes. *TypeDefinition* is the central class for the definition of a type hierarchy. It is an abstract class which is specialized by *SimpleTypeDefinition* and *Complex-TypeDefinition*.

The class *SimpleTypeDefinition* has been defined according to the XML Schema Definition Part 2 (W3C, 2001b): Each simple type is either built-in or user-defined. A simple type is constrained by a set of facets. These include the fundamental facets and constraining facets, which are instances of the *Facet* class (not shown in Figure 6b).

A *ComplexTypeDefinition* consists of *AttributeDeclarations* and *ElementDeclarations*, which can be combined using supplemental schema components. As shown in Figure 6b, a *ComplexTypeDefinition* cannot directly contain *ElementDeclarations*, but must use a *ModelGroup* ("sequence", "choice", or "all") that contains *ElementDeclarations*. It is modelled as follows: A *ComplexTypeDefinition* contains a *Particle*; the *Particle* must contain a *ModelGroup*, which is a special kind of *Term*; a *ModelGroup* may then contain other *ModelGroups* or *ElementDeclarations*.

Finally, *ElementDeclarations* can be constrained by *IdentityConstraintDefinitions*, which are the key constraints in XML Schema. The *Identity-ConstraintCategory* attribute indicates the constraint category: *key* specifies a primary key, *keyref* a foreign key, and *unique* a unique constraint (not shown in Figure 6b).

4.3 CWM SQXML Metamodel

With our approach, the essential step to resolve the structural heterogeneity between SQL:1999 and XML Schema was to unify the SQL:1999 metamodel (Section 4.1) and the XML Schema metamodel (Section 4.2) into a single SQXML metamodel. Its simplified class diagram is depicted in Figure 6c.

The top-level container of SQXML is *Schema*. A schema can contain *TypeDefinitions*, *Entities* and *Procedures*. The *Procedure* class is the same as in SQL because there are no routines in XML Schema. The *TypeDefinition* class is used for modelling the type hierarchies of SQL and XML. *TypeDefinition* is an abstract class, which can be either *SimpleType* or *ComplexType*.

The *SimpleType* class unifies the *SimpleTypeDefinition* of XML Schema with *SQLSimpleType* and *SQLDistinctType* of SQL. It is entirely the same as *SimpleTypeDefinition* of the XML Schema metamodel for two reasons: First, it is trivial to map XML Schema data types to SQXML data types (Section 4), and second, the mapping from SQL to SQXML data types can be performed using the SQL/XML standard (ISO/IEC, 2003) (a detailed discussion is omitted here due to space restrictions).

The *ComplexType* class is a unification of structured types: It unifies XML Schema's *ComplexTypeDefinition* with *Table* and *SQLStructuredType* of SQL (see also Figure 4). In SQL:1999, tables and structured user-defined types are similar enough to be unified. The information from what an SQXML *ComplexType* originates is stored in an additional *Context* class (not shown in Fig. 6c) that is associated with the *Entity* class and used during the conversion of the global SQXML schema to SQL and XML Schema in the last step of the integration process. The content of a *ComplexType* is always a *ModelGroup*. This reflects the modelling concept of XML Schema, which requires a *ComplexTypeDefinition* to contain a *Particle* containing a *ModelGroup*.

The class *Entity* unifies SQL's *Columns* with XML Schema's *ElementDeclaration* and *AttributeDeclaration* (see also Figure 4). The latter two classes are similar enough to be unified into a single class, and the origin of each of them is stored in the *Context* class.

The classes *ForeignKey*, *UniqueConstraint*, *PrimaryKey*, and *CheckConstraint* of SQL and the *IdentityConstraintDefinition* of XML Schema provide the same concepts in a different syntactical representation, so it is straightforward to convert the XML Schema concepts to SQL, i.e., *key* from *IdentityConstraintDefinition* becomes *PrimaryKey*, *keyref* becomes *ForeignKey*, and *unique* becomes *UniqueConstraint*.

To construct the SQXML metamodel, we have considered the modelling concepts of SQL:1999 and XML Schema as well as the structures (i.e., classes and relationships) of the CWM Relational and CWM XML Schema metamodels. Thus, each supported SQL:1999 and XML Schema concept is also available in SQXML and can be expressed in terms of the SQXML metamodel.

5 RESOLVING SEMANTIC HETEROGENEITY

The structural heterogeneity between SQL:1999 and XML Schema has been resolved by the unification approach described in Section 4. After the transformation step (Figure 5), the local SQL and XML schemas are represented in terms of the SQXML metamodel, later on referred to as SQXML Schema 1 and SQXML Schema 2. As shown in Figure 5, the next steps of creating the global

schema are schema matching and schema merging to resolve the semantic heterogeneity between the local schemas. The goal of schema matching is to find how the schema elements of different schemas correspond to each other and to identify representation conflicts between the schemas. Using these correspondences, the two local SQXML schemas are merged, resulting in the global SQXML schema.

5.1 Schema Matching

The traditional way to obtain a mapping model is having a domain expert create it manually, sometimes with the help of tools providing visual editing.

The desirable alternative is to conduct automatic schema matching. Different schema matching algorithms have recently been developed. A comparison and classification of their performance can be found in (Rahm and Bernstein, 2001). Based on these results, the Cupid algorithm (Madhavan et al, 2001) has been chosen as automatic schema matcher for SQXML. The matching process in Cupid is both linguistic-based and constraint-based: It compares elements by discovering element name and data type similarities and analyzing the positions of the elements in the schema. The Cupid algorithm takes two schemas **S1** and **S2** as input and returns a *mapping*. A mapping is defined as a set of *mapping elements*, each of which indicates that certain elements of schema **S1** correspond to certain elements of **S2** (Rahm and Bernstein, 2001).

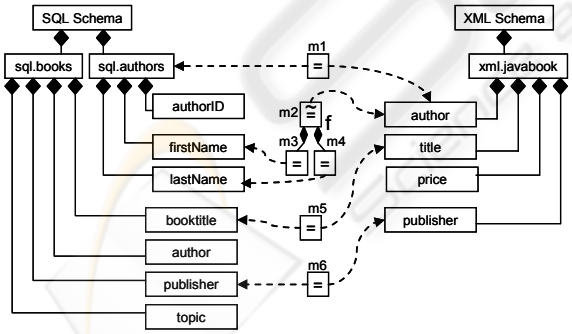

Figure 7: Sample (local) SQXML schemas and corresponding mapping model

The Cupid algorithm cannot identify and solve all representation conflicts. Thus, an enhancement of the mapping result is needed. For this, we use a notion of a *mapping model* (similar to (Pottinger and Bernstein, 2003)) that identifies (in addition to the corresponding schema elements detected by Cupid) containment relations between mapping elements, as illustrated in Figure 7: It shows the local schemas from our example (Section 2), represented as SQXML schemas, and the mapping model between them. There are 'equality' mapping elements, like **m5**, that indicate semantically equal elements, and 'similarity' mapping elements, that are represented using a *similarity mapping function*. In Figure 7, the 'similarity' mapping element **m2** and a concatenation function **f** solve the attribute concatenation conflict regarding the representation of the author name.

5.2 Schema Merging

The mapping model resulting from the schema matching is required for the schema merging phase. Schema merging is performed by a *merge operator* (Bernstein et al, 2000), which takes two schemas **S1** and **S2** and a mapping model **M** between **S1** and **S2** as input and produces the merged schema $\mathbf{S_M}$ as a result. It uses the correspondences between the schemas to merge equal or similar elements into single schema elements in the merged schema.

From the existing schema merging algorithms, the *Vanilla* algorithm has been chosen (Pottinger and Bernstein, 2003). SQXML uses an improved version of Vanilla, as discussed in the following.

One essential drawback of the Vanilla algorithm is that schema elements that are related by a 'similarity' mapping element remain separated in the merged schema. Our solution to this problem is to declare one of the schemas as 'preferable'. The 'similarity' mapping elements are then treated as 'equality' mapping elements, and the elements of the 'preferable' schema are taken over into the merged schema. During query processing, the similarity mapping function is applied to the instances of one local data source to display them according to the representation used in the global schema. In our example (Section 2), the element `author` is taken over into the global schema, so that during query processing the concatenation function is applied to the elements `firstName` and `lastName` to represent them as required by the global schema.

Another drawback of Vanilla is that it does not store information about the relationship between the local schemas and the global schema. This relationship, also called *semantic mapping*, is later on needed for query processing. Therefore, we propose to co-create the semantic mapping during the schema matching and schema merging steps when all the required information is available.

6 CONVERSION PROCESS

The integration steps described in sections 4 and 5 result in the integrated global schema, expressed in terms of the SQXML metamodel. As a last step, the global SQL:1999 schema and the global XML Schema definition are created through the *conversion* process, which is reverse to the transformation process.

Since the SQXML metamodel uses data types from XML Schema (Section 4) it is trivial to map SQXML data types to XML Schema data types. Converting SQXML types to SQL:1999 types is not that straightforward. Some XML types, e.g., the Gregorian Calendar types, have no direct correspondences in SQL:1999. For XML types with unbounded cardinality, such as integer and string, an exact mapping cannot be provided either. Only bounded subsets, i.e., integers with a maxInclusive or maxExclusive facet or subtypes of `integer` such as `int` or `long`, can be mapped exactly. In this case, `facets` in XML Schema are converted to appropriate `check` constraints in SQL:1999.

```
1 convertSchemaToSQL(Schema S)
2 begin
3  for each ComplexType t in S do
4       convertComplexType(type: t);
5  end for
6  for each global Entity e in S do
7   convertEntity(type:createTable(),
    entity:e, mixed:FALSE, alwaysNullable:FALSE);
8  end for
9  for each structured type t do
10     delete constraints from t;
11     if t is referenced only once
       or t.context is marked as table
       then eliminate t;
12 end for
13  create SQL schema from structured types,
    typed tables, tables, procedures;
14  output SQL schema;
15 end;
```

Figure 8: Conversion algorithm

The conversion algorithm, which takes an SQXML schema **S** as input and produces an SQL:1999 schema represented in terms of the CWM Relational metamodel, is outlined in Figure 8. The schema conversion is done using a top-down approach – the schema **S** is traversed starting from globally defined types and entities. First, all globally defined complex types are mapped to structured user-defined types (lines 3-5). Next, global entities are mapped to typed tables (lines 6-8). User-defined structured types may not have constraints in SQL:1999, so constraints are discarded (line 10). Furthermore, if there is only one typed table of a structured type or if there was no structured type in the original SQL schema (i.e., if it was originally a table in the local schema, according to the information stored in the Context class) the type is eliminated (line 11) because it is not required. The structured types, typed tables, tables, and procedures are glued together to an SQL:1999 schema (line 13).

The algorithms for the conversion of complex types, entities and procedures are not presented here due to space restrictions, as well as the conversion algorithm from SQXML schema to XML Schema definition.

The conversion process is also used for creating an SQL view of the local XML schema as well as an XML view of the local SQL schema, which are needed to perform query processing.

A global query (SQL or XQuery) submitted by the user to the SQXML middleware is formulated on the global schema (SQL:1999 or XML Schema respectively), so it must be split and reformulated in terms of the local schemas to access the data in the local sources (Figure 2). For this, the relationship between the global schema and the local schemas (the *semantic mapping*) is needed. SQXML uses the global-as-view (GAV) approach (Halevy, 2000), where the global schema is defined as a view on the local schemas. The GAV approach was chosen because that way the splitting of the global query into the local queries is a simple process of view unfolding. For the global SQL:1999 schema, the query plans are based on SQL, so that an SQL:1999 view on the local XML schema is needed to answer queries. Analogously, for the global XML Schema, an XML view on the local SQL schema has to be generated. This task is solved by applying the conversion process to each local SQXML schema.

7 RELATED APPROACHES

A wide spectrum of approaches related to our topic has been developed. First of all, the information integration systems developed for retrieving and managing data stored in heterogeneous sources are to be mentioned. Research and commercial approaches to the problem of information integration vary in the methods and techniques they use.

The wrapper-mediator approach, which is often used in integration systems, is based on the following idea: The data models of the local data sources are first converted to a common data model supported by the integration system. This translation is conducted by *wrappers*, that is, hard-coded. The integration rules are hard-coded as well and defined in *mediators*. The end user's view in this case is a schema that is provided by some mediator. Such approaches are quite good at resolving the

heterogeneity problems; structural heterogeneity is resolved at the wrapper level and semantic heterogeneity is homogenized by the mediators. Examples of such approaches are the TSIMMIS (Garcia-Molina et al, 1997) and Florid (Ludäscher et al, 1998) integration systems. TSIMMIS implements *virtual* (or *logical*) integration, meaning that the data stays in the sources and is delivered to the user on request only. Florid follows the *materialized integration* approach: the data from the local sources is integrated and materialized so that the global query is directly evaluated on the integrated set of data.

The Garlic (Carey et al, 1995) integration architecture is similar to those described above, with the difference that no mediators are used. Instead, the integration and query transformation are performed centralized by the 'Query Services and Runtime System' component.

Another strategy is to develop a special mapping language. Such languages, like BRIITY (Härder et al, 1999), allow the definition of mapping rules which, in turn, determine the interoperability between the global schema and the local schemas. Heterogeneity conflicts can be solved explicitly by coding appropriate integration rules.

Recently, various data integration strategies have been developed for the interoperability of XML and RDBMSs. They focus on using a relational database management system to store and query XML data: Either an RDBMS is used to store and query XML data, or existing relational data is presented as an XML view to the user or application. Commercial solutions used by object-relational database management systems (as Oracle 9i (Higgins et al, 2002), IBM DB2 (IBM Corporation, 2002), Microsoft SQL Server 2000 (Microsoft Corporation, 2004)) provide various mechanisms for mappings between relational tables and XML fragments, but they do not provide schema integration: The user still has to know both schema definitions (no global schema is created), to use two query languages, and to perform combining and cleaning of query results manually.

Among the various research approaches for XML Publishing, we mention SilkRoute, XPeranto and Agora here. SilkRoute (Fernandez et al, 2000) and XPeranto (Shanmugasundaram et al, 2001) focus on defining XML views on relational data and evaluating XML queries by decomposing the view. In both approaches, a virtual XML view is created and then the XML queries (XML-QL in SilkRoute and XQuery in XPeranto) are evaluated on this view. These approaches use only a single local relational data source, and their main task is to process XML queries on it.

The Agora (Manolescu et al, 2001) approach focuses on the problem of translating XQuery queries into SQL. Unlike SilkRoute and XPeranto, it can handle relational as well as XML data sources. In contrast to SQXML, Agora uses the local-as-view (LAV) approach (Halevy, 2000) and supports only one language, XQuery.

Integration solutions like TSIMMIS, Garlic, and BRIITY are more generic with respect to the data sources that can be integrated, and considerable efforts would be required to adapt these approaches to support SQL:1999 and XML Schema. Also, considerable programming efforts would be required to code wrappers and mediators or to define the mapping rules.

In contrast, the SQXML approach resolves the structural heterogeneity between SQL:1999 and XML Schema fully automatically: With the SQXML metamodel (Section 4), SQL:1999 schemas and XML Schema definitions can be directly transformed into uniform representations. The semantic heterogeneity is resolved in a near-automatic way, only possibly requiring some manual changes and improvements to the mapping model during the schema matching process.

The SQXML system is aimed at providing the user or the application with bilingual access, i.e., it supports both query languages, SQL and XQuery. In contrast, related approaches define a new language (e.g., Lorel in TSIMMIS or F-Logik in Florid) or use SQL with appropriate extensions (e.g., object-oriented extensions of SQL in Garlic) to provide access to the integrated data.

None of the integration systems mentioned above supports more than one query language, and most of the approaches require significant user support during the integration process. The SQXML Integration System as proposed in this paper is aimed at simplifying and automating the integration process as well as providing efficient data access.

8 CONCLUSIONS AND FUTURE WORK

This paper has presented SQXML, a system designed to implement the integration of XML and (object-)relational data sources. SQXML provides new features that have not been available in other integration systems. It aims at providing near-automatic performance, that is, user interaction is limited to the process of resolving semantic conflicts between the schemas. Structural heterogeneity between the schemas is resolved fully automatically.

To unify SQL:1999 and XML Schema, concepts of the Common Warehouse Metamodel have been

used. A new CWM metamodel for XML Schema has been developed, and based on the CWM Relational and XML Schema metamodels, the SQXML metamodel has been constructed. The problem of overcoming the structural heterogeneity has been solved by using the CWM and SQL/XML standards to transform SQL and XML schemas to a unified representation in terms of the SQXML metamodel. This technique allows for the automated resolution of structural conflicts at the data model level.

Solutions to the problems of matching and merging two SQXML schemas have been proposed. An enhanced Cupid matching algorithm is used to find correspondences between the schemas, and an improved Vanilla algorithm is used for schema merging.

An approach for converting an SQXML schema to SQL:1999 and to XML Schema has been presented. Applied to the global SQXML schema, it results in the global SQL and XML schemas, concluding the schema integration process and allowing access to the integrated information in both local sources' query languages – SQL and XQuery.

Future work is aimed at enhancing the query processing to support updates of the local data sources through the global schema. Further aspects we plan to investigate are *data cleaning* and *instance-level integration*. Data cleaning deals with the problem of handling inconsistencies between the local data sources, e.g., data entries that refer to the same real-world object, but contain contradictory values. Instance-level integration is concerned with the matching and integrated processing of local data entries that contain different aspects of the same real-world object.

REFERENCES

Bernstein, P. A. et al., 2000: A Vision for Management of Complex Models. In: SIGMOD Record, Vol. 29, pp. 55-63.

Carey, M. et al., 1995: Towards heterogeneous multimedia information systems: The Garlic approach. In: Proc. of the 5th International Workshop on Research Issues in Data Engineering, pp. 124-131.

IBM Corporation, 2002: XML Extender Administration and Programming, Version 8. ftp://ftp.software.ibm.com/ps/products/db2/info/vr8/pdf/letter/db2sxe80.pdf

Fernandez, M. et al., 2000: SilkRoute: Trading between relations and XML. In: WWW9/Computer Networks, 33(16), pp. 723-745.

Garcia-Molina, H. et al., 1997: The TSIMMIS Approach to Mediation: Data Models and Languages. In: Journal of Intelligent Information Systems, vol. 8, pp. 117-132.

Halevy, A., 2000: Logic-based techniques in data integration. In: Logic Based Artificial Intelligence, pp. 575-595.

Härder, T. et al., 1999: The intrinsic problems of structural heterogeneity and an approach to their solution. In: The VLDB Journal, 8, pp. 25-43.

Higgins, S. et al., 2002: Oracle 9i XML Developers Guide – Oracle XML DB, Release 2 (9.2). Oracle Corporation.

International Organization for Standardization. ISO/IEC 9075-14:2003: Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML).

Ludäscher, B. et al., 1998: Managing Semistructured data with FLORID: A Deductive Object-Oriented Perspective. In: Information Systems, 23(8), pp. 589-612.

Madhavan, J. et al., 2001: Generic Schema Matching with Cupid. In: Proceedings of the 27th VLDB Conference, Roma, Italy, pp. 49-58.

Microsoft Corporation, 2004: SQL Server 2000 Product Documentation. http//www.microsoft.com/sql/techinfo/productdoc/2000/books.asp

Manolescu, I. et al., 2001: Answering XML queries over heterogeneous data sources. In: Proc. of the 27th VLDB Conf., Roma, Italy.

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

Pottinger, R. A., Bernstein, P. A., 2003: Merging Models Based on Given Correspondences. In: Proc. of the 29th VLDB Conference, Berlin, Germany, pp. 862-873.

Rahm, E., Bernstein, P. A., 2001: A Survey of Approaches to Automatic Schema Matching. In: VLDB Journal 10(4), pp. 334-350.

Shanmugasundaram, J. et al., 2001: Querying XML views of relational data. In: Proc. of the 27th VLDB Conference, Roma, Italy.

Tamino XML Server, Software AG, 2004: http://www2.softwareag.com/corporate/products/tamino/default.asp

W3C, 2001a: XML Schema Part 1: Structures. W3C Recommendation.

W3C, 2001b: XML Schema Part 2: Datatypes. W3C Recommendations.