ISSN 0361-7688, Programming and Computer Software, 2008, Vol. 34, No. 1, pp. 16–26. © Pleiades Publishing, Ltd., 2008.
Original Russian Text © A.A. Simanovsky, 2008, published in Programmirovanie, 2008, Vol. 34, No. 1.

Data Schema Evolution Support in XML-Relational Database Systems

A. A. Simanovsky

IntelliJ Labs, LLC, ul. Kantemirovskaya 2a, St. Petersburg, 197342 Russia

E-mail: asimanovsky@yandex.ru

Received June 28, 2006

Abstract—Many XML-relational systems, i.e., the systems that use an XML schema as an external schema and a relational schema as an internal schema of the data application representation level, require modifications of the data schemas in the course of time. Schema evolution is one of the ways to support schema modifications for the application at the DBMS level. A number of schema evolution support systems for different data models have been suggested. Schema evolution can be applied to mapping-related evolving schemas (such as schemas of XML-relational systems), the transformation problem for which is also known as schema adaptation. In this paper, a survey of various approaches to solving the outlined problems is given.

DOI: 10.1134/S0361768808010039

1. INTRODUCTION

Many modern applications working with databases, such as electronic systems of medical records [1] and CAD systems, require modification of the data stored, as well as updating of the data schemas, in the course of time. The process of data schema modifications in such systems is an essential part of the application lifespan [2]. Thus, the support of modifications of the conceptual (and, possibly, internal) schema of the data stored is an important task for the DBMS manufacturers, since it makes the development of external schemas for the applications easier.

The applications under consideration often use XML and query languages, like XQuery [3], as an interface to the data representation level, which, in turn, uses powerful industrial relational DBMSs as data storages implementing the logic of transformation of work with the XML data model to the work with the relational model [4–6] and, thus, representing XML-relational systems. Hence, we arrive at the problem of modifying the pair of mapping-related XML and relational schemas.

The schema evolution and schema versioning support systems [7] represent one of the solutions to the schema modification problem by means of DBMS. They regulate schema modification activities and allow one to describe semantics of changes of the application domain and reflect these changes in the database extending them to the schema of the data stored and the data themselves. They allow one to avoid development of *ad-hoc* algorithms for updating the schemas and the data with every modification. Currently, there are schema evolution support systems for relational [8–14], object-oriented [15–30], and XML [31–34] data models.

Schema evolution (rather than exclusively schema integration) can be applied also to describing modifications of mapping-related schemas [35]. The goal of this paper is to substantiate this point by surveying the existing schema evolution models and mappings of XML data to relational data.

2. SCHEMAS AND DATA MODELS

Before turning to description of existing schema evolution support systems, we introduce a number of concepts used in the analysis of different systems. First of all, we discuss schemas and data models. Applications work with data using a certain data schema.

A *data schema* (or simply schema) is metainformation describing the structure of the data stored and, possibly, operations on them.

A *data model* is a description of a class of data schemas allowing to express the structure, integrity rules, and languages for accessing the data satisfying schemas from this class.

Currently, relational DBMSs are most often used. They rely on the relational data model proposed in [36, 37].

The *relational model*[1] is determined by the following five components:

• a family of scalar types, including the Boolean type;

• a generator of relational types with fixed interpretation of relational values of given types;

[1] Following [38], we consider the definition of the relational model that is most convenient for studying temporal data, which are discussed below.

• a method for determining relational variables of relational types;

• an operator for assigning values to relational variables;

• a family of relational operators for deriving relational values from other relational values.

Further details on these components can be found in [38–40]. In traditional implementations for industrial DBMSs, these components are implemented by the following entities:

• scalar types `boolean`, `integer`, `varchar(N: integer)`, and so on;

• a generator of relational types specifying a type as an ordered product of scalar types and a set of names for them; an interpretation is a predicate the preimage of the "true" value for which is tuples occurring in the relational value;

• relational variables that are determined by the name and can have primary and foreign keys;

• assignment operators the role of which is played by the modifying SQL operators (`UPDATE`, `DELETE`, and the like);

• relational operators, including the selection, projection, Cartesian product, join, set-theoretical operations, and their SQL analogues (`SELECT`, `JOIN`, `UNION`, `INTERSECT`, and `EXCEPT`).

A *relational schema* is a set of relational variables. When writing down a relational schema, sets of functional dependencies and other properties of relational variables are often separated.

Instead of "relation/relational variable," "attribute," and "domain," the terms "table," "column," and "column type" can be used. This is convenient when components of relational and XML schemas are mentioned simultaneously.

A number of data models have been suggested for XML (see, for example, [41]). The most difficult task for the researcher is to select a query language and means for describing schema constraints, which resulted in a great number of standards (XPath, XQuery, DTD, XML-Schema, and so on [42]). Currently, several XML models, e.g., XDM (XQuery and XPath Data Model [43]), are standardized and widely used. However, a number of issues, especially algebraic semantics of XML, are open research are [44]. In the proposed definition, we separate one aspect, namely, the property of weak structuredness and self-description of XML data: interpretation of data depends on the application and on values of data.

An *XML data model* is determined by the following components:

• a string type;

• a tag and attribute names alphabet generator;

• a DTD generator (the definition of DTD is given below);

• a method for determining XML documents satisfying a given DTD: an XML document corresponding to a given DTD is a hierarchy of tag names and associated pairs (attribute name and value) that is a word in the language determined by the given DTD and satisfies a condition imposed on the attribute names;

• an interpretation of an XML document satisfying a given DTD is a predicate defined on the Cartesian product of domains, where the domains are sets specified by the mapping from the set of tag names; this mapping is called a syntax interpretation of the application (or simply application interpretation);

• a language of queries to XML documents.

The *DTD* is a mapping from the alphabet of string tag names to the Cartesian product of a class of languages over a given alphabet and a set of the subsets of the alphabet attribute names.

In XML, all data is of the string type.[2] Their actual types (what is referred to as syntax interpretation of the application) are determined by the application domain. In this paper, we make an assumption on how a definition of an actual type is organized. We impose the following *hierarchy requirement* to the application interpretation. Let I be an application interpretation. Then,

$$I(t) = f(t_1, t_2, \ldots, t_k, a_1^1, a_1^2, \ldots, a_k^{N_k}), \qquad (1)$$

where t_i is a sequence of nested (in accordance with DTD) tags containing t and a_i^j are attributes corresponding (in accordance with DTD) to these tags.

The hierarchy requirement means that the application is capable of interpreting the attribute value having information on its name and the tag where it is located; and the interpretation of the tag value by the application is determined by the set of tags in which the given tag is nested, as well as by the values of its attributes and the attributes of the enclosing tags. Such an assumption agrees with informal recommendations on XML data design [46].

Currently, for XML query languages, XSLT, XPath, and XQuery are used. The distinctive feature of these languages is that the syntax form of a query does not depend on the syntax interpretation of the application (this explains the use of the word "syntax" as applied to the application interpretation). On the other hand, a modification of the structure of an XML document results in the necessity of modifying the syntax of an XPath/XQuery query in order to preserve the result produced. Such modifications will be referred to as semantic modifications. Note that this terminology is, to some extent, opposite to the terminology that could be introduced from the standpoint of the application that uses the XML data model: for the application, the informa-

[2] Strictly speaking, this is true only for DTD. Other XML schema description facilities, e.g., XML-Schema [45], use other data types. In the context of the above definition, these data types can be viewed as a partial syntax interpretation.

tion on types is semantic, whereas the structure relates to the syntax. Nevertheless, the structural information may bear semantic load from the application standpoint too.

When describing modifications of data and data schemas, the data model (relational, XML, or some other) is to be extended by facilities for expressing temporal nature of the data and schemas. The temporal nature of data is considered in studies devoted to temporal databases.

Informally, the temporal database can be defined as a database containing information history in addition to (or instead of) the actual data. Data warehouses [47] partially fulfill this role, whereas ordinary databases usually contain only current data (as long as the assertions formulated on the basis of some data become false, the data is deleted from the database). Thus, the distinctive feature of the temporal data model is that it includes a semantic concept of time. Let us give a definition of the temporal relational database.

A *temporal data type* is a type of data whose values possess semantics of time. *Temporal relational values* are relations for which at least one attribute of their tuples has temporal type. A *temporal relational variable* is a relational variable whose values are relational values. A *temporal database* is a database including temporal variables.

For a temporal data type, different data types, both discrete and continuous [38], were suggested. Currently, representation of time by an interval type with discrete values of the interval boundaries is considered to be preferable. The current time is usually represented by a special value, or the data that are true till some undetermined moment of time in the future are stored in semitemporal tables, in which time is represented by an ordinary numerical type [38].

There exist two types of time in databases, valid and transaction time.[3] The valid time is the time during which the data introduced in the database are facts of the reality modeled. The transaction time is the time during which the data are available in the database as facts. Database tables are said to be bi-temporal if they contain both columns with the valid time and columns with the transaction time.

3. DATA SCHEMA EVOLUTION SUPPORT SYSTEMS

In the classical temporal databases, only data is temporal. Adding this property to the data schema, we turn to the problem of data schemas varying in time. There exist several approaches to modifying data schemas. They have different application domains and provide one with different sets of capabilities. Schema evolution ensures high performance of the system at the expense of fewer capabilities provided. In this section, we consider schema evolution support systems working with one schema in one data model.

3.1. Schema Evolution, Schema Versioning, and Schema Integration

To avoid ambiguity, we give definitions of the terms "data schema modification," "schema evolution," and "schema versioning." These definitions agree with those in the majority of works on the data schema evolution and schema versioning.

Schema modification is introduction of changes into the schema of a populated database. Examples of such modifications are addition and deletion of types (tables in the relational case and classes in an object-oriented database), their modifications, and changing relations between types.

Schema evolution is incorporation of changes into the schema of a database without loss of data [7, 32].

A *system with data schema versioning* is a system that allows one to access any data that were ever stored in the system by means of a user-defined version interface.

Sometimes, the term schema evolution refers to modification, and the evolution is denoted by the term dynamic evolution [26]. Unlike schema versioning, the evolution does not suggest providing access to data stored in a past or future schema and, as a result, does not necessarily require storing these schemas.

A *system with partial schema versioning* is a system that allows one to access data for reading in any schema but to modify them in only one dedicated (usually current) data schema.

A *system with full schema versioning* allows one to access data and modify them in any schema version.

For many systems, such as medical record or CAD systems, in which nontemporal data elements are not modified, a system with partial schema versioning seems to be more convenient.

Another approach to solving the problem of accessing data stored in different versions of the system is schema integration.

Systems with schema integration provide access to information stored in different schemas as if the information is contained in one schema.

Unlike the schema evolution and schema versioning, schema integration does not restrict the number of schemas and can be used for working with any heterogeneous systems [49].

Schema evolution, schema versioning, and schema integration are generally different approaches to solving the problem of simultaneous accessing data that are stored, or were stored, in different schemas. Each subsequent method solves a more general problem, which is achieved through the increase of the amount of

[3] The authors of this terminology [48] introduce also a user time, which is different from the valid time in that the DBMS does not interpret it.

metainformation stored and the amount of work performed when accessing data. For the XML-relational systems considered in this work, it is sufficient to implement schema evolution or partial schema versioning.

3.2. Metainformation of the Evolution Model

Implementation of one or another evolution model may require storing additional metainformation together with the data schema. In the simplest case, when the data are always stored in the current schema and their transformation from the old schema to the current one is done upon schema modification [18], it is not required to store additional data. In the case of "lazy" update, when the data are stored in the current schema in the first access [16] or every access [24] (which takes place in the schema versioning or schema integration), it is required to store additional metainformation. This metainformation includes mapping objects, i.e., information on how to get data in one schema version based on the data stored in another version.

One of the widely used methods for specifying the mapping objects is to represent them as sequences of elementary schema modifications and data transformations related to each elementary modification. In this case, the set of elementary modifications and the conditions of their applicability specify constraints on the resulting schema. These constraints reflect limitations imposed by the application domain modeled. The constraints may be represented in different forms. One of the ways is to represent them as schema invariants and schema transformation rules that forbid some elementary operations upon fulfillment of certain conditions [16, 18, 20]. The schema invariants are a convenient way to specify semantic constraints, since the amount of memory required for storing them does not depend on the modification history.

In the case where the invariants are simultaneously operation application rules, they are called axioms [26], and the system relying on them, an axiomatic system. Examples of axiomatic models are given in [26, 32].

3.3. Relational Evolution Models

For relational models, a number of temporal extensions [13] have been suggested, including those designed for describing the metainformation. The evolution models in relational DBMSs are temporal extensions of the relational algebra [13] or extensions of the SQL language [14]. In this case, the evolution of a database schema (or each table) is represented as a sequence of ordinary (nontemporal) schemas (tables) and a sequence of mappings from the previous schema (table) to the next one. Both methods for expressing and checking temporal constraints in a relational database and methods for specifying operations that guarantee the fulfillment of these constraints have been studied [14]. For schema versioning, it was suggested in [14] to use a relational metabase for storing information on modifications in the database schema in the form of the following three bitemporal tables: `Relation(Name)`, `Attribute(Name)`, `Relation/Attribute(RelName, AttrName)`. For an SQL extension, the authors of this work suggested to substitute nonterminals `schema-time-clause` and `schema-as-at-clause` for nonterminal table expressions in order to specify tables in the schema that is current at a certain moment.

Evolution models for the "entity–relationship" model [9, 50] are similar to the relational evolution models. The work [9] introduces calculus for the "entity–relationship" model that possesses the completeness property with respect to the ordinary relational calculus [51] and the notion of a relational schema that matches the schema in the "entity–relationship" model. Further, the authors of [9] define incremental invertible operations over the matching relational schema. These operations are built on the basis of ordinary operations of addition/deletion of a relational variable by extending modifications to the keys and inclusion relations (of one variable into another) determined for the given relational variable.

In [52], an extension of the relational algebra is proposed. It adds dynamic functional dependencies imposed on the modified schema to static dependencies. In addition to the set of functional dependencies, the database schema is supplemented with a set of dynamic functional dependencies. The latter is a functional dependence on the union of old and new sets of domains, in which any dependent column, together with the determinant columns, belongs both to the new and old sets of domains.

For the SQL-1999 language standard, the part 7 SQL/Temporal for supporting temporal data (and schema evolution [53]) was developed. The project was suspended, and the SQL-2003 language standard lacks the temporality support.

3.4. Object Evolution Models

In contrast to relational models, where the suggested set of elementary transformations always preserves schema integrity, in the object models (with few exceptions [19]), sets of schema invariants [16] or sets of axioms [26] are responsible for semantic integrity.

The object evolution models appeared in CAD-related studies [7]. Their further development is associated with object systems, such as *Orion* [16], *GemStone* [18], *Encore* [15], *Tigukat* [26], and the like.

Orion is a typical example of using the invariants and rules for describing evolution models. *Orion* uses the following five invariants to modify a schema: type lattice invariant, name invariant, identity invariant, inheritance invariant, and domain invariant. In addition, there are semantic rules for resolving inheritance con-

flicts and rules for modifying the directed inheritance graph. The developers of *Orion* proposed the following taxonomy of elementary operations.

• Operations on fields and methods (properties) of a given type:

addition, deletion, and renaming of a field;

addition, deletion, and renaming of a method;

modification of field inheritance;

modification of method inheritance;

modification of a default field value;

addition, deletion, and renaming of a shared field;

deletion of the property of complex relationship from a field;

modification of a method body.

• Operations modifying the lattice:

addition, deletion, and renaming of a class;

addition and deletion of an inheritance relation between a pair of classes;

modification of an inheritance order for a given class.

For each operation, its semantics is defined. A specific feature of *Orion* is cloning complete schema versions upon each modification. Other systems use incremental schema updates. For example, *Encore* creates a new type for each schema modification only for the modified objects.

Orion uses different algorithms for processing schema versions and data versions. A solution that uses a unified approach for processing both schemas and data has been suggested in [21], in the framework of *Charly*, a DBMS for CAD applications. For this purpose, an object of a version without a reference to a fixed schema is used. Specifically, the version is defined by an object of the form `[id, gen_id, name, number, [successors], [previous], date, state, [[nodes], [edges]]]`, where `gen` is a version identifier, `gen_id` is the identifier of the object containing the sequence of versions, `name` is the version name, `number` is the pointer of the version used for organizing links to it, the lists `successors` and `previous` specify the context (the set of associated classes), `nodes` and `edges` are lists of classes and dependencies between them, `date` is the time of the last schema modification, and `state` determines whether the schema is current.

Sets of elementary transformations similar to that employed in *Orion* are used in the majority of evolution models for object systems. Some researchers suggested extending of this set by more complicated operations, such as operations of partitioning and merging two classes (see, e.g., [29]).

The *GemStone* object-oriented DBMS [18] also uses invariants for modifying schemas. These invariants include the complete inheritance invariant (the successor inherits all type properties), representation invariant (the object is determined by the type), type hierarchy invariant (an analogue of type lattice), constraint inheritance invariant, invariant of absence of links to objects of nonexisting types, and invariant of absence of information loss. *GemStone* introduces operations of property renaming, adding indexing to types, adding properties to types, deleting indexing from types, deleting type properties, changing property constraint, deleting and adding types. The key feature of *GemStone* is that the inheritance model used generates an inheritance tree rather than an arbitrary lattice. Owing to this, the semantics of the introduced operations, as well as checking of the invariants, is considerably simplified.

The most detailed presentation of the axiomatic approach to description of the evolution model is given in [26]. In this work, the schema of an object database is considered as a lattice of types possessing certain properties. The lattice is formed by the inheritance relation; when inheriting similar properties, a semantic conflict resolution is used. For each type, a set of significant properties is defined. It includes properties of the type itself and a set of significant supertypes (i.e., ancestors in the inheritance hierarchy), which includes its direct supertypes. Operations on the model include addition and deletion of types, addition and deletion of the inheritance relation between a pair of types, and addition and deletion of attributes. It is shown in [26] that the proposed axioms make it possible to recover the inheritance relation by the given pairs of sets of significant supertypes and properties. This approach has been implemented for supporting schema evolution in the framework of the object-oriented temporal DBMS *Tigukat*. The taxonomy of the operations in *Tigukat* and *Orion* in terms of the axiomatic model and their comparison is presented in [26].

It is shown in [54] that, for an extension of the *Tigukat* axiomatic model, the schema integrity property and properties of its separate elements can be calculated for exponential time with the use of polynomial memory.

The disadvantage of the above models is that is difficult to implement evolution of the model metadata, which describe the schema as schema parts. This disadvantage can be overcome by means of the approach suggested in [30]. Under this approach, the metainformation is represented as a dynamical relation between first-order objects with the use of the aspect-oriented approach for dynamical determination of feasibility of the inheritance modification operations.

3.5. Evolution Models for XML

Currently, there appeared studies in the field of evolution of XML schemas and XML data. This subject is especially popular in connection with the development of web services, new-type applications using XML as the basic format for data transmission.

In [31], XEM, a new tool for evolution of XML documents described by DTD schemas, is proposed. XEM represents DTD as a graph with marked vertices of several types: vertices–elements, vertices–constraints, and vertices–data. The set of elementary modifications includes the following operations:

- creation of an element,
- deletion of an element,
- addition of an element to another element,
- deletion of an element from another element,
- modification of a quantifier,
- transformation of nested vertices into a group,
- deletion of a group,
- modification of a group quantifier,
- addition of an attribute,
- deletion of an attribute,
- modification of an attribute type,
- modification of an attribute value,
- modification of a fixed attribute.

In [31], completeness of this set of operations is proved, and requirements of consistency and minimality are formulated.

In [32], a model of XML schema evolution based on the model used for the object-oriented DBMS described in [26] is suggested. Confining themselves to the DTD descriptions without recursively nested tags, the authors consider the lattice of tags specified by the nesting relation. The tag attributes serve as analogues of type properties. The significance property is assumed to be specified by the application domain described by the XML documents. The axiomatic model and operation taxonomy for this lattice are similar to those used in the object-oriented evolution support system.

An extension of the model proposed in [32] obtained by including the DTD semantic constraints in the evolution model is presented in [33]. The work [33] introduces also additional operations and modifies operations suggested in [32] such that they work with a more complete DTD representation.

The evolution of more complicated descriptions of XML schemas, such as XML-Schema, is considered in [55]. The problem of validity of the XML documents satisfying the original schema against the modified schema is also studied in [55].

4. XML-RELATIONAL MAPPINGS

The use of XML data requires organization of efficient storage of the data and access to them. One of the solutions is to store data in relational DBMSs. This allows us not only to store and access the data but also to use many useful mechanisms (indices, transactions, multi-user access, and the like) lacking in the majority of XML data models. The important reason affecting the adoption of such technology in the considered class of applications is the fact that, in spite of active research in this field [46, 56, 57] and existence of industrial developments [58], there does not exist an implementation of an XML-oriented DBMS satisfying requirements imposed by a wide variety of applications.

Many methods for storing XML documents in relational databases have been suggested. The XML-relational mappings used in these methods can be divided into two basic groups: methods that do not rely on the information on the schema and methods that do use this information. Methods from the latter group are especially attractive owing to great possibilities for optimizing data access.

4.1. Methods for Storing Data-Oriented and Document-Oriented Documents

Currently, there exist many works devoted to storing XML data in relational DBMSs. The methods proposed in these works can be classified in accordance with the existing types of XML documents [59]. The XML documents can be divided into two groups [58]. The first group includes data-oriented, or regular, documents, and the second group includes document-oriented, or mixed, documents. This partition is not strictly defined, and some documents may be assigned to both groups (they are referred to as hybrid).

Typical features of the documents from the first group are their rather regular structure, absence of tags that contain both nested tags and values, absence of comments, minimum number of control commands and `CDATA` sections. The order of neighboring tags in such documents is often insignificant for the application.

The document-oriented documents, on the contrary, often contain tags with mixed content, comments, and `CDATA` sections. The order of elements is usually important for such documents. The documents themselves have rather irregular structure.

The methods for storing documents of the second type are as follows:

- Storing in values of type `CLOB` and `BLOB`. This method, as well as the majority of the methods discussed below, allows us to use transactions and multi-user access. The basic disadvantage of the method is the absence of means for organizing effective queries to the document.

- XML databases. A number of native XML DBMSs have been suggested [56–58, 60]. Potentially, these methods possess maximum possibilities owing to the implementation of the access facilities to the XML data at the DBMS level. However, the current level of native XML DBMSs is not sufficient for the majority of applications [46].

- Specialized variants of the previous methods, such as PDOM technologies or control systems for XML documents. Such systems implement part of functionality of an ordinary DBMS and are designed for scenar-

ios of work with XML documents in the context of data warehouses.

As noted earlier, for the data-oriented documents, some XML components, e.g., the order of tags, comments, and so on, are insignificant. This makes it possible to use nontrivial, partially invertible transformations between the XML data and relational data. Here, the XML-relational systems make use of the following technologies:

• Middleware performing interaction of the application and DBMS. The schema evolution model can be implemented as such middleware.

• XML support functions built in the DBMS. Usually, these functions are based on the "knowledge" of the DBMS of the relational and XML schemas. A typical example is the XML type in the SQL Server. Another example is a part of the SQL/XML standard of the SQL-2003 language: SQL is supplemented with the `XML` data type and a number of the constructor–functions that allow us to create expressions of the `XML` type based on expressions with ordinary relational types by means of fixed mapping algorithms. Further studies are underway.

• XML servers, i.e., application servers implementing certain functionality of work with XML, such as XML publishing.

• XML Data Binding, automated creation of an intermediate object model, which is stored by using methods of object-relational systems.

We will discuss in more detail XML-relational mappings used for storing data-oriented documents, since the majority of the XML-relational systems work just with this type of XML documents. Such mappings can be classified into the following three basic groups: general, schema-specific, and user mappings.

4.2. General XML-Relational Mappings

General methods use knowledge on the schema of XML documents optionally. Usually, they store data in the schema that is suitable for storing arbitrary documents or documents of similar structure.

A number of works devoted to general mappings consider an XML document as a tree with its vertices marked by the labels corresponding to the names of tags, attributes, and their values [61]. The XML documents are stored by means of the *edge*, *attribute*, and *universal* or *normalized universal* mappings.

The *edge* mapping stores information in the table `Edge(source, ordinal, name, flag, target)`, where `source` and `target` are the beginning and end of the arc in the tree, `name` is the arc name, `flag` shows whether the arc points to a value or a name, and `ordinal` determines order relation for arcs originating from one vertex.

The *attribute* mapping is similar to the previous one except that it stores separate tables corresponding to the projections of the `Edge` table from the previous mapping for retrieval by each value of `name`.

The *universal* mapping stores one table corresponding to the external union of all tables of the *attribute* mapping.

The *normalized universal* mapping stores the result of grouping of the *universal* mapping table by each name; the rows lost in this operation are stored in separate overflow tables for each value of `name`, which are similar to the tables from the *attribute* mapping.

There exist variations of the above-listed methods. First of all, the values can be stored either in special tables of values for each data type or in additional columns of the existing tables. Hybrid algorithms that use several of the above-listed mappings for one document are also used.

A different general mapping is the structure-oriented mapping suggested in [62]. In the framework of this mapping, the XML document is considered as a directed tree.

The simplest implementation of the structure-oriented mapping is that where references to the parents are stored as foreign keys. Another possibility is to store index for each tree node upon the pre- and post-numbering. One more variant, SICF [62], uses sequences of incomplete quotients for storing information about positions of vertices in the tree.

In [63], for the information about the position in the tree, paths from the root to the current value or attribute are stored as XQL queries [64]. In addition, for each path, one integer interval determining the order of the element in the document and one interval with real boundaries determining the hierarchical nesting are stored. Eventually, the information is stored in the tables `Element(docId, pathId, index, reindex, pos)`, `Attribute(docId, pathId, attvalue, pos)`, `Text(docId, pathId, textvalue, pos)`, `Path(pathexp, pathId)`. This mapping is effective for the queries on the basis of which `pathexp` is built.

The *monet* mapping [65] considers the XML document as an ordered tree with vertices possessing labels and attributes with values. To each of the above-listed objects, a unique identifier is assigned. Next, the relations of nesting, order, and correspondence of an attribute to a vertex are stored as one binary relation on the identifiers and the union of identifiers, rows, and numbers.

One more general mapping assumes that the XML document has structure corresponding to a set of tables; this mapping is supported by industrial DBMSs.

4.3. Schema-Specific XML-Relational Mappings

Schema-specific mappings are based on the knowledge of the XML document schema and assume that the documents stored satisfy this schema. Currently, there exist many schema-specific methods, which differ by the underlying algorithms and the XML document

schema used (DTD, XML-Schema, etc.). They can also be classified into fixed and flexible methods; the former rely only on the schema, whereas the latter use additional information, such as statistics collected, for selecting an optimal schema.

The methods proposed in [66] are most frequently used. As all schema-specific methods, these algorithms try to improve the idea of creation of one table for each schema element by using the so-called inlining. The *Basic* algorithm [66] performs a depth-first traversal of the graph constructed from the DTD and creates a new table when finds a multiple occurrence of one element in another or a recursive occurrence of an element. The goal of the *Shared* algorithm is to reduce redundancy of the schema generated by the *Basic* algorithm. Algorithm *Shared* does not create new tables (i.e., inserts the element into parent's tables) only if the corresponding vertex of the DTD graph has incoming degree of 1. It stores separately nodes with the incoming degree of 0 (i.e., potential roots of the XML documents) and stores repeated elements in separate tables. For loops of the DTD graphs without multiple occurrences, *Shared* uses one table. The *Hybrid* algorithm combines properties of the *Basic* and *Shared* algorithms; it inserts elements whose incoming degree is greater than one under condition that they are not recursive and cannot be reached by a path passing through elements with multiple occurrences.

An improvement of the *Hybrid* algorithm has been suggested in [67]. The proposed algorithm *CPI* takes into account some DTD elements, such as constraints `NOT NULL`, `CHECK`, `UNIQUE`, and the like.

Another group of schema-specific methods has been proposed in [68]. They use an intermediate object schema. The elements containing only `CDATA` sections and attributes are mapped into scalar values, and the other elements get aggregate types in the object schema. Attributes and nested elements of a given element become simple or many-valued properties of the corresponding type. The information on the order of elements can be stored in additional properties. The methods suggested in [68] can also work with wider (compared to DTD) subsets of XML Schema. At the second transformation stage, the methods under consideration map the object schema into a relational one: tables of classes correspond to the classes; columns of scalar types, to scalar properties; pointers and links are modeled by keys; and separate tables of properties are created for many-valued properties.

The extended "entity–relationship" model can serve as an alternative intermediate schema upon transformation of the XML model to a relational one. In [69], *Constraints-Preserving Mapping* is suggested. In addition to mapping the structure of the XML document, it preserves the constraints. This is achieved owing to considering the structure of XML Schema as a grammar of a regular language, which is transformed to the extended "entity–relationship "(EER) model in accordance with certain rules.

An interesting representative of flexible mappings is the method proposed in [70]. In the framework of this method, a set of XML-to-XML transformations is considered. They transform the original schema to an "equivalent" one. For each equivalent schema, a transformation to a relational schema similar to that implemented by the *Shared* algorithm is performed; a test set of queries is executed; and an optimal equivalent schema is selected. Thus, the transformation is a composition of an XML-to-XML transformation and a fixed XML-relational transformation.

In [71], an algorithm generating a hybrid general schema-specific mapping has been suggested. It relies on the calculation of the complex element weight, which is formed by results of the analysis of the DTD structure, the structure of the XML documents, and calculation of queries. Depending on its value, either a fixed XML-relational mapping with an intermediate object schema or a mapping to a special XML type, which, in fact, is the LOB type for the relational database, is used.

4.4. User Mappings

User mappings are supported in the majority of industrial DBMSs (Oracle, IBM DB2, Microsoft SQL Server). The DBMSs suggest certain mechanisms for specifying a target schema and a mapping to it. Being the most flexible, this method requires significantly greater efforts of the application designers than the above-listed automated methods. Moreover, this method usually has some restrictions on the possible form of the mappings.

5. SCHEMA EVOLUTION IN THE PRESENCE OF XML-RELATIONAL MAPPINGS

Studies of systems using XML, relational, and object data (especially those on data integration and data schemas) led to the problem of data model management [72]. One of the subproblems of this problem is schema evolution in the presence of mappings between schemas and, as a consequence, evolution of the mappings themselves [73]. A general approach here is the reduction of the evolution of mapping-related schemas to schema integration (see, for example, [74]).

Interesting solutions to the mapping evolution problem upon parallel schema modification have been proposed in [73] and [75]. The authors of [73], using a relational model that admits nested tables for representing relational and XML schemas, suggested an algorithm of schema evolution in the presence of mappings from a wide class. The latter class includes typical mappings used in schema integration, which are described by the GLAV (global-local-as-view) and s-t tgds (source-to-target tuple-generating-dependencies) languages [76]. In [75], it is assumed that the mappings between the

schemas are bijections from a transformation class specified by a special language (which can be reduced in the case of the relational schemas to the GLAV language, i.e., to the mappings given by a set of SELECT statements relating tables of the first and second, or second and first, schemas).

As noted earlier, for the considered class of applications, schema integration is, generally, redundant. In contrast to schema integration, schema evolution does not require permanent transformations in the course of the application operation and provides the required functionality. An example of using schema evolution for describing evolutionary transformations of mapping-related schemas can be found in [35], where an intermediate mediator–schema is used for representing the dependent part of the evolving schemas. The mediator–schema is a directed graph each vertex of which has associated attributes. This model restricts the class of considered XML-relational mappings to supersets of combinations of the *Basic*, *Shared*, and *Hybrid* mappings. Schema evolution in [35] is subdivided into evolution of the schema–broker and evolution of independent parts of the XML schemas and relational schemas. For each of these evolutions, different sets of operations are introduced. The operations on the mediator–schema are similar to the operations in [26] and include addition and deletion of a vertex, edge, and attribute; merging two vertices; and splitting a vertex. Combined with operations on the XML schema, the operations on the mediator–schema allow us to obtain XML schemas of arbitrary form. The operations on independent schema parts extend their modifications to the mappings from the mediator–schema to the XML and relational schemas, implementing thus the evolution of the mapping between the schemas.

6. CONCLUSIONS

Currently, many evolution models relying on different data models have been suggested. Recently, the problem of joint modifications of schemas and mappings between them, the so-called schema adaptation problem, became of interest. The problem is studied basically in the framework of the schema integration. Although the latter approach is most general for evolving schemas, it is, at the same time, the most cumbersome approach associated with considerable overheads of the application operation.

For the considered class of XML-relational systems, such as, e.g., electronic systems of medical records or CAD systems, the joint evolution of mapping-related schemas is the most efficient solution.

REFERENCES

1. *The Computer-based Patient Record: An Essential Technology for Health Care,* Dick, R.S. and Steen, E.B., Eds., Washington, DC: National Academy, 1991.
2. Mallaug, T. and Bratbergsengen, K., Long-term Temporal Data Representation of Personal Health Data, *ADBIS*, 2005, pp. 379–391.
3. XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery
4. Krishnamurthy, R., Kaushik, R., and Naughton, J., *XML-to-SQL Query Translation Literature: The State of the Art and Open Problems*, 2003. citeseer.ist.psu.edu/krishnamurthy03xmltosql.html
5. Krishnamurthy, R., Kaushik, R., and Naughton, J. F., Efficient XML-to-SQL Query Translation: Where to Add the Intelligence? *VLDB,* 2004, pp. 144–155.
6. Pal S., Cseri, I., Seeliger, O., et al., XQuery Implementation in a Relational Database System, *VLDB'05: Proc. of the 31st Int. Conf. on Very Large Data Bases*, VLDB Endowment, 2005, pp. 1175–1186.
7. Roddick, J.F., A Survey of Schema Versioning Issues for Database Systems, *Information Software Technol.*, 1995, vol. 37, no 7, pp. 383–393, citeseer.ist.psu.edu/roddick-95survey.html
8. Clifford, J. and Croker, A., The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans, *Proc. of the Third Int. Conf. on Data Engineering*, Washington, DC: IEEE Comput. Soc., 1987, pp. 528–537.
9. Markowitz, V.M. and Makowsky, J.A., Incremental Reorganization of Relational Databases, *VLDB*, 1987, pp. 127–135.
10. Dadam, P. and Teuhola, J., Managing Schema Versions in a Time-versioned Non-first-normal-form Relational Database, *BTW*, 1987, pp. 161–179.
11. Narayanaswamy, K. and Rao, K.V.B., An Incremental Mechanism for Schema Evolution in Engineering Domains, *Proc. of the Fourth Int. Conf. on Data Engineering*, Washington, DC: IEEE Comput. Soc., 1988, pp. 294–301.
12. McKenzie, L.E. and Snodgrass, R.T., Schema Evolution and the Relational Algebra, *Inf. Systems*, 1990, vol. 15, no. 2, pp. 207–232.
13. McKenzie, L.E. and Snodgrass, R.T., Evaluation of Relational Algebras Incorporating the Time Dimension in Databases, *ACM Comput. Surv.*, 1991, vol. 23, no. 4, pp. 501–543.
14. Roddick, J.F., SQL/SE: A Query Language Extension for Databases Supporting Schema Evolution, *SIGMOD Record*, 1992, vol. 21, no. 3, pp. 10–16.
15. Skarra, A.H. and Zdonik, S.B., The Management of Changing Types in an Object-oriented Database, *OOPSLA,* 1986, pp. 483–495.
16. Banerjee, J., Kim, W., Kim, H.-J., and Korth, H. F., Semantics and Implementation of Schema Evolution in Object-oriented Databases, *SIGMOD Conf.,* 1987, pp. 311–322.
17. Kim, W. and Chou, H.-T., Versions of Schema for Object-oriented Databases, *VLDB*, 1988, pp. 148–159.
18. Penney, D.J. and Stein, J., Class Modification in the Gemstone Object-oriented DBMS, *OOPSLA'87: Conf. Proc. on Object-oriented Programming Systems, Languages and Applications*, New York: ACM, 1987, pp. 111–117.
19. Osborn, S.L., The Role of Polymorphism in Schema Evolution in an Object-oriented Database, *IEEE Trans.*

Knowledge Data Engineering, 1989, vol. 1, no. 3, pp. 310–317.

20. Lerner, B.S. and Habermann, A.N., Beyond Schema Evolution to Database Reorganization, *OOPSLA/ECOOP,* 1990, pp. 67–76.
21. Andany, J., L'eonard, M., and Palisser, C., Management of Schema Evolution in Databases, *VLDB*, 1991, pp. 161–170.
22. Monk, S. and Sommerville, I., Schema Evolution in OODBS Using Class Versioning, *SIGMOD Record*, 1993, vol. 22, no. 3, pp. 16–22.
23. Chen, J.-L., and McLeod, D., Schema Evolution for Object-based Accounting Database Systems, *ISOOMS*, 1994, pp. 40–52.
24. Ra, Y.-G. and Rundensteiner, E.A., A Transparent Object-oriented Schema Change Approach Using View Evolution, *ICDE*, 1995, pp. 165–172.
25. Ra, Y.-G. and Rundensteiner, E.A., Towards Supporting Hard Schema Changes in TSE, *CIKM'95: Proc. of the Fourth Int. Conf. on Information and Knowledge Management*, New York: ACM, 1995, pp. 290–295.
26. Peters, R.J. and Özsu, M.T., An Axiomatic Model of Dynamic Schema Evolution in Object-based Systems, *ACM Trans. Database Systems*, 1997, vol. 22, no. 1, pp. 75–114.
27. Si, A., Leong, H.V., and Wu, P.Y., 4dis: A Temporal Framework for Unifying Metadata and Data Evolution, *SAC'98: Proc. of the 1998 ACM Symp. on Applied Computing*, New York: ACM, 1998, pp. 203–210.
28. Claypool, K.T., Jin, J., and Rundensteiner, E.A., SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework, *CIKM*, 1998, pp. 314–321.
29. Lerner, B.S., A Model for Compound Type Changes Encountered in Schema Evolution, *ACM Trans. Database Systems*, 2000, vol. 25, no. 1, pp. 83–127.
30. Rashid, A. and Sawyer, P., Object Database Evolution Using Separation of Concerns, *SIGMOD Record,* 2000, vol. 29, no. 4, pp. 26–33.
31. Su, H., Kramer, D., and Chen, L., et al., XEM: Managing the Evolution of XML Documents, *RIDE-DM*, 2001, pp. 103–110.
32. Coox, S.V., Axiomatization of the Evolution of XML Database Schema, *Programmirovanie*, 2003, no. 3, pp. 140–146. [*Programming Comput. Software* (Engl. Transl.), 2003, vol. 29, no. 3, pp. 140–146].
33. Coox, S.V. and Simanovsky, A.A., Regular Expressions in XML Schema Evolution, *Vistnik Natsional'nogo tekhnichogo universitetu "Kharkiovs'kii politekhnichnii institut": Zbirnik naukovikh prats', Tematichnii vipusk "Sistemnii analiz, upravlinnya ta informatsiini tekhnologii"* (Special issue "System Analysis, Control, and Information Technologies"), 2004, vol. 1, no. 1, pp. 24–38.
34. Leonardi, E. and Bhowmick, S.S., Detecting Changes on Unordered XML Documents Using Relational Databases: A Schema-Conscious Approach, *CIKM*, 2005, pp. 509–516.
35. Simanovsky, A., Three Layer Evolution Model for XML Stored in Relational Databases, *ADBIS Research Commun.*, 2005, pp. 66–79.
36. Codd, E.F., Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks, *IBM Research Report*, San Jose, California, 1969, vol. RJ599.
37. Codd, E.F., A Relational Model of Data for Large Shared Data Banks, *Commun. ACM*, 1970, vol. 13, no. 6, pp. 377–387.
38. Date, C.J., Darwen, H., and Lorentzos, A., *Temporal Data and the Relational Model*, San Francisco: Morgan Kaufmann, 2003.
39. Meyer, D., *The Theory of Relational Databases,* Rockville, Md.: Comput. Sci., 1983. Translated under the title *Teoriya relyatsionnykh baz dannykh*, Moscow: Mir, 1987.
40. Date, C.J., *The Database Relational Model: A Retrospective Review and Analysis,* Berkeley: Addison Wesley Longman, 2000.
41. Beech, D., Malhotra, A., and Rys, M., A Formal Data Model and Algebra for XML, 1999, citeseer.ist.psu.edu/beech99formal.html
42. World Wide Web Consortium, http://www.w3.org/
43. XQuery 1.0 and XPath 2.0 Data Model (XDM), 2005. http://www.w3.org/TR/xpath-datamodel/
44. Novak, L. and Zamulin, A.V., Algebraic Semantics of XML Schema, *ADBIS*, 2005, pp. 209–222.
45. XML Schema Specification, http://www.w3.org/TR/xmlschema-0/
46. Chaudhri, A., Rashid, A., and Zicari, R., *XML Data Management: Native XML and XML-Enabled Database Systems*, Berkeley: Addison Wesley Longman, 2003.
47. Bernstein, P.A. and Rahm, E., Data Warehouse Scenarios for Model Management, *ER*, 2000, pp. 1–15.
48. Snodgrass, R.T. and Ahn, I., A Taxonomy of Time in Databases, *SIGMOD Conf.*, 1985, pp. 236–246.
49. Fan, H. and Poulovassilis, A., Schema Evolution in Data Warehousing Environments—A Schema Transformation-based Approach, *ER*, 2004, pp. 639–653.
50. Roddick, J.F., Craske, N.G., and Richards, T.J. A Taxonomy for Schema Versioning Based on the Relational and Entity Relationship Models, *ER*, 1993, pp. 137–148.
51. Atzeni, P. and Chen, P.P., Completeness of Query Languages for the Entity–Relationship Model, *ER'81: Proc. of the Second Int. Conf. on the Entity–Relationship Approach to Information Modeling and Analysis*, North-Holland, 1983, pp. 109–122.
52. Vianu, V., Dynamic Functional Dependencies and Database Aging, *ACM*, 1987, vol. 34, no. 1, pp. 28–59.
53. Türker, C., Schema Evolution in Data Warehousing Environments—A Schema Transformation-based Approach, *9th Int. Workshop on Foundations of Models and Languages for Data and Objects*, 2000, pp. 1–32.
54. Franconi, E., Grandi, F., and Mandreoli, F., Schema Evolution and Versioning: A Logical and Computational Characterization, *FMLDO*, 2000, pp. 85–99.
55. Guerrini, G., Mesiti, M., and Rossi, D. Impact of XML Schema Evolution on Valid Documents, *WIDM*, 2005, pp. 39–44.
56. Jagadish, H.V., Al-Khalifa, S., and Chapman, A., et. al., Timber: A Native XML Database, *VLDB*, 2002, vol. 11, no. 4, pp. 274–291.
57. Fomichev, A., Grinev, M., and Kuznetsov, S., Sedna: A Native XML DBMS, *SOFSEM'06, 32nd Conf. on Cur-*

rent Trends in Theory and Practice of Comput. Sci., 2006, pp. 272–281.

58. Bourret, R., XML Database Products: Native XML Databases, 2005, http://www.rpbourret.com/ xml/ProdsNative.htm
59. Mlýnková, I. and Pokorný J., XML in the World of (Object-) Relational Database Systems, 2005, http://cs.engr.uky.edu/ dekhtyar/685-Spring2005/literature/pokorny-tr.pdf
60. Tamino XML Server, http://www.softwareag.com/Corporate/products/tamino/prod_info/architecture/default.asp
61. Florescu, D. and Kossmann, D., Storing and Querying XML Data Using an RDBMS, *IEEE Data Eng. Bull.*, 1999, vol. 22, no. 3, pp. 27–34.
62. Kuckelberg, A. and Krieger, R., Efficient Structure Oriented Storage of XML Documents Using ORDBMS, *EEXTT*, 2002, pp. 131–143.
63. Shimura, T., Yoshikawa, M., and Uemura, S., Storage and Retrieval of XML Documents Using Object-Relational Databases, *DEXA*, 1999, pp. 206–217.
64. Robie, J., Lapp, J., and Schach, D., XML Query Language (XQL), *QL*, 1998.
65. Schmidt, A., Kersten, M.L., Windhouwer, M., and Waas, F., Efficient Relational Storage and Retrieval of XML Documents, *WebDB (Selected Papers),* 2000, pp. 137–150.
66. Shanmugasundaram, J., Tufte, K., and Zhang, C., et. al., Relational Databases for Querying XML Documents: Limitations and Opportunities, *VLDB*, 1999, pp. 302–314.
67. Lee, D. and Chu, W.W., CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema, *Data Knowledge Eng.*, 2001, vol. 39, no. 1, pp. 3–25.
68. Bourret, R., Bornhövd, C., and Buchmann, A.P., A Generic Load/Extract Utility for Data Transfer between XML Documents and Relational Databases, *WECWIS*, 2000, pp. 134–143.
69. Sun, H., Zhang, S., Zhou, J., and Wang, J., Constraints-Preserving Mapping Algorithm from XML-Schema to Relational Schema, *EDCIS*, 2002, pp. 193–207.
70. Bohannon, P., Freire, J., Roy, P., and Siméon, J., From XML Schema to Relations: A Cost-based Approach to XML Storage, *ICDE*, 2002, pp. 64–72.
71. Klettke, M. and Meyer, H., XML and Object-relational Database Systems—Enhancing Structural Mappings Based on Statistics, *WebDB (Selected Papers)*, 2000, pp. 151–170.
72. Bernstein, P.A., Applying Model Management to Classical Meta Data Problems, *CIDR*, 2003.
73. Velegrakis, Y., Miller, R. J., and Popa, L., Mapping Adaptation under Evolving Schemas, *VLDB*, 2003, pp. 584–595.
74. McBrien, P. and Poulovassilis, A., Schema Evolution in Heterogeneous Database Architectures: A Schema Transformation Approach, *CAiSE*, 2002, pp. 484–499.
75. Yu, C. and Popa, L., Semantic Adaptation of Schema Mappings when Schemas Evolve, *VLDB*, 2005, pp. 1006–1017.
76. Fagin, R., Kolaitis, P.G., Miller, R.J., and Popa, L., Data Exchange: Semantics and Query Answering, *Theor. Comput. Sci.,* 2005, vol. 336, no. 1, pp. 89–124.