Journal of Web Engineering, Vol. 4, No.3 (2005) 205-223

A GENERIC FRAMEWORK FOR EXTRACTING XML DATA FROM LEGACY DATABASES

PHILIPPE THIRAN

Technische Universiteit Eindhoven, The Netherlands
University of Namur, Belgium
philippe.thiran@fundp.ac.be

FABRICE ESTIEVENART

CETIC, Gosselies, Belgium
fe@cetic.be

JEAN-LUC HAINAUT

University of Namur, Belgium
jlh@info.fundp.ac.be

GEERT-JAN HOUBEN

Technische Universiteit Eindhoven, The Netherlands
g.j.houben@tue.nl

Received October 29, 2004
Revised April 29, 2005

This paper describes a generic framework of which semantics-based XML data can be derived from legacy databases. It consists in first recovering the conceptual schema of the database through reverse engineering techniques, and then in converting this schema, or part of it, into XML-compliant data structures. Both steps heavily rely on generic schema transformation techniques, while all the schemas involved in the whole process are expressed in a unique model, named GER. Transformations between schemas are expressed as sequences of reversible transformation primitives. The reversed sequence can be used to automatically translate query, data or updates posed on XML.

Keywords: XML, Legacy Database, Schema Transformation, Data Reverse Engineering

1 Introduction

The quantity of information available on the World Wide Web continuously increases, notably due to local, isolated, databases being made accessible through Web technologies. Indeed, most of this information is provided by Web information systems that combine traditional storage mechanisms, e.g., relational databases, with the easy access mechanism that gave the Web its popularity. The term *deep Web* is sometimes used to distinguish the wealth of information stored in these Web-accessible information systems from the *surface Web* of explicitly linked HTML pages. However, the information on the (deep) Web is placed there independently by different organizations. As a consequence, data sources containing related information can appear at different Websites and systems, in different formats, and for different purposes. Moreover, the current Web information systems are lacking particularly in the area of semantics, for example in terms of relationships within the data.

Currently, XML is becoming the *de facto* standard for publishing and exchanging data over the Web. The use of XML as the common format for representing, exchanging, storing, and accessing data poses new challenges to information systems. Since the majority of everyday data is still stored and maintained in *standard* database and file systems, we expect that the needs to export stored data in XML format will grow substantially. To this end, several projects recently investigated the issues of converting database schema in XML ([1], [2], [3], [4], [5]).

1.1 Proposal

In this paper, we focus on extracting an XML Schema [6] from existing and legacy databases. As mentioned, some research projects have already investigated issues related to schema conversion between XML and database models. Unlike their approaches, we analyse the problem from a *model-independent*, *conceptual* and *generic* perspective:

- *Model-independent perspective.* Current approaches for exporting databases into XML rely on pairs of models, such as those intended to produce XML views of relational schemas. In this work, we use a high-level formalism - named Generic Entity Relationship model (GER) - in which schemas can be expressed whatever their underlying data model and their abstraction level. Such a formalism defines a reference model on which transformational operators are built. Through a specialization mechanism, arbitrary models (including XML Schema) as well as schema transformations can be defined in a uniform way, even among schemas expressed in different models. As a result, our approach is not limited to any specific pair of data models.
- *Conceptual perspective.* Most current transformation strategies consist in translating each construct of the source database into the closest constructs of XML without attempting any semantic interpretation. They capture the structures of the source schema and largely ignore the implicit interconnection among data and the hidden semantic constraints implemented in the program codes of the source applications. Our strategy consists in recovering the precise semantic description, i.e., the conceptual schema, of the source database first, through *reverse engineering techniques*, then in developing the XML Schema from this schema through a semi-automated model translation.
- *Generic perspective.* The material developed in this paper is intentionally generic. Considering an ideal scenario according to which the whole contents of an existing (possibly legacy) database is migrated to XML documents, it addresses some of the most critical issues of database-to-XML conversion. More specific, and therefore practical, scenarios, can be derived easily. Database extraction for data warehouse loading, B2B messages generation, database migration or database-to-Web publishing are some processes that require strong theoretical bases of similar nature. This paper is a contribution to the development of such bases.

1.2 Paper Organization

This paper is organized as follows.

In Section 2, we present the GER model used to express all the data models involved in the XML exportation process of legacy databases. We explain how any database model currently used (be it physical or conceptual) as well as the XML Schema model can be represented by specializations of the GER.

In Section 3, we introduce the transformational approach based on the GER. We propose a set of transformational operators addressing the translation of structured schemas into XML Schema. These first two sections lead to the first technical contribution of the paper - providing a high-level reference model for database and XML models and hence, the possibility of transforming data instances between them.

In Section 4, we develop a new approach for converting database structures into an XML Schema. It comprises two steps: the first one consists in recovering the conceptual schema of the source database through reverse engineering techniques, while the second one transforms this schema into XML Schema compliant structures. Both steps rely on the schema transformation techniques built on the GER. The approach is illustrated by a case study that points at different benefits of conversion. This leads to the second technical contribution of the paper - providing a method for extracting the XML Schema of databases which captures their hidden semantic structures.

Section 5 presents an operational CASE tool - DB-MAIN - that supports this entire approach. In Section 6, we discuss related works. We give our concluding remarks in Section 7.

To focus the presentation on the translation problem, the paper has been written for a scenario in which the whole database is to be translated into XML structures. In actual situations, usually only selected parts have to be exported or transformed.

2 Data Structure Modelling

Constructs of data models such as Entity-relationship (ER for short) or XML can be defined in terms of a unique wide spectrum variant of the Entity-relationship model, the GER. This reference model makes it possible to specify different data models in a uniform formalism. In [7], we show that the GER can represent any data structure whatever its underlying data model and its abstraction level. In this section, we present the main concepts of the GER as well as three of its most important specializations that are used in this paper, namely the standard Entity-relationship model, the relational model, and the XML Schema model. Indeed, the approach described in this paper postulates that any translation process includes the abstract, technology-independent, representation of the data to be converted. Therefore, it is based on a three model architecture, comprising the source physical model (typically, but not exclusively, the relational model), the technology-independent conceptual model and the target logical model (here, the XML Schema model).

2.1 The Generic Entity-Relationship Model (GER)

For the need of this paper, the GER can be perceived as an enriched variant of the standard entity-relationship model. It includes the concepts of *entity type*, *is-a hierarchy*, *attribute*, *value domain* and *relationship type*. Attributes can be atomic or compound, mandatory or optional, single-valued or multivalued. Each role of a relationship type can be labelled and associated with a cardinality constraint, a pair of integers, such as 0-20, stating the range of the number of relationships in which any entity can appear. An attribute has a cardinality constraint too, that states how many values can be associated with each parent instance. Default constraint is 1-1 and does not appear in graphical schemas.

In general, additional properties can be declared among the components of an entity type: uniqueness, referential and existence constraints are just some of them. Due the wide variety of such properties, the GER includes the generic concept of *property group*, or *group* for short. A group is any subset of components (attributes and/or roles) of an entity type on which one or several properties are defined. The tag(s) of the group specifies its properties (id for identifier, ref for referential, excl for exclusion, and so on). For example, a group of attributes of entity type E can be declared both identifier and referential. This group models such relational pattern as a primary key that simultaneously is a foreign key. In XML representation, property groups will be used to model ID, KEY, IDREF, sequence or choice constructs. However, they can also model constraints that cannot be explicitly expressed in XML, and that will be translated in another way, through filters or procedural components for example. Finally, GER includes extension mechanisms such as *stereotypes* and *meta-attributes*, through which one can express specific characteristics or behaviour.

2.2 GER Expression of the Entity-relationship Model

Since the GER has been designed as an extension of the standard Entity-relationship model, specializing the former to any conceptual model is fairly easy. Figure 1 shows a typical conceptual schema made up of entity types, an is-a hierarchy, binary and N-ary relationship types, roles and attributes with cardinalities, single-/multi-valued attributes, atomic/compound attributes, mandatory/optional attributes, primary (id) and secondary (id') identifiers, and complex identifiers.

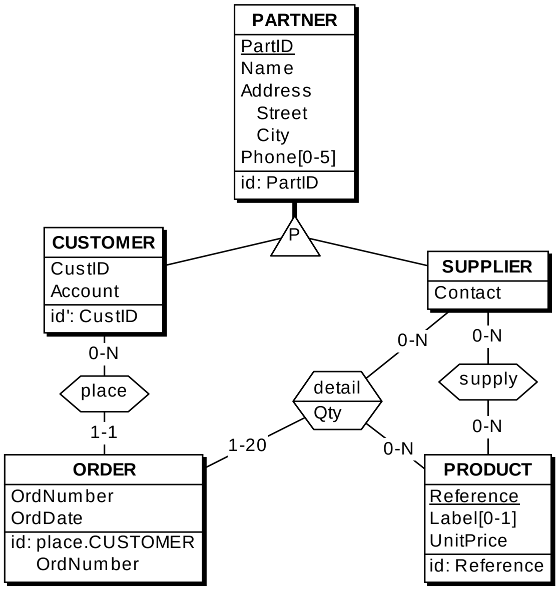


Fig. 1. The GER representation of a typical Entity-relationship schema.

2.3 GER Expression of the Relational Model

Whatever the technology, a relational schema comprises tables, value domains, mandatory/optional columns, candidate keys, including a primary key, and foreign keys. These concepts are modelled by GER objects as defined in Table 1. The GER concepts that have not been mentioned in the right column are not part of the *Relational specialization* of the GER. More detail on this mapping can be found in [7].

Table 1. The Relational specialization of GER.

Relational concept	**GER construct**
Table	Entity type with stereotype «table»
Domain	Domain
Column (nullable)	Atomic attribute with cardinality [0-1]
Column (not null)	Atomic attribute with cardinality [1-1]
Primary key	Primary identifier
Other candidate key	Secondary identifier
Foreign key	Reference group

Figure 2 shows a relational schema that derives from a part of schema of Figure 1. The complex cardinality constraint on role detail.ORDER cannot be expressed by pure relational structures and has been stated through a semi-formal annotation.

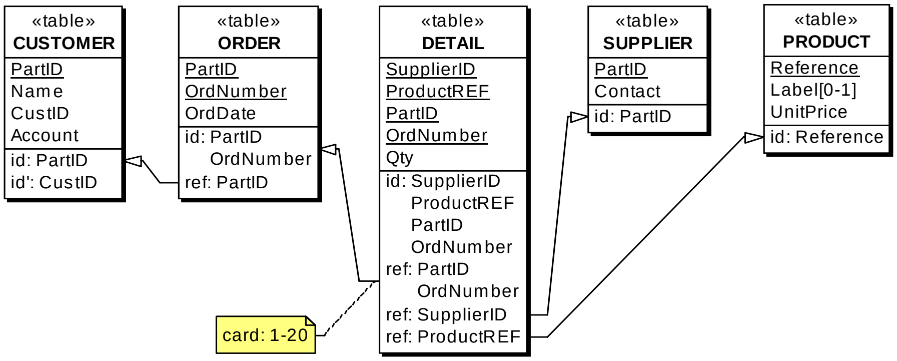


Fig. 2. The GER representation of a typical relational schema.

2.4 GER Expression of the XML Schema Model

Expressing XML structures in terms of a reference model has already received much attention. The most popular approach consists in using the ER model, or its variant in UML diagrams, for representing XML or semi-structured data ([8] and [9]). However, due to the differences between these models, such approaches cannot express all the concepts of the XML Schema model. Indeed, these models often lack features to represent specific XML concepts such as sequences, uniqueness/reference constraints, or built-in/derived data types. In [18], we have developed a specialization of the GER that is able to represent the most important constructs of XML DTD. In this paper, we extend this specialization to the much richer XML Schema model. In particular, it supports advanced constructs such as precise cardinalities, complex uniqueness constraints, connectors between elements and data types.

It is important to note that, due to the scope of the proposal, we target the subset of the XML Schema specifications, as defined by the W3C Recommendation [10], that allows us to express the semantics of any ER schema. Therefore, we will ignore the XML Schema constructs that are not used to translate ER concepts. The effort would have been more important if we intended to express the semantics of native XML Schemas, for instance in an XML reverse engineering process. We will therefore ignore the following constructions:

- Namespaces,
- Abstract elements and types,
- Data types derived (by union, by extension or by restriction) from a non built-in type,
- Attribute groups,
- Substitution groups,
- Uniqueness constraints whose field has an Xpath with a depth greater than 2 (Section 2.4.4).

In the next sections, we explain how an XML Schema is expressed in terms of the GER. We successively examine the concepts of *Simple Element Type*, *Complex Element Type*, *Data Type* and *Uniqueness/Reference Constraint*. Table 2 summarizes the correspondences between the main XML Schema concepts and their GER interpretation. The discussion that follows will be based on the example “Catalog” describing orders and their products (Figure 3), the conceptual schema of which is given in Figure 7.

Table 2. XML Schema specialization of the GER.

XML Schema concept	**GER construct**
Element type	Entity Type
Attribute type	Attribute
Element content	Binary relationship type
Occurrence constraints	Role cardinalities
Sequence/Choice/All connector	`seq/choice/all` group
Data type	User-defined attribute type
Uniqueness/Reference constraints	`id/ref` group

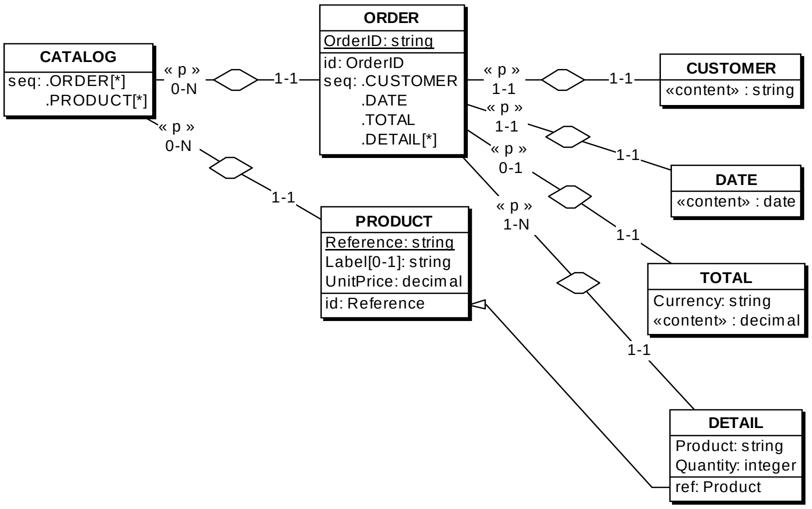


Fig. 3. The GER expression of an XML Schema Catalog.

2.4.1 Simple Element Types

A simple element type has no attributes and its content is an elementary value. In the GER representation, it appears as an entity type whose name translates that of the XML element type. A specific attribute with stereotype «content» is used to express the associated (built-in or derived) content type.

Example. In the XML Schema of Figure 3, CUSTOMER and DATE are simple element types. They are represented by eponym entity types.

2.4.2 Complex Element Types

A complex element type has attributes and/or its content references other element types. It is modelled by a GER entity type with attributes and components organized as follows.

Attribute Types

XML attribute A associated with complex element type C is translated into GER attribute A associated with the entity type that models C. XML attributes are single-valued and either mandatory or optional, these properties being translated into GER attribute cardinality 1-1 or 0-1. Graphically, the cardinality 1-1 is default, and is not represented to simplify schemas. XML Schema prohibits multivalued attributes.

Example. The attributes Reference and UnitPrice of element type PRODUCT in Figure 3 are mandatory (cardinality 1-1) while Label is optional (cardinality 0-1).

Element Content

The content of a complex entity type is made up of element types according to complex assembly rules. These components themselves are simple or complex element types. Recursive structures are allowed, that is, an element type can be a direct or indirect component of itself.

The relation between a complex element type and each of its components is modelled in GER by a one-to-many binary relationship type (called *hierarchical relationship type*) between the corresponding entity types. The parent entity type plays the *parent role* (as opposed with the *child role*) and is marked with stereotype «p».

Occurrence Constraints

The occurrence constraint of a component element type states the minimum (minOccurs) and maximum (maxOccurs) numbers of component elements that can appear for each parent element. The default values are 1-1. These constraints are translated into the cardinality constraint of the parent role of the hierarchical relationship type.

Sequence/Choice/All Connector

The components of a parent element type can be organized according to three rules, namely sequential (connector *sequence*), disjunctive (connector *choice*) or conjunctive (connector *all*). These rules are declared in the GER representation through a *property group* comprising the components and tagged with name *seq*, *choice* or *all*, respectively. A component appears in this group through the role it plays in the hierarchical relationship type. The order of the components within a *seq* group is meaningful.

Example. In Figure 3, the element type modelled by entity type ORDER is made up of a sequence of four components, namely CUSTOMER, DATE, TOTAL and DETAIL.

2.4.3 Data Types

One major initiative in XML Schema as compared with DTD is the large range of data types that can be assigned to an attribute or an element value. The GER offers the basic data types (such as char, boolean, numeric) but provides also user-defined types through which all the XML Schema data types can be defined. As already stated, the contents of a simple element type is modelled by a GER attribute, with stereotype «content» but left unnamed, as illustrated in Figure 3. The data type of the element type is associated with this technical attribute. The XML Schema feature according to which a new simple type can be derived, by restriction or extension, from existing simple types can be modelled in the GER. However, it has not been included in the discussion since it is useless when translating ER schemas into XML Schema structures.

2.4.4 Uniqueness/Reference Constraints

In contrast with global, document-wide, DTD identifiers, XML Schema provides constructs to define uniqueness properties of attribute or element values within a certain context. Such a uniqueness constraint is introduced by the *element key*[a] and is expressed by combination of (at least) two Xpath expressions: one that defines the scope of the constraint (the *element selector*) and (at least) one that identifies the attribute(s) or element(s) the value(s) of which must be unique (the *element field*) within the scope. The scope always refers to an element while a field can be an attribute or an element.

According to the structural link between the element types targeted by the selector and the field (if the field refers to an attribute, we consider the parent element type of that attribute), we distinguish three types of uniqueness constraint:

(i) The element type referred by the selector is the element type referred by the field,

(ii) The element type referred by the selector is the parent of the element type referred by the field,

(iii) The element type referred by the selector is a higher ancestor (parent or ancestor of the parent) of the element type referred by the field.

Standard Entity-relationship models include identifier patterns that obey to the first two types described here above. Therefore, we discard the last one from the target constructs in the ER-to-XML translation process. In the GER, a uniqueness constraint is expressed by means of a group labelled id.

In XML Schema, the *element keyref* translates a foreign key towards a key (or unique) construct. Based on (at least) two Xpath expressions defining the scope and the fields concerned by the constraint, its use is very similar to the use of the *element key*. Such a reference constraint is represented, in a GER schema, by a ref group.

Example. In Figure 3, a uniqueness constraint is defined on the attribute Reference within the scope of the element PRODUCT. That constraint appears in the GER schema as an id group associated with entity type PRODUCT. The group comprises attribute Reference and is additionally the target of a foreign key defined on the attribute Product of the element DETAIL.

3 Schema Transformation

The GER is the ideal support for schema transformations. Indeed, transformations can be used whatever their underlying data model and their abstraction level. In [7], we define a set of transformations for GER schemas. These transformations can be applied by a developer to formally specify mappings between schemas expressed in the same or different data modelling languages.

[a] The *element unique* is used for optional (secondary) identifiers.

3.1 Principles

A transformation consists in deriving a target schema S' from a source schema S by replacing construct C (possibly empty) in S with a new construct C' (possibly empty). More formally, considering instance c of C and instance c' of C', a transformation Σ can be completely defined by a pair of mappings <T,t> such that C' = T(C) and c' = t(c). T is the structural mapping, that explains how to replace construct C with construct C' while t, the instance mapping, states how to compute instance c' of C' from an instance c of C.

3.2 Inverse Transformation

Each transformation $\Sigma_1 \equiv <T_1,t_1>$ can be given an inverse transformation $\Sigma_2 \equiv <T_2,t_2>$, usually denoted by Σ^{-1}, such that, for any structure C, $T_2(T_1(C)) = C$. So far, Σ_2 being the inverse of Σ_1 does not imply that Σ_1 is the inverse of Σ_2. Moreover, Σ_2 is not necessarily reversible. These properties can be guaranteed only for a special variety of transformations[b], called *symmetrically reversible*. Σ_1 is said to be a symmetrically reversible transformation, or more simply semantics-preserving, if it is reversible and if its inverse is reversible too.

3.3 Transformation Sequence

A transformation sequence is a list of n primitive transformations: S1-to-S2 = (T1 T2 ... Tn). For instance, the application of S1-to-S2 = (T1 T2) on a schema S1 consists of the application of T2 on the schema that results from the application of T1, so that we obtain S2.

As for schema transformations, a transformation sequence can be inverted. The inverse sequence S2-to-S1 can be derived from the sequence S1-to-S1 and can be defined as follows: if S1-to-S2 = (T1 T2 ... Tn) then S2-to-S1 = (Tn^{-1} ... $T2^{-1}$ $T1^{-1}$) *where* Ti^{-1} is the inverse of Ti; and hence S1 = S2-to-S1(S2). In other words, S2-to-S1 is obtained by replacing each original schema transformation by its inverse and by reversing the operation order.

The concepts of sequence and its inverse are used for defining the mappings between two schemas. The transformational approach then consists in defining a (reversible) transformation sequence which, applied to the source schema, produces the target schema (Section 4).

3.4 Model Translation

A model translation is a particular case of schema transformation. It consists in translating a schema expressed in a data model Ms into a schema expressed in another data model Mt where Ms and Mt are two different specializations of GER.

A model translation is defined by a *transformation plan*, which is a high level semi-procedural script that describes how to apply a set of transformations in order to fulfil a particular task or to meet a goal. A model translation therefore consists in applying the relevant transformations on the relevant constructs of the schema expressed in Ms in such a way that the final result complies with Mt. Section 4 includes a nice illustration of a complex model translation plan that explains how to transform a conceptual schema into an XML Schema.

[b] In [7], a proof system has been developed to evaluate the reversibility of a transformation.

3.5 Transformation Operators

Transformations are operators that manipulate GER constructs. We propose in Figures 4 and 5 two subsets of the transformational operators that will be used in this paper (Section 4.2). The first one is made up of four standard transformations used in translation of relational schemas into conceptual schemas (named hereafter *standard* transformations). The second comprises additional techniques suited to derive an XML Schema from a conceptual schema (named hereafter *XML-specific* transformations). These transformation operators are formally described in [7] and in [11].

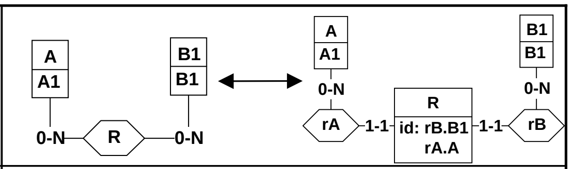
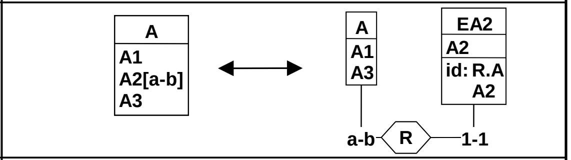
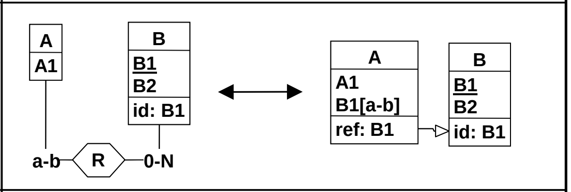
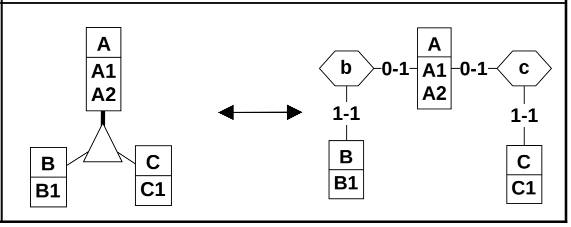
RT-ET: Transforming a relationship type into an entity type. *Inverse:* **ET-RT**	
Att-ET/inst: Transforming an attribute into an entity type (instance representation). *Inverse:* **ET-Att**	
RT-FK: Transforming a binary relationship type into a foreign key. *Inverse:* **FK-RT**	
ISA-RT: Materializing an ISA relationship type. *Inverse:* **RT-ISA**	

Fig. 4. Standard schema transformations.

4 Exporting XML Data From Legacy Databases

This section describes a conceptual approach for extracting the XML Schemas of legacy databases. Normally, a precise and detailed documentation should be associated with any database, in such a way that the XML Schema expression should be straightforward. However, the actual state of most databases can be described, at best, as *undocumented*. Hence the need to rebuild the conceptual schema of the source database when it is no longer available. This process is commonly called *reverse engineering*.

Our strategy consists in recovering the precise semantic description of the source database before converting it into XML Schema. It follows a semi-automated three-step method (Figure 6):

- *Reverse-engineering:* from the DDL code and from various other sources, the conceptual and the physical schemas are extracted (GER PS, GER CS) through specific analysis techniques,

- *Model translation:* the GER XML Schema (GER XS) is semi-automatically generated from the conceptual schema (GER CS) by means of schema transformations,
- *Schema exportation:* the XML Schema (GER XS) is translated into an XML document (XML Schema Code).

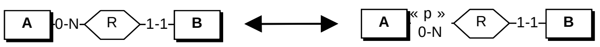
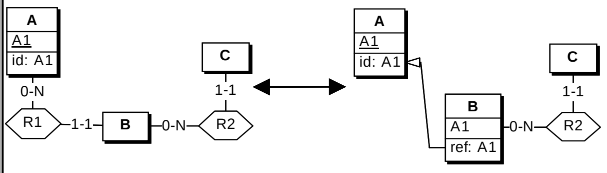
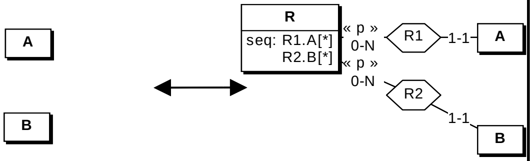
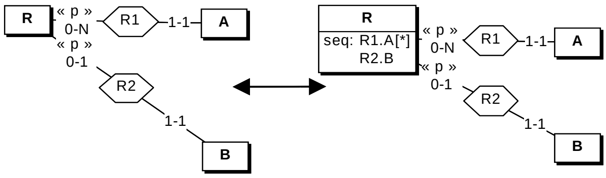
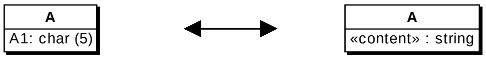
XML-RT-to-HIER: Transforming a one-to-many binary relationship type into a hierarchical relationship type. *Inverse:* **XML-HIER-to-RT**	 A 0-N R 1-1 B A « p » 0-N R 1-1 B
XML-ET-to-ROOT: Giving an entity type the status of natural root. *Inverse:* **XML-ROOT-to-ET**	 A A1 id: A1 0-N R1 1-1 B 0-N R2 1-1 C A A1 id: A1 B A1 ref: A1 0-N R2 1-1 C
XML-Sch-Create-UNIQUE-ROOT: Adding a unique root to the schema. *Inverse:* **XML-Sch-Del-ROOT**	 A B R seq: R1.A[*] R2.B[*] « p » 0-N « p » 0-N R1 1-1 A R2 1-1 B
XML-ET-Create-SEQ-GROUP: Adding a seq group to an entity type. That group contains the child roles played by its children. *Inverse:* **XML-ET-Del-SEQ-GROUP**	 R « p » 0-N « p » 0-1 R1 1-1 A R2 1-1 B R seq: R1.A[*] R2.B « p » 0-N « p » 0-1 R1 1-1 A R2 1-1 B
XML-Att-to-CONTENT: Converting the type of an attribute into the associated XML Schema type and adding the stereotype <<content>>. *Inverse:* **XML-CONTENT-to-Att**	

Fig. 5. Specific XML Schema transformations.

The goal of the first step is to recover a conceptual view of the legacy database [13] while the second step form the standard forward engineering stream, targeting the XML Schema. All the schema mappings are expressed by means of schema transformation sequences (PS-to-CS and CS-to-XS).

In the following sections, we describe the first two steps[c] and illustrate them by a small example. We consider a legacy database for order management. The DBMS is Oracle V5, which ignores the concepts of primary and foreign keys.

[c] We omit the Schema Exportation process because of its simplicity.

4.1 Data Reverse Engineering

This step consists in recovering the most faithful description of the legacy database that captures the intention, that is, the semantics, of the data structures. According to the DB-MAIN reverse engineering methodology ([12], [13]), this recovery raises two families of problems, namely implicit constructs and semantics translation. They are addressed in the following two processes.

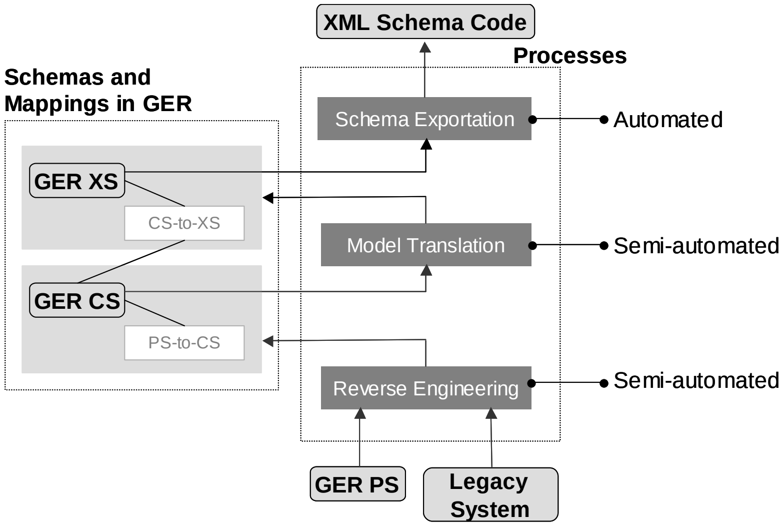


Fig. 6. Database exportation to XML: processes, schemas and transformation sequences.

4.1.1 Data Structure Extraction

The DDL code provides us with the data structures that actually were declared when the database was built (the raw physical schema or GER PS). Due to the intrinsic weaknesses of the DDL of the legacy DBMS, but also as a result of poor programming techniques, most database schemas translate only a part of the intention of the designer. The missing constructs are called *implicit*. The goal of this process is to identify them through specific analysis methods. In particular, program source code inspection, for instance by dependency analysis or program slicing, can show us the exact structure of a record type, uniqueness constraint or implicit foreign key. Data analysis can be used, either to identify specific patterns and relationships, or to evaluate hypotheses on such constructs. In the example of Figure 7, the physical schema translates the DDL code in terms of tables, columns, indexes (acc tag for *access keys*) and storage space (FileCatalog). No keys have been declared. The main goal of the extraction process is to identify primary keys and foreign keys. Source code and data analysis show that the primary keys of tables ORDER, DETAIL and PRODUCT are, respectively, OrderID, (OrderID, Reference), and Reference. In addition, columns Tot_Currency and Tot_Amount are characteristics of a same (hidden) common concept *total* that suggests an implicit compound attribute (Total). Finally, data analysis suggests that every ORDER row is referenced by at least one DETAIL row.

4.1.2 Data Structure Conceptualization

The goal of the second process is to interpret the physical structures in terms of the application domain concepts. We consider that the (enriched) physical schema results from the lossy translation of the conceptual schema and from optimization restructuring techniques. The problem is to identify which techniques were applied for translating and optimizing. The DB-MAIN approach is based on transformational techniques to reverse the effect of the translation and optimization processes. For instance, in the example of Figure 7, the implicit foreign key Reference is replaced with a relationship type (transformation FK-RT in Figure 4), then the table Detail is interpreted as (replaced by) many-to-many relationship type Detail (transformation ET-RT).

The schema GER CS (Figure 7) is the result of the conceptualization process. It is assumed to explicitly express all the semantics of the legacy database, and is a sound source for the generation of the XML structures.

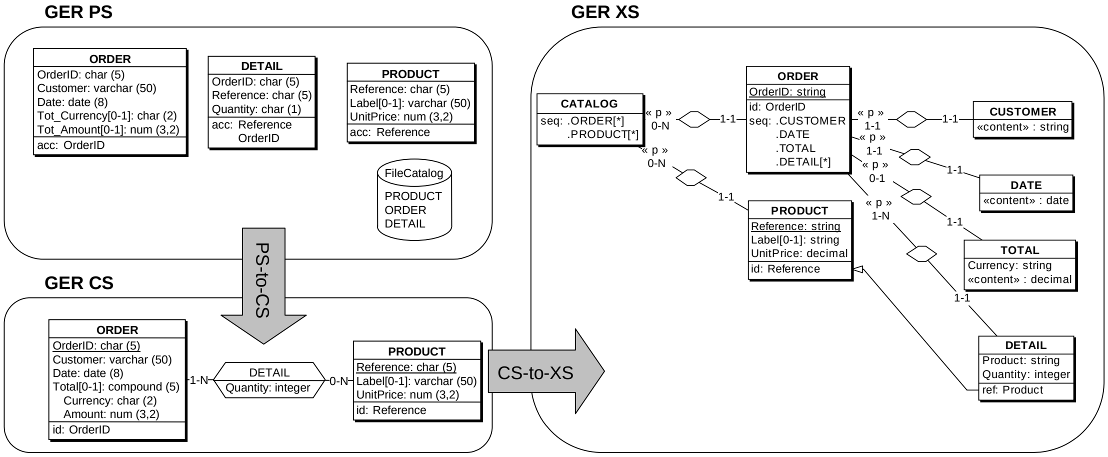


Fig. 7. Example of database exportation: physical, conceptual and XML schemas and the transformation sequences between them.

4.2 Model Translation

In this section, we describe a specific case of model translation that is to transform a conceptual schema (GER CS) into an equivalent XML Schema (GER XS). Model translation is expressed by means of schema transformations and consists, in that case, of applying relevant transformations on the constructs of the GER CS that are not compliant with the XML Schema model. Its execution produces two result types: (1) a target schema expressed in GER XS; and (2) a schema transformation sequence CS-to-XS that is made up of all the transformations applied on the source schema to get the target one. This includes standard as well as specific XML Schema transformations (Section 3.5). These model-specific transformations are illustrated in Figure 5.

The five steps composing the transformation process are the following:

- *Schema binarisation:* a flat binary schema is obtained by transformation of is-a relations, non-functional (i.e., not purely *one-to-many*) relationship types and complex attributes. This step produces a simplified schema.
- *Hierarchical structure creation:* a hierarchical organization between entity types is built by choosing the main concepts and the most natural relations between them according to the modelled application domain.
- *Attribute type conversion:* abstract attribute types defined in the GER CS are converted into an equivalent XML Schema type.

- *Attributes representation:* attributes become either XML attributes or XML elements.
- *Ordering definition*: each entity type is given a position relative to its respective parent.

In this approach, model translation can be considered as a non-deterministic process. Some steps are completely automated while others require some design choices from a database expert in charge of the model translation. These choices depend on the goal assigned to the target XML structures.

In some cases, transformations used in model translation may not preserve semantics, resulting in a GER XS that is not semantically equivalent to the source conceptual schema. A typical case is the transformation into a foreign key of a functional relationship type with both minimum cardinalities equal to 1.

In the following, we give more details on each step of the transformation process and illustrate them by processing the conceptual schema GER CS of Figure 7.

4.2.1 Schema Binarisation

The schema simplification process uses standard schema transformations (Figure 4) to remove is-a relations, N-ary (N>2) and many-to-many relationship types, multivalued and compound attributes. The result is a *flat binary schema*, i.e., a schema whose relationship types are binary and one-to-many or one-to-one and in which all attributes are atomic and single-valued.

Example. In Figure 7 (GER CS), the many-to-many relationship type DETAIL is transformed into an entity type (RT-ET) and the multivalued attribute PRODUCT.Supplier becomes an entity type (ATT-ET/instance).

4.2.2 Hierarchical Structure Creation

In that step, the structure formed by the entity types and the relationship types is transformed into a tree by choosing the natural root entity type(s), solving the parent conflict(s) and, eventually, if necessary, adding an unique root. Those three subtasks mainly use two transformations: RT-FK (a standard transformation) and XML-RT-to-HIER.

XML-RT-to-HIER (Figure 5) is a XML-specific transformation that is applied on a binary relationship type in order to make it hierarchical. That transformation introduces an explicit parent-child relation between the two concerned entity types according to the cardinality of their role. The child entity type plays the role with cardinality 1-1. In case of both roles having this cardinality, the child role must be chosen by the designer.

Root(s) Election

Some entity types naturally appear to be roots, when they do not depend on any other entity types. Typically, all their roles, if any, have a cardinality 0-j. Besides these *natural roots*, other roots can be identified. Among the other entity types, those which represent a significant concept in the application domain, can be declared user-defined roots by the designer. The XML-specific transformation XML-ET-to-ROOT (Figure 5) gives the status of root to any entity type in the schema.

For example, entity type A is a natural root in the source schema of Figure 5. The entity type B is considered to be important by the designer and is therefore given the status of user-defined root through the XML-ET-to-ROOT transformation.

Parent Conflicts Resolution

A *parent conflict* occurs as soon as an entity type has more than one parent, which leads to a non-hierarchical structure. The parent conflict resolution consists in choosing one entity type A among the potential parents of the problematic entity type B, in a such a way that the relationship type between A and B gets the status of *hierarchical relationship type*. The other relationship types for which B is a child become *inter-hierarchy relationship types*. In practice, the relationship type between the entity type and the elected parent is made hierarchical (XML-RT-to-HIER) and all other concerned relationship types become foreign keys (RT-FK).

Unique Root Addition

If the current schema contains more than one root, one particular entity type (called *technical root*) is added as the unique root of the schema. The XML-specific transformation XML-Sch-Create-UNIQUE-ROOT (Figure 5) adds that technical root that becomes the parent of all the natural and user-defined roots.

Example. The entity types ORDER and PRODUCT are two natural roots; no other roots have been elected. DETAIL has two parents. The relationship type between ORDER and DETAIL is declared *hierarchical*, so that the relationship type between PRODUCT and DETAIL has to be transformed into a foreign key.

Finally, a unique technical root (entity type CATALOG) is added to get the targeted tree organization.

4.2.3 Attribute Types Conversion

In GER, abstract data types, such as char, numeric or boolean, are assigned to attributes. To be valid against our XML target model, these types are automatically mapped and transformed to the XML Schema built-in types. For example, a GER char type is converted into an XML Schema string while a GER numeric type becomes an XML Schema byte, short, int or long, according to its initial length.

4.2.4 Attributes Representation

At this point, attributes can be handled in two different ways. They can either keep their status of attribute or be promoted to a simple element type. That decision is a pure design choice. In the latter case, the attribute is transformed into an entity type (Att-ET/Inst) and a hierarchical relationship type binds both related entity types. Logically, the content type of the new entity type is determined by the initial attribute type.

Example. The attributes ORDER.OrderID, PRODUCT.Reference, DETAIL.Product, PRODUCT.Label, PRODUCT.UnitPrice, TOTAL.Currency keep their initial status of attribute while all other attributes of the schema become entity types.

4.2.5 Ordering Definition

In the final step, a seq group is added to all entity types having more than one child. The current XML-specific transformation XML-ET-Create-SEQ-Group (Figure 5) arbitrarily adds an explicit notion of order between the different children of a same entity type. This order can be manually modified by the designer.

Example. seq groups have to be added in entity types ORDER and CATALOG.

5 CASE Support

DB-MAIN is a graphical, general-purpose, programmable, CASE environment dedicated to Database Application Engineering. Besides standard functions such as specification entry, examination and management, it includes advanced processors such as transformation tool-boxes, reverse engineering processors and schema analysis tools. In particular, DB-MAIN offers a rich set of transformational operators that allow developers to carry out database structure exportation to XML Schema in a systematic way. Another interesting feature of DB-MAIN is the Meta-CASE layer, which allows method engineers to customize the tool and to add new concepts, functions, models and even new methods. In particular, DB-MAIN offers a complete development language, *Voyager 2* [14], through which new functions and processors can be developed and seamlessly integrated into the tool. Further details on DB-MAIN can be found in [15]. In the limited scope of this paper, we describe some of the DB-MAIN assistants dedicated to database exportation in XML Schema only.

5.1 Extraction and Exportation Facilities

Database schemas can be extracted by a series of processors. These processors identify and parse the declaration part of the source texts, or analyze catalog tables, and create corresponding abstractions in the repository. Extractors have been developed for SQL, COBOL, CODASYL, IMS, RPG and XML data structures, among others. Additional extractors can be developed easily thanks to the *Voyager 2* environment. The GER PS schema is introduced in the DB-MAIN tool through such processors.

DB-MAIN includes code generators for various DDL standards. The *XML Schema generator* automatically derives the XML Schema Code from a GER XS. This *Voyager 2* program is based on the specialization rules presented in Section 2.

5.2 Schema Transformation

The processes described in this paper heavily rely on transformation techniques. For some fine-grained reasoning, precise transformations have to be carried out on individual constructs. This is a typical way of working in conceptualization activities. In other cases, all the constructs that meet a definite precondition have to be transformed (e.g., all the N-ary relationship types are transformed into entity types). Finally, some heuristics can be identified and materialized into a transformation plan. More precisely, the following three levels of transformation must be available.

- *Elementary transformations.* Transformation T is applied to object O. With these tools, the user keeps full control of the schema transformation. Indeed, similar situations can often be solved by different transformations; e.g., a multivalued attribute can be transformed in a dozen of ways.
- *Global transformations.* Transformation T is applied to all the objects of a schema that satisfy predicate P. Such transformations can be carried out through a processor that allows the analyst to define T and P independently. DB-MAIN offers some predefined global transformations, such as: *replace all many-to-many relationship types by attributes or replace all multivalued attributes by entity types*.
- *Transformation plans.* All the constructs of a schema that do not comply with a given model are processed through a transformation plan (Section 3). DB-MAIN offers a dozen of predefined model-based transformations including XML Schema (Section 4) translation and untranslation.

5.3 Reverse Engineering

DB-MAIN offers functions and processors that are specific to database reverse engineering. Besides the DDL code extractors mentioned above, two families of tools are available, addressing the main steps of the reverse engineering process.

Implicit construct elicitation is supported by a collection of processors that help analyze program code, schemas and data in order to find the intended semantics. Let us mention a dependency analyzer that detects and displays the dependencies between the objects (variables, constants, records) of a program, a program slicer and a foreign key discovery assistant. Data structure extraction can be performed in a reliable way thanks to the semantics preserving transformation toolset. Transformation scripts that implement specific heuristics can be quickly developed. A programmable schema analysis processor can be used to detect structural patterns and problematic constructs to be further processed.

6 Related Work

XML has received much attention as a language for the exchange of information on the Internet and there has been much work done on expressing XML data into a reference model and translating structured data models into XML.

Considering first the issue of expressing XML data in terms of a reference model, a common approach is to use the ER or UML models for representing XML or semi-structured data ([8] and [9]). However, due to the differences between ER (or UML) and XML models, this approach is not able to represent all the XML notions (e.g., ordering or built-in/derived data types and namespaces). Unlike these approaches, we use a high-level data model (GER) that supports the most important constructs of XML Schemas.

The issue of translation between structured data models and XML has also received considerable attention in the literature. SilkRoute [3] and XPERANTO [2] allow XML documents to be materialized from relational databases by a simple transformation mechanism. For the reverse process of translating XML into relational form, [4] considers the translation in both directions between XML DTD and the relational model. Commercial tools with more limited capabilities also exist, such as Oracle9i [16] or IBM DB2 [17].

The approach we have proposed in this paper is a general method for formally expressing translation between structured schemas and XML Schema and is not limited to any specific structured data model. Through the use of the GER model and the schema transformation operators, we have provided a transformation path from any legacy structured data models to XML. The specification of the legacy model and the XML model as GER specializations makes it possible to define mappings in terms of sequences of primitive transformations on the GER. The reversed sequence can be used to automatically translate query, data or updates posed on XML.

More generally, the transformational approach has long been proposed as a sound basis for database engineering and in database design in particular [20]. Other processes, such as conceptual normalization [21], optimization [22] and reverse engineering [23] have been modelled by transformation plans, notably to ensure semantics preservation. Transformations can be used to transform a schema from one model to another one, requiring specific operators for this couple of models. The use of a pivot model, such as the GER, allows us to develop fewer mappings in a multi-model environment, as well as to reuse operators from a model to others. The GER is a high level model, since it encompasses the constructs of a large family of models. Another approach, illustrated by [19], is based on low level models that comprise the intersection of such a family of models. Both approaches are discussed and compared in [7].

Finally, the use of database reverse engineering techniques is new in database-to-XML extraction domain. However this problem has been addressed in more traditional problems such as system renovation and migration, for which most contributions tackle specific problems or legacy technologies. The reader will find in [12] an in-depth bibliography for this domain.

7 Conclusion

Recent expansion of the Web has boosted the emergence of a new generation of applications, combining data and functionalities from various data sources and publishing them coherently in the Web. These applications generally rely on exchanging strongly structured data extracted from databases. Converting from a database model to a semi-structured format often implies a semantic loss, particularly when the meaning of the source data is not well understood, as is usual in legacy databases. Hence, the importance of a rich and expressive model to translate the semantics of both database and XML models.

The approach presented in this paper has put in light the non-deterministic aspect of the model translation process. The same legacy database can lead to different conceptual schemas, according to the precision required during the reverse engineering process. In addition, the same conceptual schema can lead to a large set of equivalent XML structures. Hence the need for more complex methods that take into account other, non-functional, criteria for target schema generation, and that have not been addressed in this paper.

Thanks to the genericity of the approach, extending the methodology and the tools to other source and target technologies is quite easy. Indeed, the new source model is described as a specialization of the GER, and, if needed, heuristics specific to the technology are translated into the conceptualization transformation plan. For new target models, such as other XML languages [24], the effort is quite similar.

The paper leaves aside the problem of data generation, that is, the automatic production of XML documents that comply with the XML Schema that has been computed. This process has been developed and is now supported by the DB-MAIN CASE tool, thanks to the analysis of the history of the transformations used to convert the database schema in XML Schema.

References

1. D. Florescu and D.Kossmann (1999), *Storing and querying XML data using an RDBMS*, IEEE Data Engineering Bulletin, Vol. 22, pp. 27-34.
2. M.J. Carey, D. Florescu, Z.G. Ives, Y. Lu, J. Shanmugasundaram, E.J. Shekita and S.N. Subramanian (2000), *XPERANTO: Publishing object-relational data as XML*, Proc. of WebDB, pp. 105-110.
3. M. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima and W.-C. Tan (2002), Silk*Route: A framework for publishing relational data in XML*, ACM Transactions on Database Systems (TODS), Vol. 27.
4. D. Lee, M. Mani, F. Chiu and W. W. Chu (2002), *NeT and CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints*, Proc. of the ACM International Conference on Information and Knowledge Management.
5. P. Rodriguez-Gianolli and J. Mylopoulos (2001), *A Semantic Approach to XML-based Data Integration*, in Proc. of the ER Conference.
6. F. Yergeau, J. Cowan, T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler (2004), *XML 1.1*, W3C Recommendation, Technical Report, W3C.

7. J.-L. Hainaut (2005), *Transformation-based Database Engineering*, Chapter in Transformation of Knowledge, Information and Data: Theory and Applications, P. van Bommel Editor, IDEA Group.
8. M.R. Jansen, Thomas H. Moller and T.B. Pedersen (2003), *Converting XML DTDs to UML Diagrams for Conceptual Data Integration*, Data and Knowledge Engineering, Vol. 44, pp. 323-346.
9. R. Conrad, D. Scheffener and J.C. Freytag (2000), *XML Conceptual Modeling Using UML*, Proc. of ER Conference, pp. 558-571.
10. W3C Working Group (2001), *XML Schema*, W3C Recommendation, Technical Report, W3C.
11. F. Estiévenart (2004), *XML Specific Predicates and Transformations*, Technical Paper, CETIC Research Center, Gosselies.
12. J.-L. Hainaut (2002), *Introduction to Database Reverse Engineering*, Technical Report, CS Department, University of Namur, http://www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf (last consult. Aug. 2005).
13. J-L. Hainaut, M. Chandelon, C. Tonneau and M. Joris (1993), *Contribution to a Theory of Database Reverse Engineering,* in Proc. of the IEEE WCRE Conference, pp. 161-170.
14. V. Englebert (2002), *Voyager 2 Manual*, DB-MAIN Series, Institut d'Informatique, University of Namur, http://www.db-main.be
15. J.-M. Hick, V. Englebert, J. Henrard, D. Roland and J.-L. Hainaut (2004), *The DB-MAIN Database Engineering CASE Tool (version 7) - Functions Overview*, Technical Report, University of Namur, http://www.db-main.be.
16. Oracle Corporation (2004), *Oracle XML SQL Utility*, Oracle Corporation, http://www.oracle.com (last consult. Feb 2005).
17. J. Cheng and J. Xu (2000), *IBM DB2 XML Extender*, Proc. of ICDE.
18. Ph. Thiran, F. Estiévenart, J-L. Hainaut and G-J. Houben (2004), *Exporting Databases in XML - A Conceptual and Generic Approach*, in Proc. of WISM (CAiSE'04).
19. P. McBrien and A. Poulovassilis (1998), *A General Formal Framework for Schema Transformation,* Data & Knowledge Engineering, 28(1), 47-71.
20. A. Rosenthal and D. Reiner (1988), *Theoretically sound transformations for practical database design,* Proc. of Entity-Relationship Approach.
21. O. Rauh and E. Stickel (1995), *Standard Transformations for the Normalization of ER Schemata,* Proc. of the CAiSE•95 Conf., Jyväskylä, Finland, LNCS, Springer-Verlag.
22. H.A. Proper and T.A. Halpin (1998), *Database Schema Transformation & Optimization*, Proc. of the 14 th International Conference on Conceptual Modeling, LNCS, 1021, 191-203, Springer.
23. J-L. Hainaut, M. Chandelon, C. Tonneau and M. Joris (1993b), *Transformational techniques for database reverse engineering,* Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, ER Institute.
24. L. Dongwon and W. C. Wesley (2000), *Comparative Analysis of Six XML Schema Languages*, http://www.cobase.cs.ucla.edu/tech-docs/dongwon/ucla-200008.html.