Improving Query Performance on XML Documents: A Workload-Driven Design Approach

Rebeca Schroeder
Informatics and Statistics Department
Federal University of Santa Catarina
Florianópolis/SC, Brazil 88040-900
rebecks@inf.ufsc.br

Ronaldo dos Santos Mello
Informatics and Statistics Department
Federal University of Santa Catarina
Florianópolis/SC, Brazil 88040-900
ronaldo@inf.ufsc.br

ABSTRACT

As XML has emerged as a data representation format and as great quantities of data have been stored in the XML format, XML document design has become an important and evident issue in several application contexts. Methodologies based on conceptual modeling are being tightly applied for designing XML documents. However, the conversion of a conceptual schema to an XML schema is a complex process. In many cases, conceptual relationships cannot be represented in a hierarchy so that they have to be represented by reference relationships in the XML schema. The problem is that reference relationships generate a disconnected XML structure and, consequently, produce an overhead cost for query processing on XML documents.

This paper presents a design approach for generating XML schemas from conceptual schemas considering the expected workload of the XML applications. Query workload is used to produce XML schemas which minimize the impact of the reference relationships on query performance. We evaluate our approach through a case study where a set of XML documents are redesigned by our methodology. The results demonstrate that query performance is improved in terms of the number of accesses generated by the queries on the XML documents designed by our approach.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: Schema and subschema

General Terms

Algorithms, Performance, Design

Keywords

Conceptual schemas, XML schemas, query performance

1. INTRODUCTION

In the last few years, the XML data model has been used as a storage format for several applications. For instance, XML documents have been directly stored in databases in their native format [9, 15]. Besides, XML documents have been more and more used as containers to store data that will be retrieved or updated by applications [12, 19]. Therefore, there is an increasing need for effective design methodologies to produce XML documents with an optimized structure to respond well to retrieval and updating operations of the applications.

XML document design has been considered in a three-level modeling approach [6, 12, 14, 19]. In the first level, a conceptual schema is generated for representing the application domain in a high-level abstraction. In the second level, the conceptual schema is translated into an XML logical schema which represents the hierarchical structure of the documents and their constraints. The logical schema is converted to an implementation schema in the last modeling level. XML schema definition languages like DTD [4] and XML Schema [18] are used to define the implementation schema. This top-down modeling approach is considered one of the best human-oriented ways to generate XML documents. It is easier for designers to specify and understand the concepts and relationships of the system in terms of conceptual models than in terms of XML models [3, 12].

The conversion of a conceptual schema to an XML schema is not a straightforward process. The most traditional conceptual models, like ER and UML represent schemas which can be undirected graphs with cycles. As XML schemas denote a tree structure, some conceptual relationships cannot be represented as a hierarchical relationship between XML elements. In these cases, a reference relationship must be established among the elements that represent the related concepts. Reference relationships are represented by *IDREF* or *keyref* definitions in the XML schema definition languages. Even though reference relationships are necessary in many cases, they generate a disconnected schema that often leads to a poor query performance [14, 19, 20]. In this paper, we present a conversion method which minimizes the impact of reference relationships in the performance of the application queries.

We propose a methodology for generating XML schemas from conceptual schemas considering the expected workload of the XML applications. Workload information is given by the designer in terms of the amount of element instances estimated for XML documents as well as of the main operations that will be performed over these documents. This information is used to determine an optimized structure in the XML schema, contributing to a better query performance of the application in general.

DocEng'08, September 16–19, 2008, São Paulo, Brazil.

We evaluate our approach through a case study where we redesign XML documents used by a purchase application. We obtain the conceptual schema of these documents through the bottom-up method proposed in [11]. Then, a new XML schema is obtained by applying our approach. We demonstrate that the XML structure generated by our approach reduces the amount of elements accessed in the application queries.

The remainder of this paper is organized as follows. Section 2 provides an overview of our conversion approach and describes the load data used by the process. The XML logical model of our conversion approach is also presented in Section 2. Section 3 presents the main contribution of this paper: a conversion algorithm for mapping conceptual constructs to suitable structures in the XML logical model considering the workload of an application. The case study is presented in Section 4 and related work is discussed in Section 5. Finally, Section 6 is dedicated to the conclusions.

2. FUNDAMENTALS

Our conversion approach starts with a conceptual schema and workload information given by the designer of the application, as shown in Figure 1. The conceptual schema is translated into an XML logical schema in the *Logical Design level*. The conversion process applied by this level is defined by a set of rules for converting each conceptual construct to an equivalent representation in the XML logical model. This logical model is an abstract model to represent the different XML implementation models. In the *Implementation Design level*, an XML logical schema is translated into a schema defined by a specific implementation model which can be, for instance, a specification in XML Schema or DTD. Since the XML schema was defined by the conversion approach, conforming XML documents can be created from the XML schema generated.

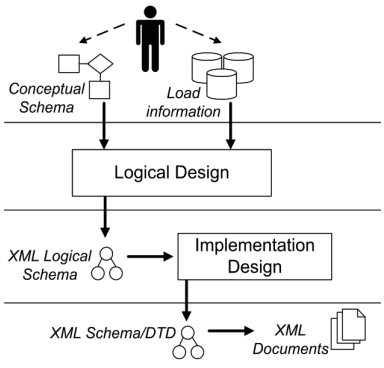


Figure 1: XML Document Design Approach

Even though the *Implementation Design level* is considered by our conversion approach, in this paper we focus on generating optimized XML structures from a conceptual schema in the *Logical Design level*. Besides, we consider that an XML logical schema generated by our approach can be converted to an implementation schema in a very direct way.

Our conversion approach adopts the EER (Extended Entity-Relationship) model [1] as the conceptual model given that EER is a suitable model for representing information concerning an application domain. Despite that, the EER model contains all the common constructs used by the conceptual modeling of the software design in general so that it can be considered a generic conceptual model. It means that our approach could be applied through other conceptual models like UML.

We assume that workload information given by the designer is estimated over a conceptual schema. This information will be used in the logical design for generating appropriate structures in an XML logical schema. We define the workload information modeling approach and the logical model applied by our approach as follows.

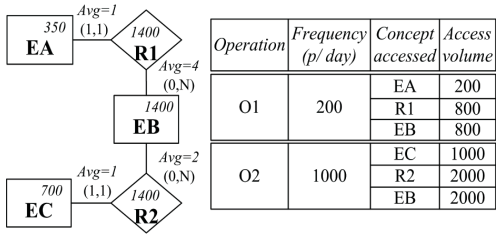
2.1 Workload Information Modeling

Load data of the application is estimated and modeled for providing information to the conversion process. This information allows our algorithm to choose an optimized XML structure to represent a conceptual schema fragment. At the end of the process, we obtain a suitable XML schema which can respond as efficiently as possible to the retrieval and updating operations considered in the workload.

Collecting information about the application load is a hard task, especially for large applications. On the other hand, it is necessary to know about it for making choices in the logical and in the implementation design levels [1, 20]. According to the authors of [1], we may concentrate on the important 20% of the operations that will be performed by the application. This assumption is rooted on the so-called 20-80 rule: 20% of the operations produce 80% of the application load.

We define an approach for modeling the main application load. It is a variant approach of the method proposed in [1] which presents a method for modeling database load. Two types of information are modeled to characterize the main load of the application: the volume of data and the description of operations.

Figure 2(a) shows an example of an EER schema augmented with volume of data. The volume of data is measured by the average number of instances estimated for each entity and relationship of an EER schema. This data is represented on an EER schema itself with the average number of instances embedded within the entities and relationships. For example, the average number of instances for the entity *EA* is 350. Besides, the average cardinality of each entity in a relationship is estimated. *Avg*=4 in Figure 2 (a) denotes that for each *EA* instance there are 4 *EB* instances related by *R1* on average.



Operation	Frequency (p/ day)	Concept accessed	Access volume
O1	200	EA	200
		R1	800
		EB	800
O2	1000	EC	1000
		R2	2000
		EB	2000

Figure 2: (a) EER schema with volume of data information, (b) operations estimated on this schema

An operation is an elementary interaction with the application, which includes retrieval or updating operations. For each operation given by the designer, there is a daily aver-

age frequency in which it is performed and a list of entity and relationship types involved. The order of the entity and relationship types in this list indicates the sequence in which they are accessed by the operation. Each concept (entity or relationship type) has a daily average number of instances that are accessed by an operation. Figure 2 (b) shows that the operation *O1* is performed 200 times a day. The entities and relationship *EA*, *R1* and *EB* are accessed, in this sequence, by *O1*. The access volume of the initial concept *EA* is 200 in *O1*. In the navigation sequence, the average number of instances of the concepts *R1* and *EB* is obtained by multiplying the access frequency of *EA* (200) by the average cardinality of *EA* in *R1* (*Avg*=4), i.e, *R1* and *EB* are accessed 800 times a day by *O1*.

Once the application volume and operations have been estimated, we proceed analyzing this information for generating data that will be used by our conversion algorithm. In order to choose a suitable structure for a concept in the XML schema, our algorithm needs to know the total number of instances accessed by all operations in which that concept is involved. We define this value as the *General Access Frequency* of a concept. For the operations in Figure 2 (b), the *General Access Frequency* of *EB* is 2800 because *EB* is accessed 800 times a day by *O1* and 2000 times by *O2*.

During the conversion process, the algorithm frequently needs to know if the *General Access Frequency* of a concept is relevant/irrelevant for the load of the application. For this purpose, we compare the *General Access Frequency* of the concept with a *Minimal Access Frequency* which is established at the beginning of the process. In order to obtain the *Minimal Access Frequency*, a value corresponding to the *Total Access Frequency* of all operations is measured. The total access is given by the sum of the *General Access Frequency* of all entities and relationships of the schema. A percent value representing the minimal access of the application over the total access is given by the user. Then, this percent value is applied over the *Total Access Frequency* value for obtaining the *Minimal Access Frequency*. According to the Figure 2 (b), the *Total Access Frequency* of the schema is 6800. Assume, for instance, that the user provides 1% as the value for denoting the minimal value related to the total access. Then, we apply this percent value on 6800 and we obtain 68 as the *Minimal Access Frequency*.

In order to briefly exemplify the application of these concepts in our approach for generating an XML schema, consider the conceptual schema and the load information provided in Figure 2. The cardinality of the relationship *R1* and the functional dependency of $EB \rightarrow EA$ determine that it is possible to represent *EB* as a child element of *EA* in an XML schema. However, we can also nest *EB* as a child of *EC* analyzing the *R2* relationship. We argue, by analyzing load data information for the XML documents, that we can determine, for example, the most convenient parent to the *EB* concept. This analysis is performed by our conversion approach through the workload information presented in this section.

2.2 XML Logical Model

We propose an XML logical model to represent the XML data model. Our conversion approach generates XML schemas defined by this logical model in the *Logical Design* level. The XML logical model is an abstract representation for different XML implementation models, like the W3C recommendations (DTD and XML Schema). Such logical model is a hierarchical model that supports the representation of the XML constructs and constraints.

Figure 3 presents an XML logical model instance. An XML logical schema is composed by attributes, elements and relationships. There are three types of attributes: *simple*, *identifier* and *reference* attribute. A *simple* attribute models an element property which is defined by a single value, like *attribute1*. An *identifier* attribute is an attribute that is part of an element identifier, like *IDattribute1*. The set of identifier attributes in an element forms the element identifier. A *reference* attribute is an attribute that refers to an element identifier, like *REFattribute1* that refers to *ComplexElement3*.

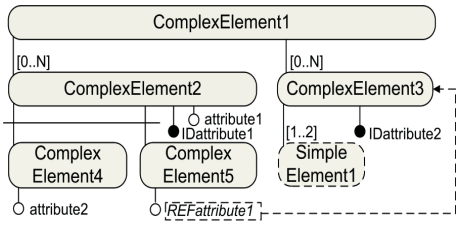


Figure 3: Example of an XML logical schema

Elements are classified as *simple* or *complex*. A *simple* element models information that is defined by a simple data type. It does not have attributes and it can be multi-valued, like *SimpleElement1* with minimum occurrence of 1 and maximum occurrence of 2. A *complex* element models information that is composed by elements and/or attributes. The component elements of a complex element are organized by one of two order constructs: *ordered* or *exclusive*. An *ordered* construct defines n ordered component elements, with $n \geq 1$. An *exclusive* construct defines n alternatives for the component elements, with $n > 1$. The default order construct of a complex element is *ordered* and the *exclusive* construct is represented by a line that crosses the component elements of a complex element. *ComplexElement1* and *ComplexElement2* are examples of complex elements with *ordered* and *exclusive* constructs, respectively.

The logical model supports two types of relationships: *hierarchical* relationship and *reference* relationship. A *hierarchical* relationship is represented by a line between a source concept and a target concept. A *hierarchical* relationship defines the minimum and maximum occurrence of a target concept in a source concept. For instance, *ComplexElement1* may have zero or more *ComplexElement2* instances. The default minimum and maximum occurrence for target concepts is 1. A *reference* relationship is represented by a dashed line between a reference attribute (or a set of reference attributes) and a complex element. A *reference* relationship specifies a set of reference attributes which refer to the identifier of other complex element, like the relationship between *REFattribute1* and *ComplexElement3*.

As the XML logical model is an abstract representation for the XML implementation models, an XML logical schema can be converted to a DTD [4] or XML Schema [18] specification in a direct way. Only a few decisions must be made in the conversion of the identifier and reference attributes because DTD and XML Schema provide different support for representing these constraints.

3. THE CONVERSION APPROACH

Our conversion approach is based on conversion rules for mapping EER constructs to equivalent XML compositions. For explanation purposes, we separate the conversion process in two parts: conversion of generalization types and conversion of relationship types. In both parts, a procedure is given for applying the conversion rules over an EER schema. Load information is used by these procedures for generating well-structured XML logical schemas. At the end of this section, we provide an algorithm which covers these procedures.

Figure 4 illustrates the application of the conversion rules and of the algorithm. Consider that load information was previously modeled so that the *General Access Frequency* of the concepts is presented in Table 1. The *Minimal Access Frequency* is defined as 130 accesses per day. The *General Access Frequency* of the concepts omitted by Table 1 is considered zero. It means that these concepts are not accessed by the operations which represent the main load of the application. However, they cannot be disregarded and must be represented in the XML schema.

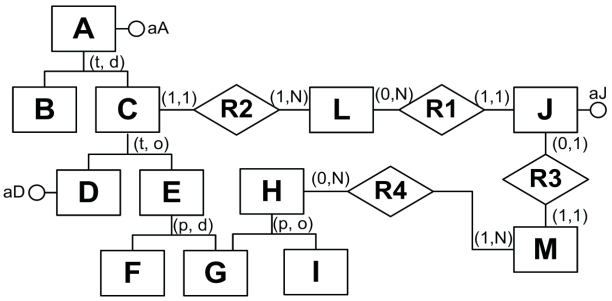


Figure 4: An Example of EER schema

Table 1: *General Access Frequency* (GAF) of the concepts of Figure 4

Concept	GAF	Concept	GAF
A	70	L	540
B	200	M	320
C	90	R1	100
E	100	R2	540
H	50	R4	320

3.1 Conversion of Generalization Types

A generalization hierarchy in the EER model defines a subset relationship between a generic entity, namely superclass, and one or more specialized entities, namely subclasses. The disjointeness and completeness constraints that can be applied on the subclasses establish four possible constraints on generalization types: total and disjoint (t, d); partial and disjoint (p, d); total and overlapping (t, o); and partial and overlapping (p, o) [1, 17].

As shown in section 2.2, our XML logical model does not consider specific constructs of XML schema languages. Thus, we proceed to represent generalization types by XML elements, attributes and relationships. In this section, we define alternative rules to convert generalization hierarchies from EER schemas to XML logical schemas and a procedure which selects the suitable rule to be applied on each EER generalization type.

3.1.1 Conversion Rules

We provide three alternatives to convert generalization types. The difference among these alternatives is given by the different size of XML schemas that each one generates and the constraints on generalization types that they can represent.

The conversion strategy defined by *Rule 1* generates only one XML element from a generalization hierarchy. The XML element is created to represent the superclass and its attributes as well as the attributes of the subclasses. Subclass attributes are treated as optional in the content model of the superclass element. On applying this rule, we assume that the subclass attributes will act as discriminating attributes to identify an instance of a subclass in the XML documents. The subclasses which were already converted become child elements of the element generated by this rule.

Rule 1. The conversion of a generalization type G proceeds as follows:

1. given an entity E_{sp} defined as the G superclass, generate a complex element ce_{sp}. The attributes of E_{sp} are defined as attributes in ce_{sp};
2. given $\{E_{sb1}, E_{sb2}, ..., E_{sbn}\}$ the set of E_{sp} subclasses, for each E_{sbi} $(1 \leq i \leq n)$ do:
   - **IF** E_{sbi} was not converted, define the attributes of E_{sbi} as optional attributes in ce_{sp};
   - **ELSE** let ce_{sbi} be the element that represents E_{sbi} and generate a hierarchical relationship from ce_{sp} to ce_{sbi} where the occurrence of ce_{sbi} in ce_{sp} is defined as [0..1].

Another alternative generates only XML elements for the subclasses, and the superclass attributes are reproduced into each subclass element. This alternative is defined by *Rule 2*.

Rule 2. Given an entity E_{sp} defined as the superclass of a generalization type and $\{E_{sb1}, E_{sb2}, ..., E_{sbn}\}$ the set of subclasses of E_{sp}, generate a complex element ce_{sbi} for each subclass E_{sbi} $(1 \leq i \leq n)$ and define the attributes of the E_{sbi} and E_{sp} as attributes in ce_{sbi}.

In the alternative defined by *Rule 3*, the superclass and subclasses are explicitly represented by a complex element. Hierarchical relationships are established among the superclass and subclass elements to represent the relationship. Disjointness constraints on a generalization type are represented by the order construct of the superclass element. Completeness constraints are represented by the minimum and maximum occurrence of the subclass elements in the superclass element.

Rule 3. The conversion of a generalization type G proceeds as follows:

1. given an entity E_{sp} defined as the G superclass, generate a complex element ce_{sp} with the order construct defined as *exclusive*, if G is a disjoint generalization; or *ordered*, otherwise. The attributes of E_{sp} are defined as attributes in ce_{sp};
2. given $\{E_{sb1}, E_{sb2}, ..., E_{sbn}\}$ the set of E_{sp} subclasses, for each E_{sbi} $(1 \leq i \leq n)$ do:
   - **IF** E_{sbi} was not marked, generate a complex element ce_{sbi} and a hierarchical relationship from ce_{sp} to ce_{sbi} where the occurrence of ce_{sbi} in ce_{sp} is defined as ([0-1],[1]), depending on the completeness constraint of G (total or partial);
   - **ELSE** let ce_{sbi} be the element that represents E_{sbi}, generate an optional reference attribute ra_{sbi} in ce_{sp} which refers to ce_{sbi}.

Rule 3 is able to convert multiple-inheritance cases. This treatment is defined in the second step of this rule. The subclasses that were processed as child elements or attributes in other elements are marked as processed by the procedure

GeneralizationConversion (see section 3.1.2). Thus, the generalization types with marked subclasses are established by reference attributes in multiple-inheritance cases. An example of the application of this rule on a multiple-inheritance case is shown in next section.

3.1.2 Conversion Procedure of Generalization Types

Notice that more than one conversion rule can be applied to convert a generalization type. We propose the procedure *GeneralizationConversion* for choosing the appropriate rule for converting each generalization type of a conceptual schema. Each generalization type is converted by analyzing the load data and the constraints over the generalization case. The procedure establishes a conversion order in which the conceptual fragments involving generalization types must be converted. A bottom-up conversion is performed when there are multiple-level hierarchies, i.e., the generalizations are converted from the bottom to the top of the hierarchy. Besides, when there is a multiple-inheritance case, the superclass with the highest *General Access Frequency* is converted first. It means that the superclass that is most frequently accessed is made the parent element of an element that represents a subclass with more than one superclass. In this case, the remaining superset relationships are represented by reference attributes as defined in *Rule 3*. Generalization types involved in multiple-inheritance cases are always converted by *Rule 3*.

Procedure 1 *GeneralizationConversion*

Input: An EER Schema with load data information: E
The *Minimal Access Frequency* of E: min
Output: Fragments of an XML logical schema: XG

Let G be the set of generalization types $\{g_1, g_2, .., g_n\}$ of E
Order G so that the generalization types at the bottom of the hierarchy with superclasses that have highest *General Access Frequency* appear first. Generate the list $G'=\{g_1,.., g_n\}$
for all $g_i \in G'$ $(1 \leq i \leq n)$ with superclass E_{sp} and subclasses $\{E_{sb1},.., E_{sbn}\}$ **do**
  if (there are no marked subclasses in g_i) AND (all subclasses in g_i have *General Access Frequency* lower than min) AND (there are no subclasses with more than one superclass) **then**
    Apply *Rule 1* and mark all the subclasses of g_i
  else if (The *General Access Frequency* of E_{sp} is lower than min) AND (there are no subclasses with more than one superclass) **then**
    Apply *Rule 2* and mark E_{sp}
  else
    Apply *Rule 3* and mark all the subclasses of g_i
  end if
end for

Since the conversion order of the generalization types was established, we proceed applying the conversion rules for generalization types (Rule 1, 2 or 3) and verifying the preconditions of each one, so that the rules which generate the smallest XML logical fragment are verified first. For *Rule 1* and *Rule 2*, we verify if the *General Access Frequency* of the entities that will by omitted is lower than the *Minimal Access Frequency* (min). If the *General Access Frequency* is higher than min, it means that these entities participate in frequent operations and the distinction between superclass and subclasses must be preserved. The last option to convert generalization types is *Rule 3*.

In order to exemplify the conversion of generalizations, consider the generalization types of Figure 4 and the load information of Table 1. Figure 5 presents the elements generated to represent the generalization types of Figure 4. As the *General Access Frequency* of the entity E (100) is higher than H (50) and both entities are involved in a multiple-inheritance case, the generalization type involving E as superclass is converted first. Thus, this generalization type is converted by *Rule 3* and the subclasses F and G are represented as child elements of the element E. In the sequence, the generalization type involving H as the superclass is also converted by *Rule 3*. However, as G was marked by E, the relationship between H and G is established by a reference attribute ($gRef$) in the element H.

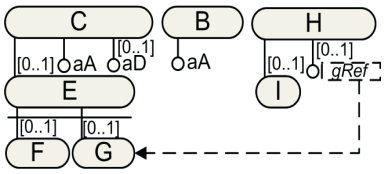


Figure 5: Conversion of generalizations of Figure 4

The generalization type where C is the superclass must then be converted, considering our bottom-up analysis of multi-level hierarchies. *Rule 1* is applied and the attribute of the subclass D is represented as an attribute in C. Also, the element E becomes a child element of C. The top generalization type involving the superclass A is converted by *Rule 2* and the attributes of A are reproduced in the elements C and B. In this case, the first alternative (*Rule 1*) cannot be applied because the *General Access Frequency* of the subclass B (200) is higher than the *Minimal Access Frequency* (130). Thus, *Rule 2* is applied because all the conditions for applying it are guaranteed.

Even though specific rules to convert union types [7] are not provided in this paper, these conceptual constructs can be converted by similar rules for converting generalization types. A discussion about the conversion of generalization and union types can be found in [16].

3.2 Conversion of Relationship Types

A relationship type is a common conceptual construct which establishes a correspondence among two or more entities [1, 17]. The cardinality of relationship types is the main constraint which can be considered on the conversion to an XML structure. In this section, we present three rules for converting relationship types and its constraints as well as a procedure to perform the conversion rules on the relationship types of an EER schema.

3.2.1 Conversion Rules

In general, each conversion rule of relationship deals with a specific constraint case on relationship types. *Rule 4* is applied only for 1:1 relationship types, *Rule 5* only for 1:N relationships, and *Rule 6* is applied for relationships with cardinality N:N, n-ary where n>2 or in cases of optional participation of the entities of 1:1 and 1:N relationships.

Rule 4 generates only one complex element to represent the relationship type and its related entities. *Rule 5* generates complex elements for each related entity, where one of them is nested to another and the relationship attributes are appended in the nested element. The last rule (*Rule 6*) generates independent elements for each entity of a relationship type and reference relationships are established among the elements generated. The conversion rules are defined as

Procedure 2 *RelationshipConversion*

Input: An EER Schema with load data information: E
The *Minimal Access Frequency* of E: *min*
Fragments of an XML logical schema generated by *GeneralizationConversion*: *XG*
Output: Fragments of an XML logical schema: *XR*

Let R be the set of relationship types $\{r_1,..,r_n\}$ of E
Order R so that the relationship types with the highest *General Access Frequency* appear first. Generate the list $R'=\{r_1,.., r_n\}$
repeat
  Select the first unmarked relationship r_i of R'
  Let $\{E_1,.., E_n\}$ be the set of entities related by r_i, such that:
  if (r_i is binary) AND (there is an unmarked entity E_i with participation (1,1) in r_i) **then**
    E_2 is E_i and E_1 is the another entity of r_i
  else
    E_1 is the entity that has the highest *General Access Frequency* in r_i
  end if
  if (r_i is 1:1) AND (the participation of E_2 is (1,1)) AND (E_2 is unmarked) **then**
    Apply *Rule 4* and mark E_2
  else if (r_i is 1:N) AND (the participation of E_2 is (1,1)) AND (E_2 is unmarked) **then**
    Apply *Rule 5* and mark E_2
  else
    Apply *Rule 6*
  end if
  Mark r_i
until all relationship types of R' have been marked

follows and an application example is presented in the next section.

Rule 4. Given a 1:1 relationship type R which relates the entities E_1 and E_2, generate a complex element ceE_1 and define the attributes of E_1 as attributes in ceE_1. Define the attributes of E_2 and R as attributes in ceE_1.

Rule 5. Given a 1:N relationship type R which relates the entities E_1 and E_2, generate a complex element ceE_1 and define the attributes of E_1 as attributes in ceE_1. Generate a complex element ceE_2 for representing E_2 and make it be a child element of ceE_1. The occurrence of ceE_2 in ceE_1 is defined as ([0-1],[N]), depending on the participation of E_1 in R (optional or mandatory). Define the attributes of E_2 and R as attributes in ceE_2.

Rule 6. Given a relationship type R and $\{E_1, E_2,.., E_n\}$ the set of entities related by R, the conversion of R proceeds as follows:

1. For each E_i ($1 \leq i \leq n$) generate a complex element ceE_i and define the attributes of E_i as attributes in ceE_i
2. IF R is a binary relationship without attributes AND the participation of E_1 in R is defined as ([0-1], 1), generate a reference attribute in ceE_1 referring to the identifier of ceE_2;
3. ELSE generate a complex element ceR as a child element of ceE_1 and define the attributes of R as attributes in ceR. For each E_i ($1 < i \leq n$) generate a reference attribute in ceR referring to the identifier of ceE_i.

3.2.2 Conversion Procedure of Relationship Types

The procedure *RelationshipConversion* is proposed to apply the conversion rules on the relationship types of an EER schema. The procedure orders the relationship types so that relationships with the highest *General Access Frequency* appear first. This order is established to give priority to the relationships that represent the largest impact on the workload of the application. Then, if there is more than one nesting possibility for an entity type considering all the relationship types in which it participates, we process this entity by the relationship with the highest *General Access Frequency* first.

For converting a relationship type, we first determine which of the entities will be the entity on the top of the hierarchy in the XML fragment. When the relationship type is binary and there is an entity with participation (1,1) in the relationship, the top-entity is the other entity of the relationship type. In other situations, the top-entity is the entity type with the highest *General Access Frequency* in the relationship, i.e., we assume that the relationship type is more frequently accessed through this entity in the navigation schema of the operations considered.

Figure 6 presents the XML logical schema generated by applying the procedure on the relationship types of Figure 4. According to Table 1, the relationship type with the highest *General Access Frequency* is *R2*(540) followed by *R4*(320), *R1*(100) and *R3*(0). Thus, *R2* is converted first and the entity L becomes a child element of C by applying *Rule 5*. In the next step, *R4* is converted by applying *Rule 6*. As *R4* is a N:N relationship, a reference relationship must be established between H and M. The entity M is selected as the top-entity of the relationship because its *General Access Frequency* (320) is higher than H (50). On applying *Rule 6*, an element to represent *R4* is created as a child element of M. Then, a reference attribute is created in the element *R4* referring to the identifier of the element which represents H.

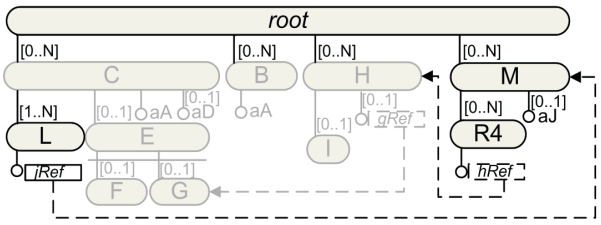


Figure 6: Conversion of relationships of Figure 4

The relationship *R1* involving the entities J and L must be converted as a reference relationship because the entity L was previously marked as a child element of C. The relationship is established from L to J because the *General Access Frequency* of L is higher than J. Notice that the reference attribute *jRef* refers to element M instead of element J. It occurs because in the next step the 1:1 relationship *R3* is converted by *Rule 4*. It means that the entity J is represented in the content model of element M.

Observe that the fragments of the XML schema generated by the conversion of the generalization types of Figure 4 are also shown in Figure 6. The algorithm which defines the order for performing the procedures and produces the final XML schema is presented in the next section.

3.3 Conversion Algorithm

An algorithm namely *EER-XML* is proposed to perform the procedures defined, generating the final XML logical schema. The generalization types are converted first, followed by the conversion of the relationships. The fragments generated by performing the procedure *GeneralizationConversion* are considered and kept by the procedure *RelationshipConversion*. At the end of the process, the root element of the XML schema is defined. If there is only one element that does not have a parent element, this element is set as the *root* element of the XML logical schema. Otherwise, an element is created to represent the *root* element of the schema. In this case, all the elements which are not target elements in the hierarchical relationships become child elements of the *root* element. Finally, Figure 6 presents the final XML schema generated by applying *EER-XML* on the

conceptual schema of Figure 4 and the load information of Table 1. An element was created to represent the *root* element of the XML logical schema, as shown in Figure 6.

Algorithm 1 *EER-XML*

Input: An EER Schema with load data information: E
The *Minimal Access Frequency* of E: min
Output: An XML logical schema: X

Perform Procedure *GeneralizationConversion*
Perform Procedure *RelationshipConversion*
$XF \leftarrow$ Fragments generated by the procedures
Let CE=$\{ce_1,..,ce_n\}$ be the set of complex elements of XF which do not have parent elements
if (n == 1) **then**
  Make ce_1 be the root element of X
else
  Generate a complex element ce_r to be the root element of X
  Make all elements of CE be child elements of ce_r
end if

4. CASE STUDY ANALYSIS

This section presents a case study to validate our conversion methodology. The goal of this experiment is to show that our method can improve query performance on XML documents by reducing the number of access of XML elements. Existing XML documents were redesigned by our approach in order to compare the number of accesses generated by queries over the initial schema and over the redesigned schema. The experimental approach and the results are presented as follows.

4.1 Experimental Approach

We apply part of the bottom-up methodology proposed in [11] to generate a conceptual schema from existing XML documents. A semi-automatic process for the integration of XML schemas namely *BInXS* is proposed in [11]. XML documents which are compliant to different schemas are taken by *BInXS* from a given domain. This approach then generates a unified conceptual schema for representing all these XML documents. For experimental purposes, we apply this method only for obtaining the conceptual schema of a set of XML documents that are compliant to the same XML schema. In the first step, *BInXS* generates an XML Schema specification for representing the set of XML documents. Then, the XML Schema specification is transformed into a conceptual schema represented in an OWL [2] document. The conceptual schema is defined in a canonical conceptual model which can be transformed into an EER schema in a straightforward way. Despite the conceptual schema, the OWL document holds the element instances of the original XML documents. These instances are used to measure the volume of data in our case study, i.e., the average number of instances of the entities and relationships as well as the average cardinality of the entities in each relationship type.

We take a set of XML documents from a purchase system. We obtain an XML schema by applying the first step of *BInXS* on the set of XML documents. All the documents are compliant to an XML Schema which is shown in Figure 7 in the XML logical format. We omit most attributes to simplify the schema. In the next step, *BInXS* generates a conceptual schema which is a conceptual representation of the XML schema generated in the previous step. This conceptual schema as well as the volume of data are shown in Figure 8.

Despite the conceptual schema and the volume of data given by *BInXS*, it is necessary to obtain the operation load of the application. The operation data was given by an expert user in the application considering the concepts (entity and relationship types) defined in the conceptual schema. A set of seven operations was given. It represents the main load of this application according to the user. The third column of Table 2 presents the operation load in terms of access frequency of each concept of the conceptual schema. The *General Access Frequency* of each concept involved in the operations on the conceptual schema was measured and it is shown in Table 3. We omit the *General Access Frequency* of concepts which were not used in the operations considered. The *Minimal Access Frequency* was given by 100 accesses.

We apply our methodology on the conceptual schema and on the load data informed by the user. The XML logical schema generated by our approach is shown in Figure 9. In order to evaluate our approach, we measure and compare the access frequency generated by each concept applying the operations on the original XML schema and on the XML schema generated by our approach. The access frequency generated by the original and the redesigned XML schema is shown in the two last columns of Table 2. We analyze the access generated by each XML schema in the next section.

4.2 Experimental Results

Analyzing the last line of Table 2, we verify that the *Total Access Frequency* increases considerably by comparing the access generated on the conceptual schema with the original and redesigned schemas. Basically, this increase is the consequence of reference relationships used to represent a few conceptual relationships in the XML schemas.

The number of the elements in the redesigned schema is smaller than in the original schema. One of the reasons for this reduction is that *Rule 1* was applied to convert the generalization types involving the superclasses *Order* and *Piece*. This rule was applied because the *General Access Frequency* of the subclasses is lower than the *Minimal Access Frequency* assumed.

A different nested structured is presented in the redesigned schema. This new structure is the reason for the reduction of the access frequency on the redesigned schema. This reduction can be seen comparing the two last columns of Table 2. Our algorithm gives preference to the conversion of the relationship types that have the highest *General Access Frequency*. For instance, consider the entity *ItemPR* and the minimum and maximum occurrence of this entity in the relationships *request* and *markedItem* in Figure 8. This entity could be converted as a child element of the *PurchaseRequisition* element or of the *Piece* element. As the *General Access Frequency* of *markedItem* relationship (750) is higher than *request* relationship (500), the entity *ItemPR* is converted first by *markedItem* so that the *ItemPR* is represented as a child element of *Piece*. The relationship *request* is converted in a subsequent step and a reference relationship is established between *PurchaseRequisition* and *ItemPR*.

Again, as shown in Figure 7 and Figure 9, the *ItemPR* concept was represented as a child element of *PurchaseRequisition* element in the original schema and as a child element of *Piece* element in the redesigned schema. These different representations generate different access frequencies on the operations *O1* and *O7* as is shown in Table 2. In

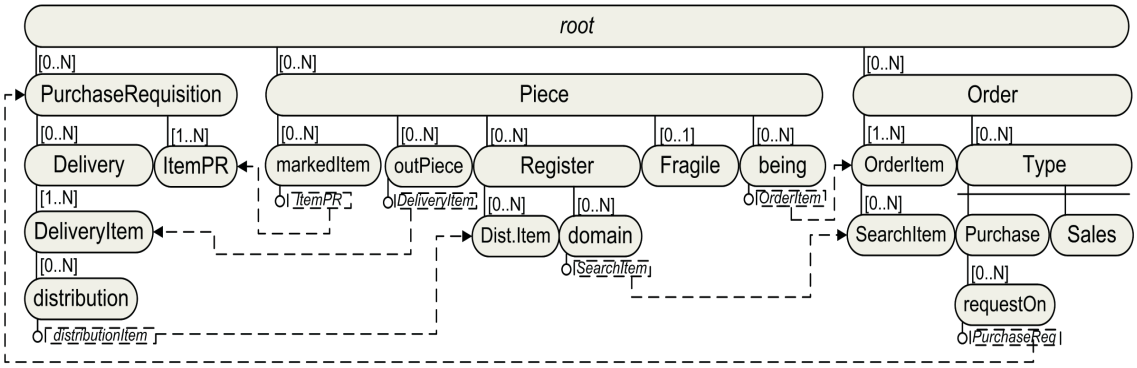


Figure 7: The original XML schema

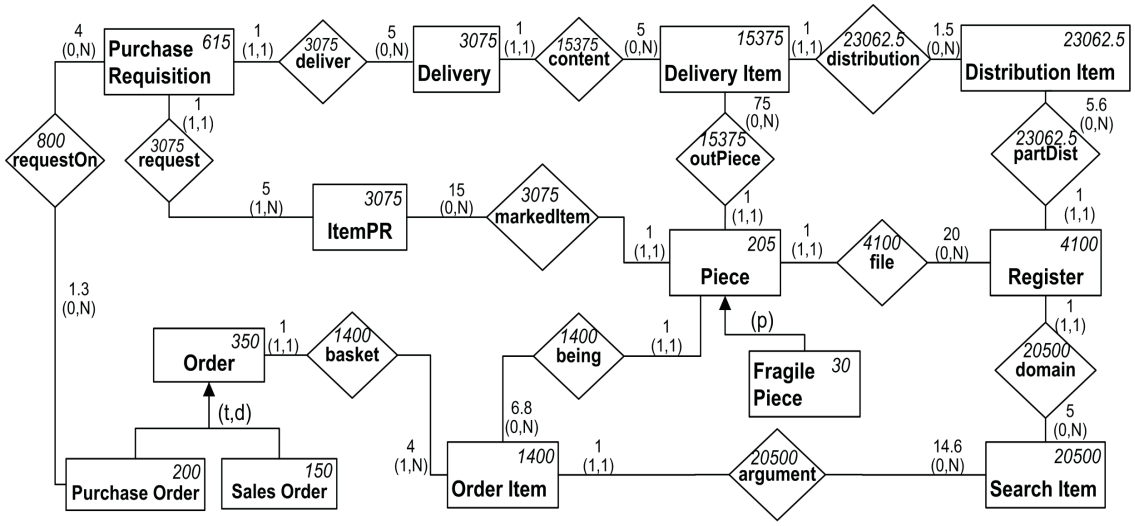


Figure 8: The EER schema generated by *BInXS*

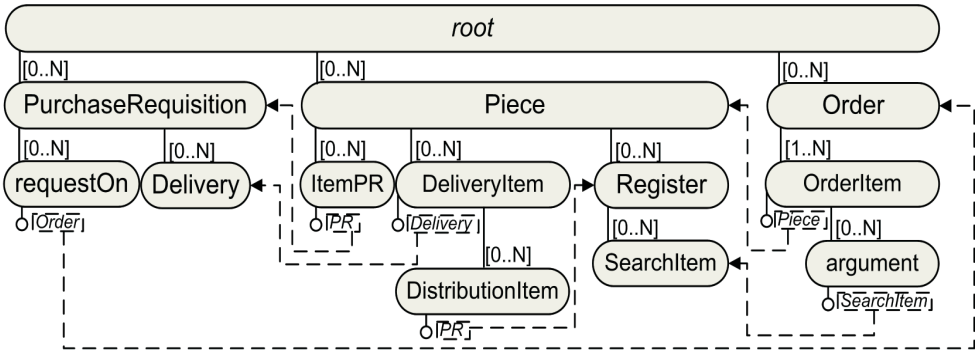


Figure 9: The redesigned XML schema

Table 2: Operations on schemas

#	Concepts	Access Frequency on...		
		Conceptual schema	Original XML schema	Redesigned XML schema
O1	Piece	50	50	50
	markedItem	750	750	-
	itemPR	750	2306250	750
	Total:	1550	2307050	800
O2	Piece	100	100	100
	outPiece	7500	7500	-
	DeliveryItem	7500	115312500	7500
	Total:	15100	115320100	7600
O3	Delivery	50	50	50
	content	250	-	768750
	DeliveryItem	250	250	250
	distribution	375	375	-
	Dist.Item	375	8648437.5	375
	Total:	1300	8649112.5	769425
O4	Piece	150	150	150
	file	3000	-	-
	Register	3000	3000	3000
	domain	15000	15000	-
	SearchItem	15000	307500000	15000
	Total:	36150	307518150	18150
O5	Order	60	60	60
	basket	240	-	-
	OrdemItem	240	240	240
	being	240	336000	-
	Piece	240	240	49200
	Total:	1020	336540	49500
O6	PR	60	60	60
	requestOn	78	48000	78
	Purch.Order	78	78	-
	Order	78	78	27300
	basket	312	-	-
	OrderItem	312	312	312
	Total:	918	48528	27750
O7	PR	100	100	100
	request	500	-	-
	ItemPR	500	500	307500
	Total:	1100	600	307600
TOTAL:		***57138***	***434180080.5***	***1180825***

O1, the original schema generates 2306250 accesses on the element *ItemPR* because it is necessary to compare the 750 values of the reference attribute in *markedItem* with all the instances of *ItemPR* element (3075). It does not occur to perform *O1* on the redesigned schema because the *ItemPR* element is represented as a direct child of the *Piece* element. In this case only 750 accesses are necessary to achieve the *ItemPR* element in *O1*.

As the relationship *request* is represented as a reference relationship in the redesigned schema, the access increases for *O7* to achieve the *ItemPR* element if compared with the access generated by the original schema. However, the lowest access generated by nesting *ItemPR* in *Piece* in *O1* overcomes the number of accesses generated by the redesigned schema in *O7*.

The nested structure of the entities *DeliveryItem*, *DistributionItem* and *SearchItem* in the redesigned schema is also different from the original schema. The impact of these different structures in the operations can be checked by comparing the access generated by the original and redesigned schemas.

Another relevant differential is related to the reference relationships. The conceptual relationship *being* is represented as a reference relationship for both the original and the redesigned schema. However, the reference attribute is located as an attribute of the *OrderItem* in the redesigned schema, and as an attribute of the *being* element in the content model of the *Piece* element in the original schema. The

Table 3: *General Access Frequency* (GAF) of the concepts of Figure 8

Concept	GAF	Concept	GAF
SearchItem	15000	domain	15000
DeliveryItem	7750	outPiece	7500
Register	3000	file	3000
ItemPR	1250	markedItem	750
OrdemItem	552	basket	552
Piece	540	request	500
DistributionItem	375	distribution	375
PR	160	content	250
Order	138	being	240
PurchaseOrder	78	requestOn	78
Delivery	50		

Algorithm *EER-XML* establishes the reference relationship from the element that represents the entity with the highest *General Access Frequency*, in this case the *OrderItem* entity. This is an appropriate treatment because in most practical cases, the entities with the highest *General Access Frequency* in a relationship also have the greatest number of instances. Thus, we can minimize the number of accesses comparing the values of a reference attribute with all the instances of the entity which has the smallest number of instances in the relationship. For example, we generate the lowest access frequency for achieving the Piece instances in *O5* comparing the 240 instances of the reference attribute in *OrderItem* with all the 205 instances of the *Piece* element. Instead, the original schema compares the 240 instances of the *OrderItem* element with all the 1400 instances of *being* for achieving the *Piece* instances. It generates the highest access frequency (336000) in *O5*.

The number of reference relationships generated by our approach in this case study is the same as those of the original schema. However, we can observe in Table 2 that the *Total Access Frequency* generated by applying the operations on the original schema is reduced by the redesigned schema. It means that our approach does not reduce the number of reference relationships but it minimizes the impact of these relationships, generating a lower diary access frequency for the whole schema. Hence, query performance will be reduced by applying the operations on XML documents compliant to the XML schema generated by the Algorithm *EER-XML*.

5. RELATED WORK

There are several approaches that deal with the conversion of a conceptual model to an XML model. We analyze approaches that consider the main constructs of a conceptual model in their conversion process. Algorithms which generate schemas in the XML Schema language are proposed in [10] and [13]. Conversion approaches to translate conceptual schemas into DTD schemas are presented in [5] and [8]. All this work determines the XML nested structure by a set of entities which are converted as *top-level-elements* in the XML schemas. *Top-level-elements* are identified by a classification of prevalent entities in [10]. However, the method to obtain these entities is not presented. In [5] and [13], these entities are selected according to their cardinalities in the relationship types. In [8], the designer determines which the most relevant entities to be converted as *top-level-elements* are. However, for most practical cases, a considerable number of entities is converted as top-level-element in [5], [8] and

[13], generating many reference relationships for representing the conceptual relationships involving these entities.

Alternative conversion rules are not supplied by the most related algorithms. Despite that, some constructs on generalization types are missed, mainly in [10] and [13]. The set of conversion rules applied by *EER-XML* attends all the possible constructs and constraints on generalization types in a conceptual model, including union types, multiple inheritance and multiple-level hierarchies. Besides, some of the related algorithms are strongly tied to an XML schema language. Our approach provides conversion rules for converting all the conceptual constructs to an abstract representation of the XML schema languages.

The approaches presented by [12] and [19] are focused, essentially, on generating optimized XML structures from a conceptual model. Compact and redundancy-free XML documents are obtained from analysis of the constraints of a schema defined by a generic conceptual model in [12]. However, generalization types are not considered in this conversion approach. These conceptual constructs are not considered in [19] either. A methodology to maximize the connectivity between entities related by relationships in an ER schema is proposed by [19]. For achieving this connectivity, they generate an XML schema with *multi-colors*, where each *color* is used to represent one aspect of the complete XML schema. The main problem with this approach is that it is necessary to extend XML schema definitions and XML query languages for supplying this *multi-color* approach. Above all, an overhead is generated on query performance to consider these multiple *colors*.

6. CONCLUSION

This paper provides a methodology for designing XML schemas from conceptual schemas and load information. In our approach, an EER schema is translated into an XML schema defined by an XML logical model. The XML logical model is an abstract model to represent the most common XML schema languages. Load information is considered to generate optimized XML structures in terms of the main load of an application.

The volume of data and the operations estimated lead our algorithm to an appropriate conversion rule for translating a specific conceptual fragment into an XML construction. Despite that, load data is also used to determine the best nested structure for the concepts of the conceptual schema in the XML schema. A case study presented in this paper demonstrates that our design approach can improve the query performance on XML documents by reducing the access frequency of the elements of the XML schema. This reduction depends on the possible advances which can be made in an XML schema. Hence, the improvements achieved by our methodology depend on the conceptual schema and the load information provided by an expert user.

As future work, we are exploring the possibility to reduce the number of reference relationships on XML schemas when a relationship is frequently accessed through entities which have a low access frequency in updating operations. In this case, our schemas could allow data redundancy, however, the overhead cost of update anomalies would be reduced. We are also working on a tool for implementing the process. This tool will support the automatic process as presented in this paper, and also a semi-automatic process which the user can choose, for example, the most convenient rule for converting a special EER fragment. In spite of that, a dynamic methodology for redesigning XML documents is being considered. This methodology could be used in applications in which the XML schemas are always evolving. Queries that are frequently performed on XML documents would be taken as a parameter for redesigning the XML schemas.

7. REFERENCES

[1] C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. The Benjamin/Cummings Publishing Company, 1992.

[2] S. Bechhofer, F. Harmelen, and J. Hendler. Owl web ontology language reference. 2002.

[3] L. Bird, A. Goodchild, and T. A. Halpin. Object role modeling and xml-schema. In *International Conference on Conceptual Modeling*, pages 661–705. Springer Heidelberg, 2000.

[4] T. Bray and J. P. et. al. Extensible markup language (xml) 1.0 w3c recommendation, 2000.

[5] M. Choi, J. Lim, and K. Joo. Developing a unified design methodology based on extended entity-relationship model for xml. In *International Conference on Computational Science*, pages 920–929. Springer Heidelberg, 2003.

[6] R. Conrad, D. Scheffner, and J. C. Freytag. Xml conceptual modeling using xml. In *International Conference on Conceptual Modeling*, pages 558–571. Springer, 2000.

[7] R. Elmasri, J. Weeldreyer, and A. R. Hevner. The category concept: An extension to the entity-relationship model. In *Data Knowledge Engineering*, number 1, pages 75–116, 1985.

[8] J. Fong and A. F. et. al. Translating relational schema with constraints into xml schema. In *International Journal of Software Engineering and Knowledge Engineering*, number 16, pages 201–244, 2006.

[9] H. Jagadish and S. A.-K. et. al. Timber: A native xml database. In *International Journal on Very Large Databases*, volume 4, pages 274–291. Springer-Verlag New York, 2002.

[10] C. Liu and J. Li. Designing quality xml schemas from e-r diagrams. In *Advances in Web-Age Information Management*, pages 508–519. Springer Heidelberg, 2006.

[11] R. S. Mello and C. A. Heuser. Binxs: A process for integration of xml schemata. In *International Conference on Advanced Information Systems Engineering*, pages 151–166. Springer Heidelberg, 2005.

[12] W. Y. Mok and D. W. Embley. Generating compact redundancy-free xml documents from conceptual-model hypergraphs. In *IEEE Transactions on Knowledge and Data Engineering*, number 18, pages 1082–1096, 2006.

[13] P. Pigozzo and E. Quintarelli. An algorithm for generating xml schemas from er schemas. In *Italian Symposium on Advanced Database Systems*, pages 192–199, 2005.

[14] N. Routledge, L. Bird, and A. Goodchild. Uml and xml schema. In *Australian Database Conference*, pages 157–166. IEEE, 2002.

[15] H. Schöning. Tamino - a dbms designed for xml-schema. In *International Conference on Data Engineering*, pages 149–154. IEEE, 2001.

[16] R. Schroeder and R. S. Mello. Conversion of generalization hierarchies and union types from extended entity-relationship model to an xml logical model. In *ACM Symposium on Applied Computing*, pages 1036–1037. ACM Press, 2008.

[17] J. M. Smith and D. C. P. Smith. Database abstractions: Aggregation and generalization. In *ACM Transactions on Database Systems*, volume 2, pages 105–133. IEEE, 1977.

[18] H. Thompson and D. B. et. al. Xml schema part 1: Structures w3c recommendation, 2004.

[19] N. Wiwatwattana and H. J. et. al. Making designer schemas with colors. In *International Conference on Data Engineering*. IEEE, 2006.

[20] Z. Xu and Z. G. et. al. Dynamic tuning of xml storage schema in vxmlr. In *International Database Engineering and Applications Symposium*, pages 76–86. IEEE, 2003.