XML Schemas: From Design to Exploitation and Back Again

Martin Nečaský and Irena Mlýnková

Charles University, Faculty of Mathematics and Physics,
Department of Software Engineering,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
{martin.necasky,irena.mlynkova}@mff.cuni.cz

Abstract. The knowledge of XML schema of XML documents is a crucial aspect to ensure that we work with correct data, as well as a key optimization approach to XML data processing. In this paper we provide an overview of research topics related to XML schemas that we have been dealing with during our work on Information Society Project (1ET100300419). We briefly describe the current state of the art and open problems we have encountered in particular areas, the results we have acquired, as well as remaining open issues and ongoing aims we have stated.

1 Introduction

Without any doubt the XML [14] is currently a de-facto standard for data representation. Its popularity is given by the fact that it is well-defined, easy-to-use and, at the same time, enough powerful. To enable users to specify own allowed structure of XML documents, so-called *XML schema*, the W3C[1] proposed two languages – DTD [14] and XML Schema [64, 11]. The former one is directly a part of XML specification and due to its simplicity it is one of the most popular formats for schema specification. The latter language was proposed later, in reaction to the lack of constructs of DTD. The key emphasis is put on simple types, object-oriented features (such as user-defined data types, inheritance, substitutability, etc.) and reusability of parts of a schema or whole schemas.

An XML schema of XML documents is currently exploited mainly for two purposes – data-exchange and optimization. In general, almost any approach that deals with XML data can benefit from the knowledge of their structure, i.e. XML schema. The only question is to what extent. But, on the other hand, statistical analyses of real-world XML data show that a significant portion of XML documents (52% [29] of randomly crawled or 7.4% [43] of semi-automatically collected) still have no schema at all. What is more, XML Schema definitions (XSDs) are used even less (only for 0.09% [29] of randomly crawled or 38% [43] of semi-automatically collected XML documents) and even if they are used, they often (in 85% of cases [9]) define so-called *local tree grammars* [45], i.e.

[1] www.w3.org

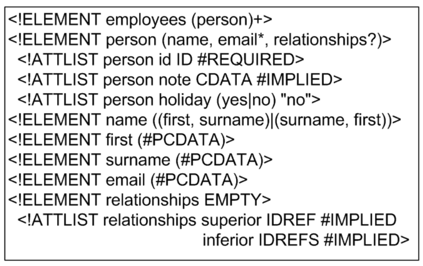
languages that can be defined using DTD as well. Consequently, it is necessary to focus on research dealing with XML schemas and to extend its popularity and exploitation.

In this paper we provide an overview of research topics that we have been dealing with during our work on Information Society Project (1ET100300419) [2]. We briefly describe the current state of the art and open problems we have encountered in particular areas, the results we have acquired, as well as remaining open issues and ongoing aims we have stated.

The paper is structured as follows: Section 2 provides a brief introduction to XML schema languages recommended by the W3C. Section 3 deals with design of XML schemas and Section 4 with a related topic of their inference. Section 5 describes selected ways of exploitation of XML schemas. In Section 6 we discuss the problems caused by their evolution. And, finally, Section 7 provides conclusions.

2 XML Schema Languages

The simplest and most popular language for description of the allowed structure of XML documents is currently the Document Type Definition (DTD) [14]. It enables one to specify allowed elements, attributes and their mutual relationships, order and number of occurrences of subelements, data types and allowed occurrences of attributes. A simple example describing a database of employees is depicted in Figure 1.



```
<!ELEMENT employees (person)+>
<!ELEMENT person (name, email*, relationships?)>
  <!ATTLIST person id ID #REQUIRED>
  <!ATTLIST person note CDATA #IMPLIED>
  <!ATTLIST person holiday (yes|no) "no">
<!ELEMENT name ((first, surname)|(surname, first))>
<!ELEMENT first (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT relationships EMPTY>
  <!ATTLIST relationships superior IDREF #IMPLIED
                          inferior IDREFS #IMPLIED>
```

Fig. 1. An example of a DTD of employees

At first glance it seems that the specification of the allowed structure is sufficient. Nevertheless, even in this simple example we can find several problems. For instance, we are not able to specify the correct structure of an e-mail address. Similarly, we cannot simply specify that a person can have four e-mail addresses at maximum. And, as we can see, the fact that the order of elements `first` and `surname` is not significant cannot be expressed simply as well. Therefore, the W3C proposed a more powerful tool – the XML Schema language [64, 11]. For example, an XSD equivalent[2] to the example of a DTD in Figure 1 is depicted in Figure 2.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="employees">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="person" minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="person">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="name"/>
    <xs:element name="email" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="relationships" minOccurs="0" maxOccurs="1"/>
   </xs:sequence>
   <xs:attribute name="id" type="xs:ID" use="required"/>
   <xs:attribute name="note" type="xs:string"/>
   <xs:attribute name="holiday" default="no">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>

 <xs:element name="name">
  <xs:complexType>
   <xs:all>
    <xs:element name="first" type="xs:string"/>
    <xs:element name="surname" type="xs:string"/>
   </xs:all>
  </xs:complexType>
 </xs:element>

 <xs:element name="relationships">
  <xs:complexType>
   <xs:attribute name="superior" type="xs:IDREF"/>
   <xs:attribute name="inferior" type="xs:IDREFS"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Fig. 2. An example of an XSD of employees

In general, the constructs of XML Schema can be divided into *basic*, *advanced* and *auxiliary*. The basic constructs involve simple data types (`simpleType`), complex data types (`complexType`), elements (`element`), attributes (`attribute`), groups of elements (`group`) and groups of attributes (`attributeGroup`). Simple data types involve both built-in data types (except for `ID`, `IDREF`, `IDREFS`), such as, e.g., `string`, `integer`, `date`, etc., as well as user-defined data types derived from existing simple types using `simpleType` construct. Complex data types enable one to specify both content models of elements and their sets of attributes.

[2] Having the same set of document instances.

The content models can involve ordered sequences (`sequence`), choices (`choice`), unordered sequences (`all`), groups of elements (`group`) or their allowable combinations. Similarly, they enable one to derive new complex types from existing ones. Elements simply join simple/complex types with respective element names and, similarly, attributes join simple types with attribute names. And, finally, groups of elements and attributes enable one to globally mark selected schema fragments and exploit them repeatedly in various parts using so-called *references*. In general, basic constructs are present in almost all XSDs.

The set of *advanced* constructs involves type substitutability and substitution groups, identity constraints (`unique`, `key`, `keyref`) as well as related simple data types (`ID`, `IDREF`, `IDREFS`) and assertions (`assert`, `report`). Type substitutability and substitution groups enable one to change data types or allowed location of elements. Identity constraints enable one to restrict allowed values of elemets/attributes to unique/key values within a specified area and to specify references to them. Similarly, assertions specify additional conditions that the values of elements/attributes need to satisfy, i.e. they can be considered as an extension of simple types.

The set of *auxiliary* constructs involves wildcards (`any`, `anyAttribute`), external schemas (`include`, `import`, `redefine`), notations (`notation`) and annotations (`annotation`).

3 Design of XML Schemas

The first problem related to XML schemas is naturally their design. Designing XML schemas has been solved intensively in recent research. However, since XML is applied at different layers of information systems (such as database, data exchange or presentation layer), it is still a hot topic. There can be various XML schemas in a particular system and an effective tool for their design would be really helpful. On the other hand, it is believed that it is easier for data designers to work firstly at a level abstracted from technical details of a particular logical data model such as XML.

During past decades, conceptual modeling has proved itself that it could provide a good level of such abstraction. Motivated by this experience, researchers have developed various conceptual models for XML data. A common idea of these approaches is that a designer first designs a conceptual diagram of an XML schema. The conceptual diagram is then translated to a representation in an XML schema language automatically. The resulting XML schema can be further edited if necessary. This provides a sufficient level of abstraction from technical details of XML schema languages.

There are several existing approaches to conceptual modeling for XML. These approaches are based on the ER model (such as [17, 28, 58, 60]) or the UML class model (such as [8, 46, 59]). They extend ER or UML with new constructs for modeling how the data is represented in XML documents utilizing special features of XML such as hierarchical and irregular structure, ordering, and mixing structured and unstructured data.

However, recent approaches have ignored the fact that XML is applied at different layers of information systems as we already mentioned. E.g., at the database layer, we need to store data in XML formats optimized for the storage. At the data exchange layer, we need to publish data in XML formats suiting the requirements of our partners. And, not finally, at the presentation layer, we need to present the data in XML formats suiting the requirements of different groups of users viewing the data from different perspectives. These typical scenarios show that there is not a single XML schema in the system, but several XML schemas describing various XML formats applied in different situations. On the other hand, these formats usually represent the same data, e.g. data on customers, products or purchases. Therefore, the same concept can be represented in various XML formats in different ways. Current methods for designing XML schemas are not sufficient in these situations because they require to model each XML format with a different conceptual diagram that is not related to conceptual diagrams modeling other XML formats. Therefore, different representations of the same concepts in various XML formats are modeled separately. This leads to repeating information in the conceptual diagrams which brings problems not only during the design of XML formats but also during their later maintenance.

In our work, we developed a conceptual model that allows to design such XML formats more easily and effectively. This is because we do not consider XML formats to be designed in separate conceptual diagrams. Instead, we consider two layers of abstraction. First, a conceptual diagram that describes the problem domain (i.e. data) independently of its representations in various XML formats is designed. A particular XML format is then designed as a separate diagram that is semi-automatically derived from the conceptual diagram. This diagram uses the components of the conceptual diagram and specifies how they are represented in the XML format. This description can then be automatically translated to a representation in an XML schema language. There can be various diagrams derived from the same conceptual diagram, each describing a separate XML format. The designed XML formats represent the same data, modeled by the conceptual diagram, but in different ways. The conceptual diagram therefore serves as an interrelation of the XML formats at the conceptual level.

Our results in this area were published in several papers and their summary forms a Ph.D. thesis [50]. In [47] we provided a complete survey of the area. In [48], we published a first version of the conceptual model. In [55], we extended the model for modeling XML keys and in [56] we provided constructs for modeling IS-A hierarchies.

3.1 Conceptual Model for XML

Our approach to conceptual modeling for XML resembles an approach to data design (or generally system design) called *Model-Driven Architecture (MDA)*. MDA considers several models that allow to describe data on different levels of abstraction. Two of the models are interesting for us: *Platform-Independent Model* (PIM) and *Platform-Specific Model* (PSM). From our point of view, the

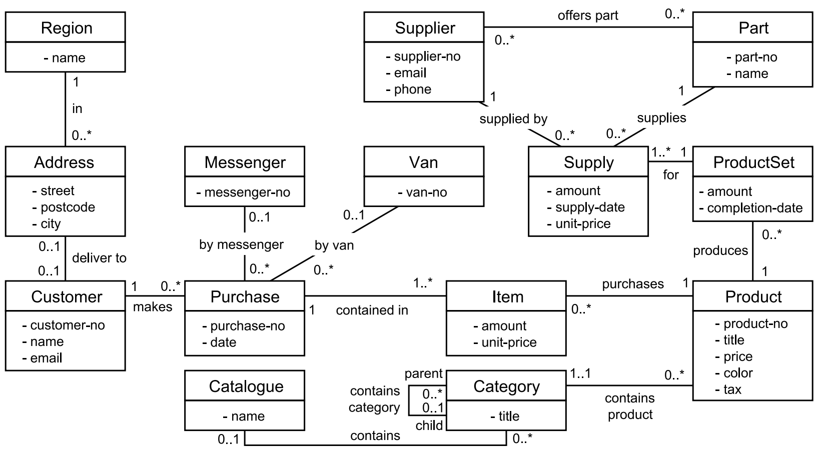


Fig. 3. Company PIM diagram

platform is XML. A PIM diagram is a conceptual diagram of the problem domain. A PSM diagram describes how a certain portion of the PIM diagram is represented in a given XML format. In the remainder of this section, we use the MDA notation for our two abstraction layers. We therefore call them *Platform-Independent Model* and *Platform-Specific Model*, respectively.

As a **Platform-Independent Model (PIM)**, we apply the well-known UML class model. We consider only its basic modeling constructs, i.e. classes for modeling concepts and binary associations for modeling relationships between the concepts. Classes can have attributes that model relevant characteristics of the concepts.

Example 1. Figure 3 shows a sample PIM diagram modeling the problem domain of a business company. For example, there is a class *Customer* modeling customers. It has attributes *customer-no*, *name* and *email* modeling relevant customer characteristics. An example association is *makes.* It connects the classes *Customer* and *Purchase* and models that customers make purchases.

The next step is to specify how the concepts and relationships modeled by the PIM diagram are represented in particular XML formats. For this, we propose a **Platform-Specific Model (PSM)** that is also based on the UML class model but adds some extensions for modeling XML specifics. Each PSM diagram models one XML format. It specifies what classes and associations from the PIM diagram are represented in the XML format and how.

Example 2. Figure 4 shows two sample PSM diagrams that model the two XML formats. The right-hand side diagram models an XML format for purchase requests while the other models an XML format for product catalogs. The PSM diagrams are derived from the PIM diagram in Figure 3. It can be easily seen that both formats represent products and related concepts but in different ways that respect the needs of respective users and situations in which the users process the data.

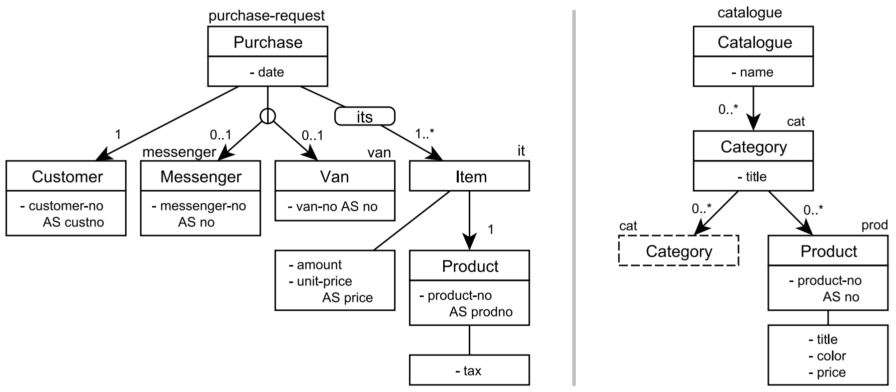


Fig. 4. Purchase and Catalogue PSM diagram

Similarly to PIM, PSM diagrams contain classes and binary associations. However, their formal semantics is different. A PSM class represents a PIM class and models XML elements that represent instances of the PIM class in the corresponding XML documents. The name of the XML elements is given by an *element label* depicted above the PSM class. The PSM class can have one or more attributes of the PIM class that are represented as XML attributes in the XML documents. Each PSM class has a content composed of the PSM associations going from the class and other components assigned to the class (i.e. attribute containers, content containers, and content choices described later). A PSM association represents a path composed of PIM associations in the PIM diagram. It specifies that an instance of its child is nested in an instance of its parent if the instances are associated by the path. Therefore, associations model the required hierarchical structure of the XML format. PSM further offers the following extending constructs:

- *Attribute container* modeling that some attributes of a PSM class are represented as XML elements not XML attributes. It is depicted using a box containing the attributes.
- *Content container* modeling an XML element enclosing a part of the XML code modeled by a PSM class. It is depicted using a box containing the name of the XML element.
- *Content choice* modeling variants in a content. It is depicted using a circle containing symbol |.
- *Structural representative* is a PSM class that inherits the attributes and content from another PSM class. It is depicted using a dashed box.

Example 3. Figure 4 shows an attribute container assigned to the PSM class *Item*. The container contains two attributes, *amount* and *unit-price*, and specifies that the attributes are represented in the XML format as XML elements not attributes. The figure also shows a content container *its* assigned to the PSM class *Purchase*. It contains a PSM association going to *Item* and models that the XML code modeled by the association is enclosed in an XML element `its`. Further, the figure shows a content choice assigned to *Purchase*. It contains two PSM associations going to *Messenger* and *Van*, respectively. It models that an XML element `purchase-request` contains an XML element `messenger` or `van` but not both. Finally, the figure shows a structural representative of a PSM class *Category*. It models that categories can contain subcategories.

Currently, we are finishing a prototype implementation of a case tool for designing XML schemas using the proposed conceptual model [3]. In our future work, we will demonstrate the power of the proposed model by modeling selected standardized XML schema languages such as ISO20020 [1]. We are also dealing with other related problems, as described in the following subsections. The results published in the Ph.D. thesis will be also published as a book [51].

3.2 Reverse-Engineering of XML Schemas

To be applicable in practice, an approach to conceptual modeling for XML must be supplemented with so-called *reverse engineering* capabilities. This is because in current information systems, several XML formats are already applied. Even though there usually exists a UML class diagram or an ER diagram that describes the data at the conceptual level, the XML formats are designed separately from this diagram. Data designers usually type them manually directly in an XML schema language. Therefore, the resulting XML schemas are not explicitly mapped to the conceptual diagram which makes the maintenance of the system harder (e.g. data evolution, change impact analysis, etc.). Suppose for example that we need to make a change in an XML schema, e.g. to add a new element. This change can cause additional changes in other XML schemas as well. Today, it is necessary to identify these additional changes manually which is time-consuming and error-prone. If we had a conceptual diagram and each XML schema was mapped to the conceptual diagram, we could propagate the change to the conceptual diagram first. And from here, we could propagate the change to other XML schemas automatically. This would greatly automatize the whole evolution process and enable more effective maintenance of all the XML schemas.

In [52], we proposed a new method for reverse engineering of XML schemas to conceptual diagrams. In particular, we showed how to map existing XML schemas to an existing conceptual diagram, i.e. a PIM diagram in the terminology adopted in this paper. The result of such a process is that the semantics of the components of the XML schemas is expressed in terms of the PIM diagram. In other words, the XML schemas are explicitly interrelated at the conceptual level. Because manual reverse engineering would be a time-consuming and error-prone activity, we proposed a semi-automatic reverse engineering method, i.e. a method that is still performed by a data designer but supported by a computer.

In the proposed solution, we assume a set of existing XML schemas and a UML class diagram modeling the same information as modeled by the XML schemas. For a given XML schema, the goal is to construct a PSM diagram that describes the same XML format as the XML schema but uses the components of the PIM diagram. We proceed in two steps. In the first step, a first approximation of the PIM diagram is mechanically derived from the XML schema. In the second step, the first approximation is refined by mapping its components to the components of the PIM diagram. While the first step is fully automatic, the second step must be assisted by a designer. Our algorithm only suggests best candidates for mapping but the final decision must be done by the designer. For making suggestions we measure the following similarities between the components of the first approximation and the components of the PIM diagram:

1. String similarity of the names of concepts and their attributes.
2. Graph structure similarity of the close neighborhood of concepts.

The result of the reverse-engineering process is a set of PSM diagrams modeling the same XML formats as modeled by the input XML schemas. The PSM diagrams organize the components of the input PIM diagram into the corresponding XML formats. Therefore, they serve as mappings between the XML schemas and the PIM diagram. A problem occurs if the PIM diagram does not cover the whole domain represented in the XML schemas. In that case the PIM diagram must be complemented by the designer with a new classes, associations or attributes otherwise it is not possible to map the XML schemas on its components. An extreme case is when a PIM diagram is missing at all.

3.3 Designing XML Databases

An existence of different XML formats representing the same data but in different ways naturally leads to a question of storing such data into a database. Storing the data directly in their XML representation would lead to a lot of redundancies in the database which is undesirable. This is because different XML formats can represent the same data repeatedly in different XML structures. We therefore need to identify these redundancies and eliminate them before the storage. Hence, designing an optimal database schema for a set of XML formats described by XML schemas is a challenging task.

In our work we tried to offer a solution to this problem. We comprehend the XML schemas as descriptions of different views on the data. Finding an optimal schema is then divided to two steps: In the first step, we identify possible sources of redundancies in the given XML schemas and normalize them to remove the redundancies from the respective XML data. The resulting XML schemas can be used as a normalized schema of an XML database. If we need to use an (object-)relational database, the second step must be performed which comprises of mapping the normalized XML schemas to an (object-)relational database schema. This second step is covered by another theme of our research described in Section 5.2. Independently of whether the second step was applied or not, we further need to reconstruct the original XML formats from the data stored in the database.

In our work, we proposed how to perform the first step at the conceptual level. In [49] we introduced an algorithm that has a PIM diagram and a set of PSM diagrams modeling the XML formats as an input and produces a set of normalized PSM diagrams as an output. From these normalized PSM diagrams, a set of normalized XML schemas is then derived automatically. In [53] we complemented this solution with a method for reconstructing the original XML formats from the normalized data. The proposed algorithm generates an XQuery expression for each non-normalized PSM diagram. The expression operates over the normalized data stored in the database.

In our future work we will concentrate on designing XML databases in conjunction with XML data evolution. This is a hot research topic today, since XML data can evolve dynamically and changes in XML data and their schemas therefore have to be somehow propagated to the underlying database. Since XML data evolution is an interesting problem on its own we dedicate it a separate Section 6.

3.4 Designing Semantic Web Services

Another area where we applied the proposed conceptual model is designing web services and describing their semantics. We published the results in [57]. Technologies of web services complemented with a layer describing semantics of web services is called *Semantic Web Services (SWS)*. The basic idea is that a syntactical description of a SWS (usually in WSDL [15]) is mapped to an ontology that provides a semantical description of the SWS. In practice there are usually various web services in the system. Each has a different syntactical description since the web services receive and send data in different XML formats. On the other hand, they share the same problem domain with the same conceptualization, i.e. ontology. Therefore, the syntactical descriptions of the web services must be mapped to the ontology. However, the mapping must be created manually. In our approach, we comprehend a PIM diagram as a simplified ontology and a set of PSM diagrams as syntactical descriptions of the data exchanged among the web services. Therefore, we are able to automatically derive the mappings of the syntactical descriptions to the ontology and save time to designers.

In our future work in this area, we will mainly concentrate on data evolution since the interfaces of web services can evolve significantly in time or even replaced at all.

4 Inference of XML Schemas

The problem of inference of XML schemas can be viewed as a subproblem of their design. In particular, we are interested in automatic inference of an XML schema from a given sample set of XML documents. Such automatically inferred schema can be then used as a candidate schema further improved by a user using an appropriate editor. From another point of view, we need to infer an XML schema whenever we are provided with a schema-less document collection, whereas its existence is crucial for further processing.

The existing solutions to the problem of automatic inference of an XML schema can be classified [32] according to several criteria. Probably the most interesting one is the type of the result (i.e. DTD or XSD) and the way it is constructed, where we can distinguish heuristic methods and methods based on inferring of a grammar.

Heuristic approaches [44, 67, 22] are based on experience with manual construction of schemas. Their result does not belong to any special class of grammars and, hence, we cannot say anything about its features. They are based on generalization of a trivial schema using a set of predefined heuristic rules, such as, e.g., "if there are more than three occurrences of an element, it is probable that it can occur arbitrary times". On the other hand, methods based on *inferring of a grammar* [4, 10] output a particular class of languages with specific characteristics. Although grammars accepting XML documents are context-free, the problem can be reduced to inferring of a set of regular expressions, each for a single element. But, since according to Gold's theorem [23] regular languages are not identifiable in the limit only from positive examples (i.e. sample XML documents which should conform to the resulting schema), the existing methods exploit restriction to an *identifiable* subclass of regular languages.

In our research we focussed on the following open issues [32]:

1. Most of the existing works infer a DTD or an XSD whose expressive power does not get beyond expressive power of DTD.
2. Only a few approaches enable one to infer integrity constraints such as keys and foreign keys.
3. Most of the existing approaches do not exploit additional available information.

In the first case we proposed two improvements: Paper [65] describes an approach that enables one to infer two purely XSD constructs – elements having the same name but different structure and unordered sequences of elements. For this purpose it utilizes several verified approaches, such as *ACO heuristics* [18], *s,k-string* [67] or *k,h-context* [4]. On the other hand, paper [37] deals with a different aspect of XML Schema – plenty of "syntactic sugar", i.e. the ability to express equivalent XSDs using distinct constructs. To determine the optimal syntax we exploit semantics of element/attribute names and statistical analysis on the input XML data. This way we are able to output schemas that carry additional information of the data, such as shared contents or more precise content models.

In the second case we proposed an approach [54] that enables one to infer identity constraints, i.e. keys and foreign keys. What is more, since the approach is based on the analysis of XML queries, it is much more efficient than approaches analyzing the input data. We propose a set of rules that enable one to determine whether a selected value certainly is or certainly is not an identity constraint. In addition, due to its nature, the approach can be used an an extension of any existing XML schema inference method.

As for the exploitation of additional input information we focussed on two aspects – exploitation of possibly existing XML queries [54] and exploitation of existing but already obsolete XML schema [36]. The former approach has already been described. In the latter case we exploit an observation that an XML schema is often [43] considered as a kind of documentation. Hence, if the data evolve, the respective schema is not adapted to the changes. In the proposed approach we exploit the information from such obsolete schema assuming that it still carries several valid information. In case there is no useful information, the inference complexity is the same as in existing works. However, in case there are usable fragments, we do not need to infer them again.

Our current and future work in this area focuses mainly on integrating of user interaction which is the key aspect in case multiple solutions are available and the searching is made only using heuristics. In fact, there seems to be no work, that would deal with this topic in detail, taking into account reasonable requirements for user's skills and amount of decisions to be made. Next, we will deal with inference of further XSD specific features, in particular integrity constraints. And, finally, since for further processing of the respective XML data also constraints that cannot be expressed in XSD may be useful, we will try to get also beyond its expressive power.

5 Exploitation of XML Schemas

Currently there are two key purposes of XML schemas. Firstly, an XML schema is considered as a set of rules the input XML documents must fulfill to be processed correctly. In other words, they ensure validity of the input XML data. Secondly, the information XML schemas bear is exploited in various approaches for optimization purposes.

In general, one of the classical optimization strategies is exploitation of similarity of data. In case of XML data we can analyze similarity of XML documents, XML schemas or between the two groups. We focussed mainly the second approach, i.e. similarity of XML schemas, which we exploit in our next research topic XML-to-relational mapping strategies. In particular, having the input data expressed in XML, a natural requirement of any application is to store and process them efficiently. One of the most reliable approaches is to map the XML data into relations, i.e. to exploit verified and robust relational database management systems (RDBMSs).

5.1 Similarity of XML Schemas

The existing methods for measuring similarity of XML schemas [40] combine various supplemental information and auxiliary similarity measures such as, e.g., predefined similarity rules, similarity of element/attribute names, equality of data types, similarity of schema instances or previous results, etc. [16, 27] But, in general, the approaches focus mostly on semantic aspects, whereas structural ones are of marginal importance. And, what is more, most of the existing works consider only DTD constructs, whereas if the XML Schema language is supported, the constructs beyond DTD expressive power are often ignored.

In our research we focussed on the following open issues:

1. Deeper involvement of structural information of the compared schema fragments.
2. Similarity of complex XML Schema constructs.
3. Reasonable tuning of weights of the similarity measure.

In the former case we proposed an approach [66] that analyzes the structure of input DTD fragments in detail. For this purpose we exploit and modify a well-known and verified strategy – tree-edit distance. However, at the same time, we enable one to involve also the semantics of element/attribute names and, hence, to provide more precise information on similarity of the input. Next, on the basis of this preliminary approach, we further proposed its extension to XSD [33, 34]. Firstly, we defined equivalence classes of XML Schema constructs in terms of both structure and semantics. Then, we modified the original approach so that it considers or omits the equivalencies depending on user requirements. And, in addition, we included also the analysis of specific intervals of occurrences and various simple data types supported by XML Schema.

Last but not least, we focussed on the problem of tuning of weights of the similarity measure. As we have mentioned, each of the XML schema similarity measures consist of several specific functions expressing similarity of particular aspect of the analyzed data. These partial results are then combined into a single result, usually using a weighted sum. The question is how to set these weights realistically. For this purpose we proposed a similarity measure [41] that exploits characteristics and results of statistical analysis of real-world XML data [43], i.e. an information based on observations of reality and not a setting based on "author's experience" commonly used in existing works.

Our future work in the area of XML similarity will focus mainly on further exploitation of similarity of XML schemas in various areas, as well as optimization strategies of similarity evaluation. In the former case we intent to deal with the problem of data integration and related issue of similarity of XML documents and XML schemas. In the latter case we are dealing with the problem of setting the weights of similarity measure as well as similarity threshold using reasonable user interaction. In general, most of the exiting works consider the ability to set these parameters according to requirements of a specific application as an advantage. However, it is not an easy task to find the reasonable setting.

5.2 XML-to-Relational Mapping Strategies

At present, there exists a plenty of works concerning database-based[3] XML data management [38]. All major database vendors support XML and even the SQL standard was extended with a new part (SQL/XML) which introduces operations on XML data. The main concern of the techniques is the choice of the way XML data are stored into relations – so-called *XML-to-relational mapping*. On the basis of exploitation of information from XML schema we distinguish *schema-oblivious* (e.g. [21]) and *schema-driven* (e.g. [61]) methods. From the point of view of the input data we distinguish *fixed* methods (e.g. [21, 61]) which store the data on the basis of their model and *adaptive* methods (e.g. [26, 12, 69, 68]), where also additional information on the future application are taken into account. And there are also techniques based on user involvement which can be divided to *user-defined* (see [6]) and *user-driven* (e.g. [7, 19, 31]), where in the former case a user defines both the relational schema and the required mapping, whereas in the latter case a user specifies just local mapping changes of a default storage strategy.

Currently the most efficient XML-to-relational storage strategies are the adaptive methods [39]. The reason is that they evaluate several fixed mapping strategies for particular XML schema fragments and choose the combination which suits the target application (specified using a sample set of XML queries and XML documents) the most. The user-driven methods can be viewed as a special type of adaptive methods too. The difference is that the mapping strategies are for selected schema fragments specified by a user.

In our recent research we proposed several improvements of adaptive methods. In papers [42, 30] we propose an approach which extends user-driven strategies with exploitation of similarity of XML schema fragments (as described before). It is based on a simple observation that it is highly probable that structurally and/or semantically similar schema fragments should be stored in a similar way. The approach focuses on the way of finding such similar fragments and their combination to the resulting mapping.

On the other hand, in paper [35] we further improve the classical adaptive strategies, in particular the process of searching the optimal strategy using the ACO heuristics [18]. It enables one to find the optimum more efficiently than simple heuristics used in the existing papers and without getting stuck in local optimums. In addition, we describe the way classical adaptive strategies can be combined with user-driven methods. We result from the idea that for selected schema fragments a user may specify directly the mapping strategy, while for others it is much convenient to specify the set of expected queries. We propose an approach that enables to exploit both these information to find the optimal mapping strategy.

[3] In the rest of the paper the term "database" represents a RDBMS.

6 XML Data Evolution

Another problem related to XML is XML data evolution. Since most of the applications using XML are usually dynamic, sooner or later the structure of the XML data needs to be changed. At the same time, we still need to be able to work with the old as well as new data without any loss. In relation to this topic, we usually speak about so-called *schema evolution*, i.e. a situation that a schema of the data is updated and we need to apply these updates on the respective data to revalidate them against the evolving schema.

Currently there exist several works dealing with this topic, however most of them focus on separate aspects such as evolution of XML schemas [5, 24, 62, 63], XML data [13], or conceptual diagrams [20, 25]. In our recent work, we concentrate on this problem from a more general perspective. An important point is that it is necessary to consider various XML formats in a system that represent the same data in different XML structures. Therefore, a change in one XML format (i.e. its XML schema) can cause additional changes in the others. With this on mind, we show that schema evolution has several different levels that are highly related and influence each other. In particular we deal with five levels – extensional, operational, logical, platform-specific and platform-independent. The overall picture of the proposed XML data evolution architecture is depicted in Figure 5.

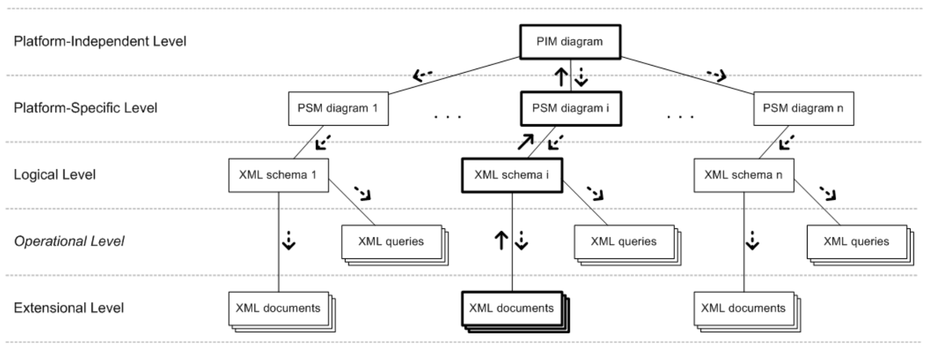


Fig. 5. 5-level XML Evolution Approach

The topmost level contains a PIM diagram of the problem domain and it is therefore called *platform-independent level*. It describes the domain independently of various representations of the domain in XML formats. The next level is called *platform-specific level* and contains a PSM diagram for each considered XML format. The level beneath the platform-specific level is called *logical level* and contains for each considered type of XML documents an XML schema that specifies the logical structure of the XML documents. Its components are mapped to the PIM diagram via the corresponding PSM diagram from the upper level. The lowest level is called *extensional level* and contains XML documents. It contains a set of XML documents for each considered type. The level between the extensional and logical level is called *operational level* and contains XML queries over the XML documents. For each type of XML documents we consider several queries.

The architecture allows a designer to make a change at any level (except the query level). The locations of possible changes are depicted in a bold line. It further ensures a propagation of the change to the other levels as depicted by the arrows. In particular, if a change in a given XML format is made, the architecture ensures an automatic propagation of the change to the other XML schemas. For this automatic propagation, we define a set of atomic edit operations for each evolution level as well as mappings between the operations at each two subsidiary levels.

A natural question is how we can build the proposed architecture. In the worst case, only raw XML data and queries over the XML data are present. Therefore, there are neither XML schemas, nor PSM and/or PIM diagrams. In that case we can utilize our results presented in the previous sections. In particular, we are able to infer XML schemas of the XML documents using the methods presented in Section 4 and reverse engineer these XML schemas to a PIM diagram as we showed in Section 5.2.

This research topic is mainly a matter of our future work even thought some results are ready to be published. The most important problem to solve is to define a consistent set of edit operations and their mappings between the levels. An important issue is also extending our prototype case tool [3] with the proposed evolution approach. This would allow to evaluate our evolution approach experimentally.

7 Conclusion

The knowledge of the structure of XML data, i.e. their XML schema is a crucial aspect not only to ensure that we work with correct data, but, in particular, as a key optimization approach to XML data processing. In this paper we have described the topics related to XML schema we have been dealing with during our recent research, as well as the respective results and achievements. In particular, this includes designing and inferring XML schemas, measuring similarity of XML schemas, storing XML data in relational databases and XML data evolution. We have also described the remaining open issues or challenges we are currently dealing with to optimize the process of creating XML schemas, as well as the ways they can be exploited.

Acknowledgement

This work was supported in part by the National Programme of Research (Information Society Project 1ET100300419).

References

1. *Universal Financial Industry Message Scheme (ISO 20022).* http://www.iso20022.org/.
2. *SemWeb – Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation.* 2004–2008. http://www.cs.cas.cz/semweb/index.php?content=homepage&lang=en.
3. *XCase – A Tool for XML Data Modeling.* 2008. http://kocour.ms.mff.cuni.cz/~necasky/xcase/.
4. H. Ahonen. *Generating Grammars for Structured Documents Using Grammatical Inference Methods.* Report A-1996-4, Dept. of Computer Science, University of Helsinki, 1996.
5. L. Al-Jadir and F. El-Moukaddem. Once Upon a Time a DTD Evolved into Another DTD... In *OOIS'03*, pages 3–17, Berlin, Heidelberg, 2003. Springer-Verlag.
6. S. Amer-Yahia. *Storage Techniques and Mapping Schemas for XML.* Report TD-5P4L7B, AT&T Labs-Research, 2003.
7. A. Balmin and Y. Papakonstantinou. Storing and Querying XML Data Using Denormalized Relational Databases. *The VLDB Journal*, 14(1):30–49, 2005.
8. M. Bernauer, G. Kappel, and G. Kramler. Representing XML Schema in UML - An UML Profile for XML Schema. Technical report, Department of Computer Science, National University of Singapore, 2003.
9. G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: a Practical Study. In *WebDB'04*, pages 79–84, New York, NY, USA, 2004. ACM Press.
10. G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML Schema Definitions from XML Data. In *VLDB'07*, pages 998–1009, Vienna, Austria, 2007. ACM Press.
11. P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition).* W3C, 2004. http://www.w3.org/TR/xmlschema-2/.
12. P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML Schema to Relations: A Cost-based Approach to XML Storage. In *ICDE'02*, pages 64–75, Washington, DC, USA, 2002. IEEE Computer Society.
13. B. Bouchou, D. Duarte, M. Halfeld Ferrari Alves, D. Laurent, and M. A. Musicante. Schema Evolution for XML: A Consistency-Preserving Approach. In *MFCS'04*, pages 876–888, Prague, Czech Republic, 2004. Springer-Verlag.
14. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* W3C, 2006. http://www.w3.org/TR/REC-xml/.
15. C. K. Liu D. Booth. *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer.* W3C, June 2007. http://www.w3.org/TR/wsdl20-primer/.
16. H. H. Do and E. Rahm. COMA – A System for Flexible Combination of Schema Matching Approaches. In *VLDB'02*, pages 610–621, Hong Kong, China, 2002. Morgan Kaufmann.
17. G. Dobbie, W. Xiaoying, T.W. Ling, and M.L. Lee. ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. Technical report, Department of Computer Science, National University of Singapore, Singapore, 2000.
18. M. Dorigo, M. Birattari, and T. Stutzle. *Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique.* Report TR/IRIDIA/2006-023, IRIDIA, Bruxelles, Belgium, 2006.
19. F. Du, S. Amer-Yahia, and J. Freire. ShreX: Managing XML Documents in Relational Databases. In *VLDB'04*, pages 1297–1300, Toronto, ON, Canada, 2004. Morgan Kaufmann.

20. G. Fiedler and B. Thalheim. An Approach to Conceptual Schema Evolution. Technical Report 0701, Institut fur Informatik der Christian-Albrechts-Universitat, Kiel, 2007.
21. D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.
22. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: a System for Extracting Document Type Descriptors from XML Documents. In *SIGMOD'00*, pages 165–176, New York, NY, USA, 2000. ACM Press.
23. E. M. Gold. Language Identification in the Limit. *Information and Control*, 10(5):447–474, 1967.
24. G. Guerrini, M. Mesiti, and D. Rossi. Impact of XML Schema Evolution on Valid Documents. In *WIDM'05*, pages 39–44, New York, NY, USA, 2005. ACM Press.
25. M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63. Verlagshaus Mainz, Aachen, 2007.
26. M. Klettke and H. Meyer. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. In *WebDB'00 Workshop*, pages 151–170, London, UK, 2001. Springer-Verlag.
27. M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. XClust: Clustering XML Schemas for Effective Integration. In *CIKM'02*, pages 292–299, New York, NY, USA, 2002. ACM Press.
28. M. Mani. Semantic Data Modeling Using XML Schemas. In *ER'01*, pages 149–163, Yokohama, Japan, 2001. Springer-Verlag.
29. L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *WWW'03*, pages 500–510, New York, NY, USA, 2003. ACM Press.
30. I. Mlynkova. A Journey towards More Efficient Processing of XML Data in (O)RDBMS. In *CIT'07*, pages 23–28, Aizu-Wakamatsu City, Fukushima, Japan, 2007. IEEE Computer Society.
31. I. Mlynkova. *UserMap – an Enhancing of User-Driven XML-to-Relational Mapping Strategies.* Report 2007/3. Charles University, Prague, Czech Republic, 2007.
32. I. Mlynkova. An Analysis of Approaches to XML Schema Inference. In *SITIS'08 (to appear)*, Bali, Indonesia, 2008. IEEE Computer Society.
33. I. Mlynkova. Equivalence of XSD Constructs and its Exploitation in Similarity Evaluation. In *ODBASE'08*, volume 5332 of *LNCS*, pages 1253–1270, Monterrey, Mexico, 2008. Springer-Verlag.
34. I. Mlynkova. Similarity of XML Schema Definitions. In *DocEng'08*, pages 187–190, Sao Paulo, Brazil, 2008. ACM Press.
35. I. Mlynkova. Standing on the Shoulders of Ants: Towards More Efficient XML-to-Relational Mapping Strategies. In *XANTEC workshop of DEXA'08*, pages 279–283, Turin, Italy, 2008. IEEE Computer Society.
36. I. Mlynkova. On Inference of XML Schema with the Knowledge of an Obsolete One. In *ADC'09 (to appear)*, volume 92 of *CRPIT*, Wellington, New Zealand, 2009. Australian Computer Society.
37. I. Mlynkova and M. Necasky. Towards Inference of More Realistic XSDs. In *SAC'09 (to appear)*, Honolulu, Hawaii, USA, 2009. ACM Press.
38. I. Mlynkova and J. Pokorny. XML in the World of (Object-)Relational Database Systems. In *ISD'04*, pages 63–76, Vilnius, Lithuania, 2004. Springer-Verlag.
39. I. Mlynkova and J. Pokorny. Adaptability of Methods for Processing XML Data using Relational Databases – the State of the Art and Open Problems. In *RCIS'07*, pages 183–194, Ouarzazate, Morocco, 2007. Ecole Marocaine des Sciences de l'Ingnieur.

40. I. Mlynkova and J. Pokorny. Similarity and XML Technologies. In *ICWI'07*, pages 277–287, Vila Real, Portugal, 2007. IADIS.
41. I. Mlynkova and J. Pokorny. Similarity of XML Schema Fragments Based on XML Data Statistics. In *Innovations'07*, pages 243–247, Al Ain, UAE, 2007. IEEE Computer Society.
42. I. Mlynkova and J. Pokorny. UserMap – an Adaptive Enhancing of User-Driven XML-to-Relational Mapping Strategies. In *ADC'08*, volume 75 of *CRPIT*, pages 165–174, Wollongong, New South Wales, Australia, 2008. Australian Computer Society.
43. I. Mlynkova, K. Toman, and J. Pokorny. Statistical Analysis of Real XML Data Collections. In *COMAD'06*, pages 20–31, New Delhi, India, 2006. Tata McGraw-Hill.
44. C.-H. Moh, E.-P. Lim, and W.-K. Ng. Re-engineering Structures from Web Documents. In *DL'00*, pages 67–76, New York, NY, USA, 2000. ACM Press.
45. M. Murata, D. Lee, and M. Mani. Taxonomy of XML Schema Languages Using Formal Language Theory. *ACM Trans. Inter. Tech.*, 5(4):660–704, 2005.
46. K. Narayanan and S. Ramaswamy. Specifications for Mapping UML Models to XML. In *WiSME'05*, page 10, Montego Bay, Jamaica, 2005.
47. M. Necasky. Conceptual Modeling for XML: A Survey. Technical Report TR 2006-3, Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, 2006. `http://www.necasky.net/papers/tr2006.pdf`.
48. M. Necasky. XSEM – A Conceptual Model for XML. In *APCCM'07*, volume 67 of *CRPIT*, pages 37–48. Australian Computer Society, 2007.
49. M. Necasky. Conceptual Model Based Normalization of XML Views. In *Dateso'08*, pages 13–24. CEUR-WS, Vol. 330, 2008.
50. M. Necasky. *Conceptual Modeling for XML*. PhD thesis, Charles University, 2008. `http://kocour.ms.mff.cuni.cz/~necasky/dw/thesis.pdf`.
51. M. Necasky. *Conceptual Modeling for XML (In press)*, volume 99 of *Dissertations in Database and Information Systems*. IOS Press, Amsterdam, Netherlands, 2009.
52. M. Necasky. Reverse Engineering of XML Schemas to Conceptual Diagrams. In *APCCM'09 (to appear)*, volume 96 of *CRPIT*. Australian Computer Society, 2009.
53. M. Necasky and T. Knap. Reconstruction of Normalized XML Documents. In *Innovations'08 (to appear)*, Al Ain, UAE, 2008. IEEE Computer Society.
54. M. Necasky and I. Mlynkova. Enhancing XML Schema Inference with Keys and Foreign Keys. In *SAC'09 (to appear)*, Honolulu, Hawaii, USA, 2009. ACM Press.
55. M. Necasky and J. Pokorny. Extending ER for Modeling XML Keys. In *ICDIM'07*, pages 236–241, Lyon, France, 2007. IEEE Computer Society.
56. M. Necasky and J. Pokorny. Conceptual Modeling of IS-A Hierarchies for XML. In *EJC'08*, Tsukuba, Japan, 2008.
57. M. Necasky and J. Pokorny. Design and Management of Semantic Web Services using Conceptual Model. In *SAC'08*, pages 2243–2247, Fortaleza, Caerá, Brazil, 2008. ACM Press.
58. G. Psaila. ERX: A Conceptual Model for XML Documents. In *SAC'00*, pages 898–903, Como, Italy, 2000. ACM Press.
59. N. Routledge, L. Bird, and A. Goodchild. UML and XML Schema. In *ADC'02*, volume 5 of *CRPIT*, Melbourne, Australia, 2002. Australian Computer Society.
60. A. Sengupta, S. Mohan, and R. Doshi. XER - Extensible Entity Relationship Modeling. In *XML'03*, pages 140–154, Philadelphia, USA, 2003.
61. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and

Opportunities. In *VLDB'99*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann.

62. H. Su, D. K. Kramer, and E. A. Rundensteiner. XEM: XML Evolution Management. Technical Report WPI-CS-TR-02-09, Computer Science Department, Worcester Polytechnnic Institute, Worcester, Massachusetts, 2002.
63. M. Tan and A. Goh. Keeping Pace with Evolving XML-Based Specifications. In *EDBT '04 Workshops*, pages 280–288, Berlin, Heidelberg, 2005. Springer-Verlag.
64. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition)*. W3C, 2004. `http://www.w3.org/TR/xmlschema-1/`.
65. O. Vosta, I. Mlynkova, and J. Pokorny. Even an Ant Can Create an XSD. In *DASFAA'08*, LNCS, pages 35–50. Springer-Verlag, 2008.
66. A. Wojnar, I. Mlynkova, and J. Dokulil. Similarity of DTDs Based on Edit Distance and Semantics. In *IDC'08*, volume 162 of *Studies in Computational Intelligence*, pages 207–216, Catania, Italy, 2008. Springer-Verlag.
67. R. K. Wong and J. Sankey. *On Structural Inference for XML Data*. Report UNSW-CSE-TR-0313, School of Computer Science, The University of New South Wales, 2003.
68. W. Xiao-ling, L. Jin-feng, and D. Yi-sheng. An Adaptable and Adjustable Mapping from XML Data to Tables in RDB. In *VLDB'02 Workshop EEXTT and CAiSE'02 Workshop DTWeb*, pages 117–130, London, UK, 2003. Springer-Verlag.
69. S. Zheng, J. Wen, and H. Lu. Cost-Driven Storage Schema Selection for XML. In *DASFAA'03*, pages 337–344, Kyoto, Japan, 2003. IEEE Computer Society.