A Methodology for Integrating XML Data into Data Warehouses

Boris Vrdoljak, Marko Banek, Zoran Skočir

University of Zagreb
Faculty of Electrical Engineering and Computing
Address: Unska 3, HR-10000 Zagreb, Croatia
E-mail: boris.vrdoljak@fer.hr, marko.banek@fer.hr, zoran.skocir@fer.hr

Abstract -- ***Data warehousing systems enable managers and analysts to acquire, integrate and flexibly analyze information from different sources. Since XML has become a standard for data exchange over the Internet, especially in B2B and B2C communication, there is a need of integrating XML data into warehousing systems. In this paper we describe our methodology for data warehouse design, when sources of data are XML Schemas and conforming XML documents. Special attention is given to conceptual multidimensional design. The paper also presents the main features of the prototype tool we have implemented to support our methodology.***

I. INTRODUCTION

Data warehousing system is a set of technologies and tools that enable managers and analysts to acquire, integrate and flexibly analyze information coming from different sources. The central part of the system is a database specialized for complex analysis of historical data, called a data warehouse. The process of building a data warehouse system includes analysis of different data sources, design of data warehouse model, definition of transformation and integration processes, construction of data warehouse and implementation of tools that users employ to get the wanted data from the warehouse.

The increasing use of XML in business-to-business (B2B) applications and e-Commerce Web sites, suggests that a lot of valuable external data sources will be available in XML format on the Internet. Large volumes of XML data already exist in information systems of various companies and organizations. The possibility of integrating available XML data into data warehouses will play an important role in providing enterprise managers with up-to-date and comprehensive information about their business domain.

Recent research on data warehouse systems has yielded solutions for the warehouse design from relational sources. As a lot of data is becoming available in XML format, and much of database research focus is shifting from the traditional relational model to semi-structured data and XML, data warehouse design from XML sources and integration of XML data into warehousing systems becomes a hot topic.

In this paper, a methodology for semi-automated design of data warehouses from XML Schemas [11] and conforming XML documents is proposed. When designing data warehouse from XML sources, two main issues arise: first, since XML models semi-structured data, not all the information needed can be safely derived; second, different ways of representing relationships in XML Schemas are possible, each achieving different expressive power. Relationships can be specified either by sub-elements with different cardinalities or by a mechanism that is similar to the concept concerning keys and foreign keys in relational databases. To support the methodology for data warehouse design from XML sources, a Java-based prototype tool has been developed.

Some approaches concerning related issues have been proposed in the literature. In [8] DTDs are used as a source for designing multidimensional schemas (modeled in UML). Though that approach bears some resemblance to ours, XML and relational data are not physically stored into a data warehouse, but fetched on-demand from respective sources. Furthermore, the unknown cardinalities of relationships are not verified against actual XML data, but they are always assumed to be to-one. The approach described in [9] is focused on populating multidimensional cubes by collecting XML data, but assumes that the multidimensional schema is known in advance (i.e., that conceptual design has been already carried out). In [10], the author shows how to use XML to directly model multidimensional data, without addressing the problem of deriving the multidimensional schema.

In [3], a technique for conceptual design starting from DTDs is outlined. That approach is now partially outdated due to the increasing popularity of XML Schema. In [4], conceptual design from XML Schema, including some complex modeling situations, has been presented. However, in [3] and [4] a complete methodology that includes conceptual, logical and physical design has not been explained, and the functional architecture of the system has not been presented.

The paper is structured as follows. After explaining multidimensional modeling in Section II, in Section III we show how relationships are modeled in XML Schemas. In Section IV we propose our methodology and functional architecture for data warehouse design from XML sources. Section V shortly describes the prototype tool we have developed to support the methodology. Finally, in Section VI the conclusions are drawn.

II. MULTIDIMENSIONAL MODELING

Multidimensional data model [5] is used in data warehouses in order to make the data accessible to OLAP and reporting tools and enable efficient analysis of a large amount of data.

In this paper the Dimensional Fact Model [2] is used as a conceptual multidimensional model, in which a data warehouse is represented by means of a set of *fact schemes*. A fact scheme is structured as a rooted graph whose root is a fact. The components of fact schemes are facts, measures, dimensions and hierarchies. A *fact* is a focus of interest for the decision-making process. It typically corresponds to events occurring dynamically in the enterprise world (such as sales or orders, for example). *Measures* are continuously valued (typically numerical) attributes that describe the fact. *Dimensions* are discrete attributes which determine the minimum granularity adopted to represent facts. *Hierarchies* are made up of discrete dimension attributes linked by -to-one relationship, and determine how facts may be aggregated. In other words, each hierarchy includes a set of attributes linked by functional dependences; for instance, *city* functionally determines *country*.

The fact scheme, as a conceptual scheme, can be implemented either in a relational database or in a proprietary structure called multidimensional database. End users of OLAP tools should never be concerned about the storage of data, and should be able to treat the resulting database as a conceptually coherent multidimensional structure. When implementing the fact scheme in a relational database, the *star schema* is typically used. It is composed of one table with a multi-part key, called the *fact table*, and a set of tables with a single-part key, called *dimensional tables*. Figure 1 shows the star schema for the sales example. Every element of the multi-part key in the fact table is a foreign key to a single dimension table. The dimensions in the sales example are product, customer and time. In the case of implementing the fact scheme in a multidimensional database, data is stored in an array structure similar to the programming language array.

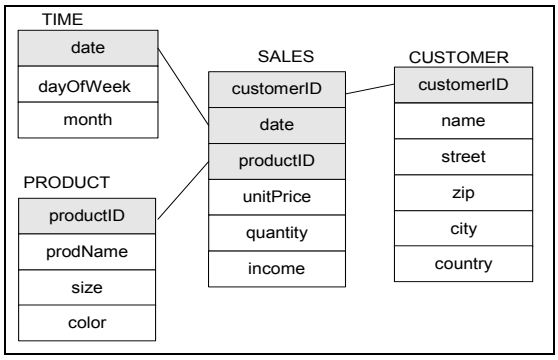

Figure 1. Star schema

III. EXPRESSING RELATIONSHIPS IN XML SCHEMA

In this paper we focus on using XML Schema and XML data as a source for designing data warehouses. To be able to navigate the functional dependencies (i.e. to-one relationships) and derive a correct multidimensional representation of the XML data, different ways of expressing relationships in XML Schema should firstly be examined.

The structure of XML data can be visualized by using a *schema graph* derived from the Schema describing the data, similarly as it has been proposed in [7] for DTDs. The schema graph for the XML Schema describing the sales of different products to different customers is shown in Figure 2. In addition to the schema graph vertices that correspond to elements and attributes in the XML Schema, the operators inherited from the DTD element type declarations are also used because of their simplicity. They determine whether the sub-element or attribute may appear one or more ("+"), zero or more ("*"), or zero or one times ("?"). The default cardinality is exactly one and in that case no operator is shown. Attributes and sub-elements are not distinguished in the graph.

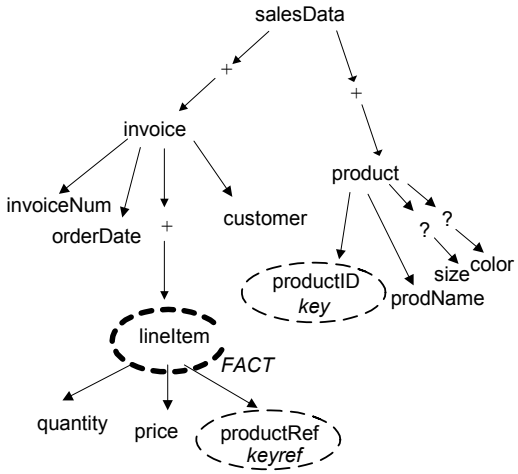

Figure 2. Schema graph

Since our design methodology is primarily based on detecting many-to-one relationships, in the following we will focus on the way those relationships can be expressed. There are two different ways of specifying relationships in XML Schemas.

- First, relationships can be specified by sub-elements with different cardinalities. However, given an XML Schema, we can express only the cardinality of the relationship from an element to its sub-elements and attributes. The cardinality in the opposite direction cannot be discovered by exploring the Schema; only by exploring the data that conforms to the Schema or by having some knowledge about the domain described, it can be concluded about the cardinality in the direction from a child element to its parent.
- Second, the *key* and *keyref* elements can be used for defining keys and their references. The *key* element indicates that every attribute or element value must be unique within a certain scope and not null. If the key is an element, it should be of a simple type. By using *keyref* elements, keys can be referenced. Not just attribute values, but also element content and their combinations can be declared to be keys, provided that the order and type of those elements

and attributes is the same in both the *key* and *keyref* definitions. In contrast to *id/idref* mechanism in DTDs, *key* and *keyref* elements are specified to hold within the scope of particular elements. In the schema graph presented in Figure 2, the detailed data about products is stored in a separate sub-graph. However, the connections between portions of data in the graph can be made using the key/keyref mechanism.

IV. METHODOLOGY FOR DATA WAREHOUSE DESIGN FROM XML SOURCES

A design methodology is an essential requirement to ensure the success of complex data warehousing project. Basic phases in data warehouse design, when data sources are either E/R diagrams or relational logical schemes, have been presented in [6]. However, there are still many challenges for scientific community in the field of data warehouse design when data sources are in XML format, because of semi-structured nature of XML data.

In the following, a methodology for multidimensional design starting from XML Schema is proposed. The methodology from [1] and [6] has been adapted in order to address various issues emerging from the semi-structured nature of XML data

A. Functional architecture

The functional architecture for data warehouse design from XML sources is presented in Figure 3. The main functions of the functional architecture, i.e. the basic phases of the methodology, are explained in the following.

B. Preliminary work

B.1. Analysis

Before starting the data warehouse design, available XML sources are analyzed, and user requirements are specified. Designer of the data warehouse analyzes XML Schema and XML documents conforming to the XML Schema, as well as available documentation, in order to determine data semantics, data quality, the number of available XML documents, etc. Requirement specification involves designer and final users. User requirements are collected and filtered. Requirement specification determines the choice of *facts* and *preliminary workload*. As already explained, each fact is a focus of interest for the decision-making process and it becomes the root of the fact scheme in the Dimensional Fact Model. *Preliminary workload* is a set of most frequent/interesting queries on fact schemes, i.e. a set of queries final users will most likely use when querying the data warehouse.

B.2. Storing XML

After XML Schemas and XML documents have been extracted from the Internet, they should be stored locally, in the way that allows validation and querying of XML document. A storage that keeps the XML documents as a whole, and enables validating them against their XML Schema is needed. The best solutions are native XML databases or XML-enabled relational databases, which use Large Object (LOB) data types and other features for storing and retrieving XML. It is also possible to use file systems, but in that case the validation should be provided manually, and the support database systems usually provide would not be available.

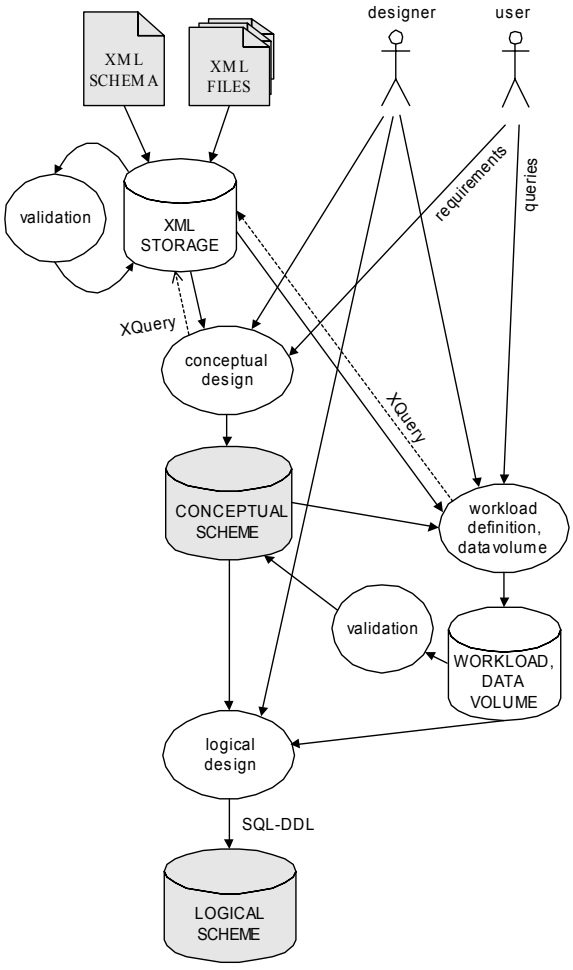

Figure 3 - Functional architecture for DW design from XML sources

C. Design

Figure 3 shows the main steps of data warehouse design starting from XML sources: conceptual design, workload definition, and logical design. To complete the design process, ETL (Extraction, Transformation and Loading) design and physical design should also be included. All the design phases are explained in the following.

C.1. Conceptual design

While conceptual design from E/R diagrams or relational logical schemes has already been explored [1], conceptual design from XML sources brings a large number of specific challenges to be solved. The challenges emerge from the semi-structured nature of XML data. As the hierarchies included in the multidimensional schema represent many-to-one relationships, the main problem when building a conceptual multidimensional model is to identify those relationships. It has already been shown that two different ways of specifying relationships in XML Schemas exist; relationships can be specified either by sub-elements with different cardinalities or by defining keys and their references.

We propose a methodology for conceptual multidimensional design starting from XML sources that consists of the following steps:

1. Preprocessing the XML Schema.
2. Creating a schema graph.
3. Choosing facts.
4. For each fact:

   4.1 Building a dependency graph from the schema graph.

   4.2 Rearranging the dependency graph.

   4.3 Defining dimensions and measures.

   4.4 Creating the fact scheme.

After a schema graph that represents the structure of the simplified XML Schema has been automatically created (steps 1 and 2), and the designer has chosen a fact as a focus of interest for analysis (step 3), a dependency graph is built in a semi-automated way (step 4). The dependency graph is an intermediate structure used to provide a multidimensional representation of the data describing the fact. In particular, it is a directed rooted graph initialized with the fact vertex. The vertices of the dependency graph are a subset of the element and attribute vertices of the schema graph, and its arcs represent associations between vertices. The dependency graph is enlarged by recursively navigating the functional dependencies between the vertices of the schema graph. The navigation goes in three directions:

- "direction down" (towards the descendants of the fact).
- direction up" (towards the ascendants of the fact),
- following the key/keyref mechanism.

Relationships in the direction from the fact to its descendants ("direction down") are expressed by arcs of the schema graph, and the cardinality information is expressed either explicitly by "?", "*" and "+" vertices, or implicitly by their absence. The dependency graph is enlarged by recursively navigating parent-child relationships in the schema graph. After a vertex *v* of the schema graph is inserted in the dependency graph, it should be decided which of its children will be included in the dependency graph. The algorithm for the "direction down" is presented as follows.

For each vertex w that is a child of v in the schema graph:

- *If w corresponds to an element or attribute in the schema, it is added to the dependency graph as a child of v.*
- *If w is a "?" operator, its child is added to the dependency graph as a child of v.*
- *If w is a "*" or "+" operator, the cardinality of the relationship from u, child of w, to v is checked by querying the XML documents using the XQuery language [12]. Designer chooses the identifiers for v and u. If the relationship is to-many, the designer decides whether the many-to-many relationship between v and u is interesting enough to be inserted into the dependency graph or not.*

In the direction from the fact to its ascendants ("direction up") the schema yields no information about the relationship cardinality. The dependency graph is enlarged in this direction by recursively navigating child-parent relationships in the schema graph. After a vertex *v* of the schema graph is inserted in the dependency graph, the algorithm for examining whether a parent vertex z will be added to the dependency graph is described in the following.

For each vertex z that is a parent of v in the Schema graph:

When examining relationships in this direction, vertices corresponding to "?", "" and "+" operators are skipped as they only express the cardinality in the opposite direction. Since XML Schema offers no possibility to define the occurrences in this direction, it is necessary to examine the actual data by querying the XML documents conforming to the schema. Before the relationship in the direction from the v to z is examined, the designer chooses the identifiers of both vertices. If a -to-many relationship is detected, z is not included in the dependency graph. Otherwise, we still cannot be sure that the cardinality of the relationship from v to z is -to-one. In this case, only the designer can tell, leaning on her knowledge of the business domain, whether the actual cardinality is -to-one or -to-many. Only in the first case, z is added to the dependency graph.*

The third part of the algorithm is concerning the case that a vertex referencing a key vertex is reached in the schema graph. It is specified as follows.

Let k and r be vertices of the schema graph SG, each of them corresponding to an attribute or a simple type element in the schema S, where k is specified as a key, and r as a keyref referencing k. Let z be a vertex of the dependency graph DG that corresponds to the vertex r in

the SG. When building the dependency graph DG, whenever the vertex r is reached in the SG:

- *the navigation algorithm "jumps" to the vertex k,*
- *the vertex k is swapped with its parent vertex p,*
- *the descendants of k in SG become the descendants of the z in DG.*

The usage of the key/keyref mechanism will be shown on the sales example. The schema graph for sales has been presented in Figure 2. The *lineItem* vertex has been chosen as a fact. Following functional dependences represented by –to-one relationships, vertices *quantity*, *price* and *productRef* are added to the dependency graph, as shown in Figure 4.

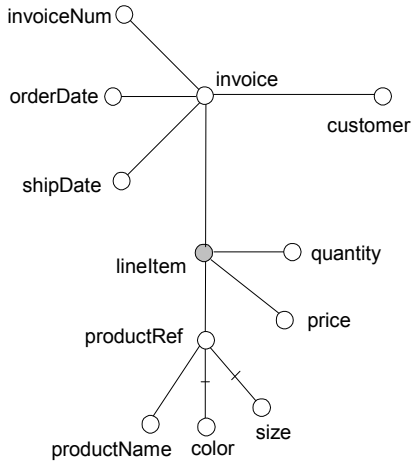

Figure 4. Dependency graph for sales

Since *productRef* is referencing *productID*, the algorithm "jumps" to *productID*, which is swapped with *product*, and *product* is then eliminated since it carries no value. *prodName*, *size*, and *color* become children of *productID* in the schema graph. Those vertices are then added to the dependency graph. The operation of replacing the foreign key vertex with the primary key vertex and its sub-graph is similar to the natural join in the relational model, and it prevents from losing the schema graph vertices that can be interesting for a more precise and detailed description of the chosen fact, and therefore for making useful aggregations of data.

When navigating in the direction from the *lineItem* fact vertex to its parent vertex *invoice*, the relationship between those vertices should be examined. After the examination, the *invoice* vertex and its children (*invoiceNum*, *orderDate*, *shipDate* and *customer*) are included in the dependency graph.

After deriving the dependency graph from the schema graph, it may be rearranged (step 4.2); typically, some uninteresting attributes are dropped. This phase of design necessarily depends on the user requirements and cannot be carried out automatically. After the designer has selected dimensions and measures among the vertices of the dependency graph (step 4.3), the dependency graph can easily be translated into a fact scheme as a conceptual multidimensional scheme (step 4.4).

C.2 .Workload definition and data volume acquisition

This phase can be divided in two parts. First, the preliminary workload (i.e. set of most frequent/interesting queries) is refined, by reformulating it in deeper detail. The workload is checked against the conceptual scheme, and this way the conceptual scheme is validated. Second, XQuery is used to query XML data sources in order to determine the current data volume.

Both the query workload and the data volumes will have an important role in logical and physical design; they represent key considerations in tuning a data warehouse.

C.3. Logical design

Logical design includes a set of steps that lead to the definition of a logical scheme starting from the previously defined conceptual scheme. We use relational logical model, using star schemas and their derivations. The reason for this choice is that relational databases are scalable, standardized, widely known, and flexible for advanced design problems. For each fact scheme defined, and taking both the workload and the data volume into account, SQL DDL (Data Definition Language) statements will be generated in order to define a star (or similar) schema, such as the one presented in Figure 1.

C.4. ETL design

After dimensional tables and fact tables have been created, they should be populated. Data is extracted from XML documents by using XQuery. Necessary data transformations and cleansing are provided. After the initial data loading, additional data is loaded into data warehouse periodically.

C.5. Physical design

Physical design deals primarily with the optimal selection of indices, which plays a crucial role in optimization of data warehouse performance.

V. PROTOTYPE TOOL

In order to test and verify the proposed methodology, a Java-based prototype tool (Figure 5) has been developed. The prototype reads an XML Schema and conforming XML documents. Schema graph is created automatically from the source XML Schema and shown in the graphical interface. The fact is chosen using the graphical interface and the corresponding dependency graph created semi-automatically. In some cases source XML documents are examined using XQuery language to get information

about relationships. Designer can manually rearrange the dependency graph according to its semantics. Finally the star schema is produced in output.

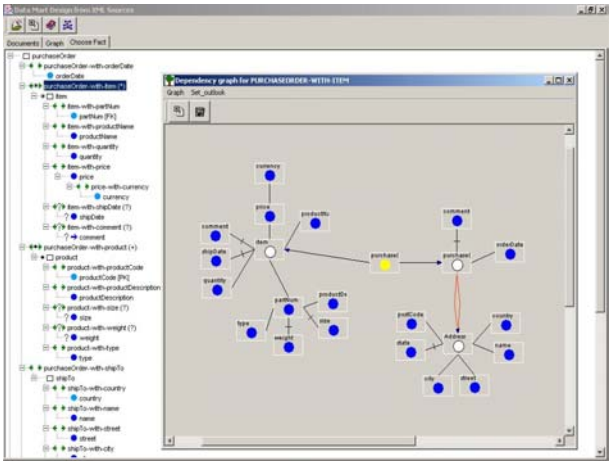

Figure 5 – Prototype tool

VI. CONCLUSION

In this paper we have proposed a methodology for data warehouse design, when sources of data are XML Schemas and conforming XML documents. Special attention is given to conceptual multidimensional design, which is the biggest challenge because of semi-structure nature of XML data. Once a conceptual scheme has been obtained, the logical and physical schemes for the warehouses are derived.

The algorithm has been implemented within a prototype tool which thus acts as a valuable support for the proposed methodology.

REFERENCES

[1] M. Golfarelli, D. Maio, and S. Rizzi, "Conceptual design of data warehouses from E/R schemes", *Proc. HICSS-31*, vol. VII, Kona, Hawaii, pp. 334-343, 1998.

[2] M. Golfarelli, D. Maio, S. Rizzi, "The Dimensional Fact Model: a Conceptual Model for Data Warehouses", *International Journal of Cooperative Information Systems*, vol. 7, n. 2&3, pp. 215-247, 1998.

[3] M. Golfarelli, S. Rizzi, and B. Vrdoljak, "Data warehouse design from XML sources", *Proc. Data Warehousing and OLAP (DOLAP'01)*, Atlanta, USA, 2001.

[4] B. Vrdoljak, M. Banek, S. Rizzi, "Designing Web Warehouses from XML Schemas", *Proc. Int'l Conf. on Data Warehousing and Knowledge Discovery (DaWaK)*, Prague, Czech Republic, 2003.

[5] M- Blaschka, C. Sapia, G. Hofling, and B. Dinter, "Finding Your Way through Multidimensional Data Models", *Proc. DWDOT*, Wien, 1998.

[6] M. Golfarelli, S. Rizzi. Designing the data warehouse: key steps and crucial issues. *Journal of Computer Science and Information Management*, 2(3), 1999.

[7] J. Shanmugasundaram et al., "Relational Databases for Querying XML Documents: Limitations and Opportunities", *Proc. 25th VLDB*, Edinburgh, 1999.

[8] M. Jensen, T. Møller, and T.B. Pedersen, "Specifying OLAP Cubes On XML Data", *Journal of Intelligent Information Systems*, 2001.

[9] 8 T. Niemi, M. Niinimäki, J. Nummenmaa, and P. Thanisch, "Constructing an OLAP cube from distributed XML data", *Proc. DOLAP'02*, McLean, 2002.

[10] J. Pokorny, "Modeling stars using XML", Proc. DOLAP'01, 2001.

[11] World Wide Web Consortium (W3C), "XML Schema", http://www.w3.org/XML/Schema

[12] World Wide Web Consortium (W3C), "XQuery 1.0: An XML Query Language (Working Draft)", http://www.w3.org/TR/xquery/