A Tool for Data Warehouse Design from XML Sources

Marko Banek, Zoran Skočir, Boris Vrdoljak
FER - University of Zagreb, Unska 3, HR-10000 Zagreb, Croatia
E-mail: {marko.banek, zoran.skocir, boris.vrdoljak}@fer.hr

Abstract – Large volumes of XML data exist on the Internet and in information systems of various companies and organizations. Therefore, there is an increasing need of integrating XML data into data warehousing systems. We have proposed a methodology for data warehouse design, when sources of data are XML Schemas and conforming XML documents. A prototype tool has been developed to verify and support our methodology. The tool automates many parts of the conceptual and logical design process, thus helping the designer in designing faster and more accurately. In this paper the main features of the tool for data warehouse design from XML sources are presented.

I. INTRODUCTION

The process of building a data warehousing system includes analysis of different data sources, design of a data warehouse model, definition of transformation and integration processes, construction of a 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. In [5], we proposed methodology for semi-automated design of data warehouses from XML Schemas [11] and conforming XML documents. Special attention was given to conceptual multidimensional design, which was the biggest challenge because of the semi-structured nature of XML data. Once a conceptual scheme is obtained, the logical scheme can be derived. To support the methodology for data warehouse design from XML sources, a Java-based prototype tool has been developed. The tool assists the designer in creating a logical multidimensional scheme starting from XML Schema and corresponding XML documents that are available. The conceptual design is carried out semi-automatically. The result of the conceptual design is a conceptual multidimensional scheme, which represents an input for the logical design process. In output, a star schema is derived.

The paper is structured as follows. In Section II our methodology for data warehouse design from XML sources is explained. Section III describes the features of the prototype tool we have developed to support the methodology. Finally, in Section IV the conclusions are drawn.

II. METHODOLOGY FOR DATA WAREHOUSE DESIGN FROM XML SOURCES

A design methodology is an essential requirement to ensure the success of a complex data warehousing project. The basic phases in data warehouse design, when data sources are either E/R diagrams or relational logical schemes, have been presented in [1]. Some approaches concerning the data warehouse design from XML sources have been proposed in the literature. In [8] DTDs (Document Type Definition) 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. We believe that a physical integration should be achieved in cases when the external data is structured enough to derive a meaningful conceptual model of a warehouse.

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. In [5] a complete methodology that includes conceptual, logical and physical design has been explained, and the functional architecture of the system has been presented. The methodology from [1] and [6] has been adapted in
order to address various issues emerging from the semi-structured nature of XML data. The basic phases of the methodology are briefly explained in the following.

A. Preliminary work

A.1. Analysis

The designer of the data warehouse analyzes XML Schema and XML documents conforming to the XML Schema, as well as the available documentation, in order to determine data semantics, data quality, the number of available XML documents, etc. Requirement specification involves designer and final users.

A.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.

B. Design

The main steps of data warehouse design starting from XML sources are: 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.

B.1. Conceptual design

In our design methodology, the Dimensional Fact Model (DFM) [2] is adopted as the conceptual model. In the DFM a data warehouse is represented by 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 many-to-one relationship, and determine how facts may be aggregated. In other words, each hierarchy includes a set of attributes linked by functional dependencies; for instance, city functionally determines country.

Conceptual design starts from an XML Schema (or several XML Schemas linked by their namespaces). It is primarily based on detecting many-to-one relationships. 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 examining the XML documents that conform 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 in the XML Schema in order to define 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. Attribute values, 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.

The result of conceptual design from XML sources is a conceptual scheme, made according to Dimensional Fact Model.

B.2. Workload definition and data volume acquisition

This phase can be divided in two parts. First, the workload is defined in detail. The 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. The workload is then checked against the conceptual scheme, and this way the conceptual scheme is validated. Second, XQuery [12] 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.

B.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 variations. 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.

B.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.
B.5. Physical design

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

III. PROTOTYPE TOOL

In order to test and verify our methodology, a Java-based prototype tool (Figure 1) has been developed. The tool automates many steps of the conceptual and logical design.

A. Conceptual design

The algorithm for conceptual multidimensional design starting from XML sources consists of the following steps:

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.

The structure of the simplified XML Schema is firstly visualized by using a schema graph (SG). The SG is automatically derived from the XML Schema. The method is adopted from [7], where simpler, but less efficient DTD is still used as a grammar. A schema graph for the XML Schema describing purchase order is shown in Figure 2. The XML Schema has been taken from the W3C’s document [11] and slightly extended.

In addition to the SG 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.

After the schema graph has been produced, a designer can choose a fact among all the vertices and arcs of the SG. It is up to the designer to decide what the event of interest in decision-making is. In order to obtain a meaningful fact schema, it is crucial that the fact is properly chosen. The vertices or arcs representing frequently updated archives are good candidates for defining the facts. When the arcs are chosen as facts, they generally represent many-to-many relationships. In the purchase order example, after the elimination of the items vertex (because it has no content and serves only for emphasizing the structure of the document), the relationship between purchaseOrder and item is chosen as a fact, as shown in Figure 3.
A dependency graph (DG) 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 DG are a subset of the element and attribute vertices of the schema graph, and its arcs represent associations between vertices. The DG is enlarged by recursively navigating the functional dependencies between the vertices of the schema graph.

The building of a DG is a combination of four procedures, as follows.

- **Navigation in the “direction down”**: The 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 DG 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. When the relationship is –to-one, the child element will be included in the DG.

- **Navigation in the “direction up”**: In the direction from the fact to its ascendants (“direction up”) the schema yields no information about the relationship cardinality. In order to enlarge the dependency graph in this direction, the cardinality of child-parent relationships must be stated, which is done by examining the available XML documents using XQuery statements. A part of the query is presented in Figure 4: it counts the number of distinct child elements for each child element and returns the maximum number. If the returned number is greater than one, the relationship is -to-many. In this case the parent element is not included into the DG.

```xquery
max{...
  for $p in
  distinct-values($retValue/child)
  let $expr:= for $exp in $retValue
      where deep-equal($exp/child,$c)
    return $exp/parent
  return count(distinct-values($p))
}
```

**Figure 4. A part of the XQuery query for examining the relationship cardinality**

- **Following the key/keyref mechanism**. The third part of the algorithm concerns the case that a vertex referencing a key vertex is reached in the schema graph. As an example, consider the schema graph for purchase order shown in Figure 2. From the fact, following a -to-one relationship, the item vertex is added to the dependency graph. Then, `productName`, `quantity`, `price`, `currency`, `shipDate`, and `partNum` are added too. Note that the `partNum` vertex is defined as a key reference to the `productCode` attribute. The key element or attribute should be swapped with its parent vertex. As the `productCode` vertex is defined as a key, it is swapped with `product`, which is then dropped because it carries no atomic value. Vertices `productName`, `type`, `weight`, and `size` become the children of `productCode`. During the creation of the dependency graph, these attributes become descendants of the `partNum` attribute. The reason is that they are functionally dependent on `productCode` and that `partNum` references `productCode`. Figure 5 shows the resulting segment of the DG for the purchase order example.

`Figure 5. Following the key/keyref mechanism`

- **Convergence and shared hierarchies**. Whenever a complex type has more than one instance in the SG, and all of the instances have a common ancestor vertex, either a convergence or a shared hierarchy may be implied in the DG. A convergence holds if an attribute is functionally determined by another attribute along two or more distinct paths of to-one associations. For instance there can be two paths of functional dependencies for a geographical hierarchy over the stores: `store → city → region → state` and `store → saleDistrict → state`. It is assumed that there is no inclusion relationship between the sale district and regions, and that every district makes part of only one state. Irrespective of how aggregation is done, each store always belongs to exactly one state.

On the other hand, it often happens that whole parts of hierarchies are replicated twice or more. In this case we talk of a shared hierarchy, to emphasize that there is no convergence. In our approach the examination is made by querying the available XML documents conforming to the given Schema. In the purchase order example, following a to-one relationship from the fact, the `purchaseOrder` vertex is added to the DG. It has two children elements, `shipTo` and `billTo` (see Figure 2), that are of same complex type. Therefore, they have the same combination of sub-elements and attributes. The `purchaseOrder` element is the closest common ancestor of `shipTo` and `billTo`, thus all the instances of the `purchaseOrder` element have to be retrieved. For each `purchaseOrder` instance, the content of the first child, `shipTo`, is compared to the content of the second one, `billTo`, using the `deep-equal` XQuery operator as shown in Figure 6.

```xquery
let $x:= for $c in $retValue
where not(deep-equal($c/first/content, $c/second/content))
return $c
return count($x)
```

**Figure 6. A part of the XQuery query for distinguishing convergence from shared hierarchy**

The query returns the number of couples with different contents. If at least one couple with different contents is counted, a shared hierarchy is introduced. Otherwise, since in principle there still is a possibility that documents
in which the contents of the complex type instances are not equal will exist, the designer has to decide about the existence of convergence by leaning on her knowledge of the application domain. Examining the XML documents, it is found that `shipTo` and `billTo` in some cases have different values and a shared hierarchy is introduced.

The derived DG may then be rearranged in order to create a usable and efficient conceptual scheme. This phase of design necessarily depends on the user requirements and cannot be carried out automatically. The designer usually has to:

- remove some existing vertices from the DG,
- add some new vertices to the DG,
- change the position of some existing vertices in the DG.

All described methods of rearranging the DG are implemented in the prototype tool.

Some parts of the purchase order DG also need rearranging. Considering `item` and `partNum`, only the latter is left. The `comment` and `shipDate` attributes are dropped to eliminate unnecessary details. Finally, the shared hierarchy root is named `customer` in order to clarify its role.

The final steps of building a multidimensional schema include the choice of dimensions and measures. In the purchase order example, `price` and `quantity` are chosen as measures, and the `income` measure is derived as a product of `quantity` and `price`. The dimensions are: `orderDate`, `partNum`, `shipToCustomer`, and `billToCustomer`. The latter two form the shared hierarchy and are joined into the `customer` vertex. The time hierarchy is enriched by introducing vertices `dayOfWeek`, `month`, `noOfWorkingDays` and `year`. The final dependency graph is shown in Figure 7.

After defining dimensions and measures among the vertices of the DG, the dependency graph can easily be translated into a fact scheme as a conceptual multidimensional scheme.

### B. Logical design

After completing the conceptual design by defining dimensions and measures, the tool automatically produces the logical scheme based on the star schema. The star schema consists of one fact table, describing the fact, and a set of dimensional tables. Each dimension table has a single-part primary key. The fact table contains all measures of the fact and a multi-part key, each part referencing a dimensional table as foreign key.

The specified tables can be seen in the GUI, and then the SQL statements for creating the specified tables in a database are produced. The tool connects to the database via JDBC interface and executes the SQL statements. Figure 8 shows the produced SQL statement for creation of the `CUSTOMER` dimension table.

As a general rule, a logical scheme in the relational environment is obtained by translating each n-dimensional fact scheme into one fact table and n dimensional tables. However, the number of dimensional tables may in certain cases be smaller or larger than the number of dimensions in the conceptual model. Such a case is the shared hierarchy on `customer` in the purchase order example.
In the logical model the shared hierarchy is represented by only one dimension table named CUSTOMER, because the hierarchy for the customer to whom the products will be shipped (shipToCustomer) is exactly the same as the hierarchy for the customer who pays for the products ordered (billToCustomer). When modeling two hierarchies that have exactly the same attributes, but used with different meaning, it is sufficient to import the key of the dimension table twice into the fact table. Therefore, the compound key of the fact table in Figure 9 has both shipToCustomer and billToCustomer foreign keys related to only one dimension table CUSTOMER.

Figure 9. The fact table

IV. CONCLUSION AND FUTURE WORK

In this paper we presented a prototype tool we developed to support and verify our methodology for data warehouse design from XML sources. The tool automates many parts of the conceptual and logical design process, thus helping the designer in designing faster and more accurately. If all needed information about the relationships cannot be inferred from the XML Schema, the source XML documents are queried by using XQuery statements, and, if necessary, the designer is asked for help. All the phases of the conceptual and logical design are controlled and monitored by the designer through a graphical interface. At the end of the design process, the tool connects to a database and creates tables according to the derived star schema.

Our future work will examine the integration of many different XML Schemas describing the same domain. For instance, purchase orders can be described by several, independently created XML Schemas. For that purpose, a self-learning framework with reasoning capabilities can be developed. The framework should be based on ontologies. Each ontology of a domain could be incrementally expanded each time a new XML Schema is joined. Moreover, by using ontologies, relational sources can also be integrated with XML data.

REFERENCES