Designing Web Warehouses from XML Schemas

Boris Vrdoljak[1], Marko Banek[1], and Stefano Rizzi[2]

[1] FER – University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
{boris.vrdoljak,marko.banek}@fer.hr
[2] DEIS – University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
srizzi@deis.unibo.it

Abstract. Web warehousing plays a key role in providing the managers with up-to-date and comprehensive information about their business domain. On the other hand, since XML is now a standard de facto for the exchange of semi-structured data, integrating XML data into web warehouses is a hot topic. In this paper we propose a semi-automated methodology for designing web warehouses from XML sources modeled by XML Schemas. In the proposed methodology, design is carried out by first creating a schema graph, then navigating its arcs in order to derive a correct multidimensional representation. Differently from previous approaches in the literature, particular relevance is given to the problem of detecting shared hierarchies and convergence of dependencies, and of modeling many-to-many relationships. The approach is implemented in a prototype that reads an XML Schema and produces in output the logical schema of the warehouse.

1 Introduction

The possibility of integrating data extracted from the web into data warehouses (which in this case will be more properly called *web warehouses* [1]) is playing a key role in providing the enterprise managers with up-to-date and comprehensive information about their business domain. On the other hand, the Extensible Markup Language (XML) has become a standard for the exchange of semi-structured data, and large volumes of XML data already exist. Therefore, integrating XML data into web warehouses is a hot topic.

Designing a data/web warehouse entails transforming the schema that describes the source operational data into a multidimensional schema for modeling the information that will be analyzed and queried by business users. In this paper we propose a semi-automated methodology for designing web warehouses from XML sources modeled by XML Schemas, which offer facilities for describing the structure and constraining the content of XML documents. As HTML documents do not contain semantic description of data, but only the presentation, automating design from HTML sources is unfeasible. XML models semi-structured data, so the main issue arising is that not

Y. Kambayashi, M. Mohania, W. Wöß (Eds.): DaWaK 2003, LNCS 2737, pp. 89-98, 2003.

all the information needed for design can be safely derived. In the proposed methodology, design is carried out by first creating a *schema graph*, then navigating its arcs in order to derive a correct multidimensional representation in the form of a *dependency graph* where arcs represent inter-attribute relationships. The problem of correctly inferring the needed information is solved by querying the source XML documents and, if necessary, by asking the designer's help.

Some approaches concerning related issues have been proposed in the literature. In [4] a technique for conceptual design starting from DTDs [12] is outlined. That approach is now partially outdated due to the increasing popularity of XML Schema; besides, some complex modeling situations were not specifically addressed in the paper. In [5] and [6] DTDs are used as a source for designing multidimensional schemas (modeled in UML). Though that approach bears some resemblance to ours, the unknown cardinalities of relationships are not verified against actual XML data, but they are always arbitrarily assumed to be to-one. Besides, the *id*/*idref* mechanism used in DTDs is less expressive than *key*/*keyref* in XML Schema. The approach described in [8] 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 [9], the author shows how to use XML to directly model multidimensional data, without addressing the problem of how to derive the multidimensional schema.

Differently from previous approaches in the literature, in our paper particular relevance is given to the problem of detecting shared hierarchies and convergence of dependencies, and of modeling many-to-many relationships within hierarchies. The approach is implemented in a prototype that reads an XML Schema and produces in output the star schema for the web warehouse.

2 Relationships in XML Schema

The structure of XML data can be visualized by using a *schema graph* (SG) derived from the Schema describing the data. The method is adopted from [10], where simpler, but less efficient DTD is still used as a grammar. The SG for the XML Schema describing a purchase order, taken from the W3C's document [14] and slightly extended, is shown in Fig. 1. 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. Attributes and sub-elements are not distinguished in the 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.

3 From XML Schema to Multidimensional Schema

In this section we propose a semi-automatic approach for designing a web warehouse starting from an XML Schema. The methodology consists of the following steps:

1. Preprocessing the XML Schema.
2. Creating and transforming the SG.
3. Choosing facts.
4. For each fact:
   - 4.1 Building the dependency graph from the SG.
   - 4.2 Rearranging the dependency graph.
   - 4.3 Defining dimensions and measures.
   - 4.4 Creating the logical schema.

Given a fact, the *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 whose vertices are a subset of the element and attribute vertices of the SG, and whose arcs represent associations between vertices. The root of the DG corresponds to the fact.

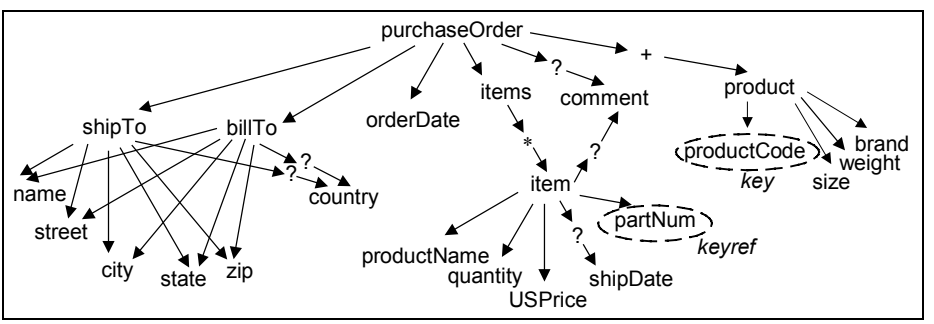

Fig. 1. The Schema Graph

While in most cases the hierarchies included in the multidimensional schema represent only to-one associations (sometimes called *roll-up* relationships since they support the roll-up OLAP operator), in some applications it is important to model also many-to-many associations. For instance, suppose the fact to be modeled is the sales of books, so *book* is one of the dimensions. Although books that have many authors certainly exist, it would be interesting to aggregate the sales by author. It is remarkable that summarizability is maintained through many-to-many associations, if a normalized weight is introduced [7]. Besides, some specific solutions for logical design in presence of many-to-many associations were devised [11]. However, since modeling many-to-many associations in a warehouse should be considered an exception, their inclusion in the DG is subject to the judgment of the designer, who is supposed to be an expert of the business domain being modeled.

After the DG has been derived from the SG, it may be rearranged (typically, by dropping some uninteresting attributes). This phase of design necessarily depends on the user requirements and cannot be carried out automatically; since it has already been investigated (for instance in [2]), it is considered to be outside the scope of this paper. Finally, after the designer has selected dimensions and measures among the vertices of the DG, a logical schema can be immediately derived from it.

3.1 Choosing Facts and Building Dependency Graphs

The relationships in the Schema can be specified in a complex and redundant way. Therefore, we transform some structures to simplify the Schema, similarly as DTD was simplified in [10] and [6]. A common example of Schema simplification concerns the *choice* element, which denotes that exactly one of the sub-elements must appear in a document conforming to that Schema. The *choice* element is removed from the schema and a *minOccurs* attribute with value 0 is added to each of its sub-elements. The resulting simplified structure, although not being equivalent to the *choice* expression, preserves all the needed information about the cardinalities of relationships.

After the initial SG has been created [10], it must undergo two transformations. First, all the key attributes or elements are located and swapped with their parent vertex in order to explicitly express the functional dependency relating the key with the other attributes and elements. Second, some vertices that do not store any value are eliminated. A typical case is an element that has only one sub-element of complex type and no attributes, and the relationship with its sub-element is to-many. We name such an element a *container*. Note that, when a vertex v is deleted, the parent of v inherits all the children of v and their cardinalities.

The next step is choosing the fact. The designer chooses the fact among all the vertices and arcs of the SG. An arc can be chosen as a fact if it represents a many-to-many relationship. For the purchase order SG presented in Fig. 1, after the *items* element has been eliminated as a container, the relationship between *purchaseOrder* and *item* is chosen as a fact, as in Fig. 2.

For each fact f, the corresponding DG must be built by including a subset of the vertices of the SG. The DG is initialized with the root f, to be enlarged by recursively navigating the relationships between vertices in the SG. After a vertex v of the SG is inserted in the DG, navigation takes place in two steps:

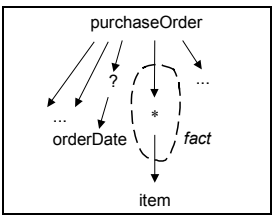

Fig. 2. Choosing a fact

1. *For each vertex w that is a child of v in the SG*: When examining relationships in the direction expressed by arcs of the SG, the cardinality information is expressed either explicitly by "?", "*" and "+" vertices, or implicitly by their absence. If *w* corresponds to an element or attribute in the Schema, it is added to the DG as a child of *v*; if *w* is a "?" operator, its child is added to the DG 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 (see Section 4.5): if it is to-many, the designer decides whether the many-to-many relationship between *v* and *u* is interesting enough to be inserted into the DG or not.
2. *For each vertex z that is a parent of v in the SG*: When examining relationships in this direction, vertices corresponding to "?", "*" and "+" operators are skipped since they only express the cardinality in the opposite direction. Since the Schema yields no further information about the relationship cardinality, it is necessary to examine the actual data by querying the XML documents conforming to the Schema (see Section 4.5). If a to-one relationship is detected, *z* is included in the DG.

Whenever a vertex corresponding to a *keyref* element is reached, the navigation algorithm "jumps" to its associated *key* vertex, so that descendants of the key become descendants of the *keyref* element. A similar approach is used in [3], where the operational sources are represented by a relational schema, when a foreign key is met during navigation of relations. See for instance Fig. 3, showing the resulting DG for the purchase order example. From the fact, following to-one relationship, the *item* vertex is added to the DG. Vertex *productCode* is defined to be a key (Fig.1). It is swapped with *product*, which then is dropped since it carries no value. The *partNum* vertex is a child of *item* and is defined as a key reference to the *productCode* attribute. *size*, *weight* and *brand*, the children of *productCode*, become descendants of the *partNum* attribute in the DG.

3.2 Querying XML Documents

In our approach, XQuery [15] is used to query the XML documents in three different situations:

1. examination of convergence and shared hierarchies
2. searching for many-to-many relationships between the descendants of the fact in SG
3. searching for to-many relationships towards the ancestors of the fact in the SG

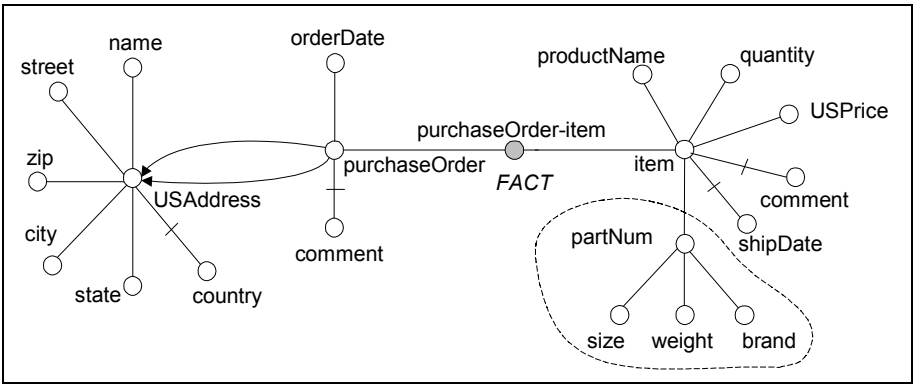

Fig. 3. The DG for the purchase order example

Note that, since in all three cases querying the documents is aimed at counting how many distinct values of an attribute *v* are associated to a single value of an attribute *w*, it is always preliminarily necessary to determine a valid identifier for both *v* and *w*. To this end, if no *key* is specified for an attribute, the designer is asked to define an identifier by selecting a subset of its non-optional sub-elements.

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. On the other hand, it often happens that whole parts of hierarchies are replicated two or more times. 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, *shipTo* and *billTo* (Fig. 1), that have the same complex type *USAddress*. 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 Fig. 4.

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

Fig. 4. A part of the XQuery query for distinguishing convergence from shared hierarchy

By using the *COUNT* function, 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. In our example, supposing it is found that *shipTo* and *billTo* have different values in some cases, a shared hierarchy is introduced.

Many-to-Many Relationships between the Descendants of the Fact. While in most cases only to-one associations are included into the DG, there are situations in which it is useful to model many-to-many associations. Consider the SG in Fig. 5, modeling the sales of the books, where the *bookSale* vertex is chosen as the fact. After the *book* vertex is included into the DG, a to-many relationship between *book* and *author* is detected. Since including a one-to-many association would be useless for aggregation, the available XML documents conforming to the *bookSale* Schema are examined by using XQuery to find out whether the same author can write multiple books. A part of the query is presented in Fig. 6: it counts the number of distinct books (i.e. different *parent* elements) for each author (*child*) and returns the maximum number. If the returned number is greater than one, the relationship is many-to-many, and the designer may choose whether it should be included in the DG or not. If the examination of the available XML documents has not proved that the relationship is many-to-many, the designer can still, leaning on his or her knowledge, state the relationship as many-to-many and decide if it is interesting for aggregation.

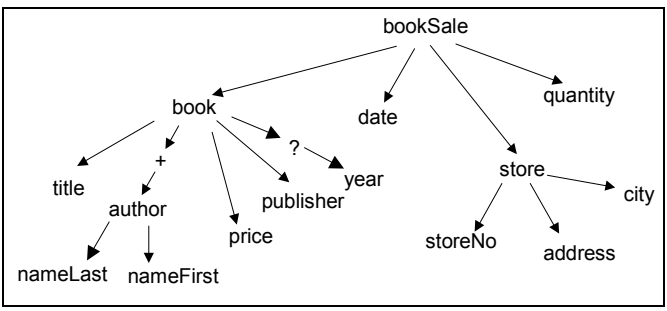

Fig. 5. The book sale example

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

Fig. 6. A part of a query for examining many-to-many relationships

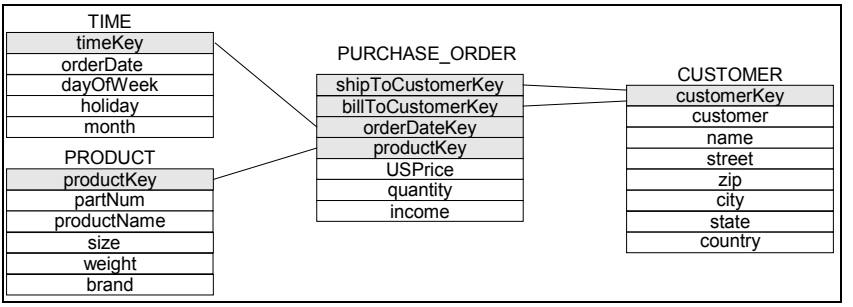

Fig. 7. The star schema for the purchase order example

To-Many Relationships towards the Ancestors of the Fact. This type of search should be done because the ancestors of the fact element in the SG will not always form a hierarchically organized dimension in spite of the nesting structures in XML. When navigating the SG upwards from the fact, the relationships must be examined by XQuery since we have no information about the relationship cardinality, which is not necessarily to-one. The query is similar to the one for examining many-to-many relationships, and counts the number of distinct values of the parent element corresponding to each value of the child element.

3.3 Creating the Logical Scheme

Once the DG has been created, it may be rearranged as discussed in [3]. Considering for instance the DG in Fig. 3, we observe that there is no need for the existence of both *purchaseOrder* and *purchaseOrder-item*, so only the former is left. Considering *item* and *partNum*, only the latter is left. The *comment* and *shipDate* attributes are dropped to eliminate unnecessary details. Finally, attribute *USAddress* is renamed into *customer* in order to clarify its role.

The final steps of building a multidimensional schema include the choice of dimensions and measures as described in [2]. In the purchase order example, *USPrice* and *quantity* are chosen as measures, while *orderDate*, *partNum*, *shipToCustomer*, and *billToCustomer* are the dimensions.

Finally, the logical schema is easily obtained by including measures in the fact table and creating a dimension table for each hierarchy in the DG. Fig. 7 shows the resulting star schema corresponding to the DG in Fig. 3; note how the shared hierarchy on customer is represented in the logical model by only one dimension table named *CUSTOMER*, and how a derived measure, *income*, has been defined by combining *quantity* and *USPrice*. In the presence of many-to-many relationships one of the logical design solution proposed in [11] is to be adopted.

4 Conclusion

In this paper we described an approach to design a web warehouse starting from the XML Schema describing the operational source. As compared to previous approaches

based on DTDs, the higher expressiveness of XML Schema allows more effective modeling. Particular relevance is given to the problem of detecting shared hierarchies and convergences; besides, many-to-many relationships within hierarchies can be modeled.

The approach is implemented in a Java-based prototype that reads an XML Schema and produces in output the star schema for the web warehouse. Since all the needed information cannot be inferred from XML Schema, in some cases the source XML documents are queried using XQuery language, and if necessary, the designer is asked for help. The prototype automates several parts of the design process: preprocessing the XML Schema, creating and transforming the schema graph, building the dependency graph, querying XML documents. All phases are controlled and monitored by the designer through a graphical interface that also allows some restructuring interventions on the dependency graph.

References

[1] S. S. Bhowmick, S. K. Madria, W.-K. Ng, and E. P. Lim, "Web Warehousing: Design and Issues", Proc. DWDM'98, Singapore, 1998.

[2] 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.

[3] 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.

[4] M. Golfarelli, S. Rizzi, and B. Vrdoljak, "Data warehouse design from XML sources", Proc. DOLAP'01, Atlanta, pp. 40-47, 2001.

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

[6] M. Jensen, T. Møller, and T.B. Pedersen, "Converting XML Data To UML Diagrams For Conceptual Data Integration", Proc. DIWeb'01, Interlaken, 2001.

[7] R. Kimball. "The data warehouse toolkit". John Wiley & Sons, 1996.

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

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

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

[11] I.Y. Song, W. Rowen, C. Medsker, and E. Ewen, "An analysis of many-to-many relationships between fact and dimension tables in dimensional modeling", Proc. DMDW, Interlaken, Switzerland, pp. 6.1-6.13, 2001.

[12] World Wide Web Consortium (W3C), "XML 1.0 Specification", http://www.w3.org/TR /2000/REC-xml-20001006.

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

[14] World Wide Web Consortium (W3C), “XML Schema Part 0: Primer”, http://www.w3.org /TR/xmlschema-0/.
[15] World Wide Web Consortium (W3C), “XQuery 1.0: An XML Query Language (Working Draft)”, http://www.w3.org/TR/xquery/.