Automating Conceptual Design of Web Warehouses

Boris Vrdoljak, Marko Banek
FER – University of Zagreb
Zagreb, Croatia

Stefano Rizzi
DEIS - University of Bologna
Bologna, Italy

***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 conceptual design of web warehouses from XML sources modeled by XML Schemas. In our methodology, conceptual design is carried out by first creating a Schema graph, then navigating the functional dependencies expressed by its arcs in order to derive a correct multidimensional representation. 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. The approach is implemented in a prototype that reads an XML Schema and produces in output the conceptual scheme for the web warehouse.**

***Index terms*—e-Commerce & e-Government, Data Warehousing, Software Engineering, XML**

I. Introduction

Data warehousing systems support the enterprises in the process of extracting useful, concise and handy information for decision-making out of the huge quantity of data stored in their information systems. Since conventional design techniques cannot be successfully applied to build data warehouses, a substantial effort has been made to devise ad hoc methodologies for seamlessly integrating data from heterogeneous sources and putting them into multidimensional form in order to feed them into the warehouse and make them accessible to OLAP (On-Line Analytical Processing) and reporting tools.

Recently, as the Internet has evolved into a global platform for information exchange, and e-commerce has emerged as a strongly competing reality, a large number of organizations view the web as an integral part of their communication and business. In this process, the possibility of integrating data extracted from the web into data warehouses (which in this case will be more properly called *web warehouses* [2]) 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 [1], and large volumes of XML data already exist. Therefore, integrating XML data into web warehouses is a hot topic.

XML documents can be associated with or validated against either a Document Type Definition (DTD) [13] or an XML Schema [14]. Although XML data are self-describing, important information about their structure, that is necessary for directly building a warehouse, cannot be obtained without seeing their DTD or XML Schema.

XML Schemas considerably extend the capabilities of DTDs, especially from the point of view of data typing and constraining. In particular, the cardinality can be specified in more detail. Furthermore, XML Schemas introduce more powerful and flexible mechanisms for defining keys and their references in the way that is similar to key and foreign key mechanism in relational databases. Because of all its advantages comparing to the DTD, XML Schema is becoming more used than DTD.

In this paper we propose a semi-automated methodology for conceptual design of web warehouses from XML sources modeled by XML Schemas. Several conceptual models for data/web warehouses were devised in the literature [3]; in this paper we will adopt the *Dimensional Fact Model* (DFM) described in [6]. We believe that conceptual design has a key role in determining the quality of the warehouse in terms of documentation, user satisfaction, and reusability; once a conceptual scheme has been obtained, the logical and physical schemes for the warehouse are mainly determined by the target platform for implementation.

In general, conceptual design of data/web warehouses entails transforming a schema that describes source operational data into a multidimensional schema for modeling the information that will be analyzed and queried by business users. For instance, [5] discusses how this can be achieved by navigating many-to-one relationships when the source operational data are described by Entity/Relationship schemas. When the sources are modeled by XML Schemas, two main issues arise: firstly, since XML models semi-structured data, not all the information needed for design can be safely derived; secondly, two different ways of representing relationships in XML Schemas are possible, each achieving different expressive power. In our methodology, conceptual design is carried out by first creating a Schema graph, then navigating the functional dependencies expressed by its arcs in order to derive a correct multidimensional representation. 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. The approach is implemented in a prototype which reads an XML Schema and produces in output the conceptual scheme for the web warehouse.

The paper is structured as follows. After briefly discussing some related approaches in Section II and explaining multidimensional modeling in Section III, in Section IV we show how relationships are modeled in XML Schemas. In Section V we propose our methodology for conceptual design

and show how an XML Schema can be converted into a multidimensional schema that conceptually models a web warehouse. Finally, in Section VI the conclusions are drawn.

II. Related literature

The approach described in [10] 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 [11], the author shows how to use XML to directly model multidimensional data, without addressing the problem of how to derive the multidimensional schema.

In [7] a technique for conceptual design starting from DTDs 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 [8], 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 are always assumed to be -to-one. Besides, the *id*/*idref* mechanism used in DTDs is less expressive than *key*/*keyref* in XML Schema.

One alternative approach to design from XML sources consists in first translating them into an equivalent relational schema, then starting from the latter to design the warehouse. Some approaches for translating XML documents into a relational database are proposed in the literature, both leaning on the DTD [9][12] or not [4], but insufficient emphasis is given to the problem of determining the cardinality of relationships, which instead has a primary role in multidimensional design.

III. MULTIDIMENSIONAL MODELING

Data from heterogeneous sources are collected and integrated into the data/web warehouse, which is aimed to support complex data analysis and decision making process. In order to make the data accessible to OLAP and reporting tools and enable efficient analysis of a large amount of data, a multidimensional data model is used in the warehouse.

The Dimensional Fact Model [6] is a conceptual model, in which a data/web 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. Figure 1 presents a fact scheme describing purchase orders as a fact, with *unitPrice*, *quantity* and *income* as measures. *Dimensions* are discrete attributes which determine the minimum granularity adopted to represent facts. The dimensions in the purchase order example are product, customer and date. *Hierarchies* are made up of discrete dimension attributes linked by -to-one relationship, and determine how facts may be aggregated. In our example, there are hierarchies: *customerID* → *city* → *country*, *productID* → *brand*, and *date* → *month*. In other words, each hierarchy includes a set of attributes linked by functional dependences; for instance, *city* functionally determines *country* and *productID* determines *brand*.

When building the fact scheme starting from an E/R scheme, the fact scheme is constructed by navigating the functional dependences starting from the chosen fact and by defining dimensions, measures and hierarchies. A fact may be represented either by an entity or by an n-ary relationship.

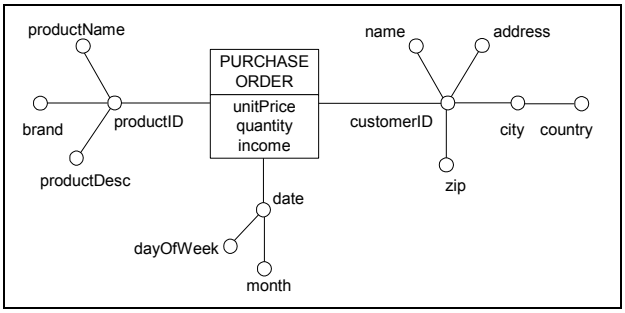

Figure 1. Fact scheme

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.

In the case of multidimensional database storage, data are stored in an array structure similar to the programming language array. On the other hand, 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 2 shows the star schema for the purchase order example. Every element of the multi-part key in the fact table is a foreign key to a single dimension table.

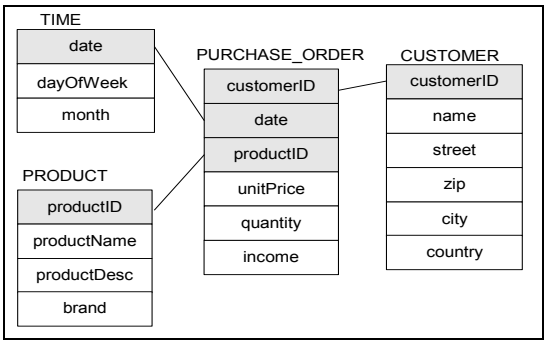

Figure 2. Star schema

In this paper we focus on using XML Schema and XML data as a source for designing web 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.

IV. Relationships in XML Schema

An XML Schema consists of type definitions, which can be derived from each other, and element declarations. The possibility of separating an element declaration from the definition of its type enables sharing and reusing of simple and composite types. The structure of XML data can be visualized by a Schema graph derived from a Schema describing the XML data source; the vertices of a Schema graph either correspond to elements/attributes or describe cardinalities of the relationships between them. The graph contains only data that are relevant for conceptual design of a web warehouse. Relationships precisely described in a Schema conform to only four relationship types; attributes and elements are not distinguished. The method has been adopted from [12], where DTD has still been used as a grammar.

The basic principles for representing an XML Schema by a Schema graph will be discussed with reference to the purchase order example, taken from the W3C's document [15]. A portion of an XML document describing a purchase order is presented in Figure 3.

```
<?xml version="1.0"?>
<purchaseOrder
       orderDate="1999-10-20">
       <shipTo country="US">
          <name>Alice Smith</name>
                ...
       </shipTo>
       <billTo country="US">
          <name>Robert Smith</name>
                ...
       </billTo>
       <items>
                ...
       </items>
</purchaseOrder>
```

Figure 3. XML data describing a purchase order

The purchase order document consists of a main element, *purchaseOrder*, and the sub-elements *shipTo*, *billTo*, and *items*. These sub-elements in turn contain other sub-elements. *orderDate* is an attribute of the *purchaseOrder* element.

Elements that contain sub-elements or carry attributes have complex types. On the other hand, simple type elements contain numbers, strings, dates, etc. and are neither allowed to have sub-elements nor attributes. Attributes always have simple types. The document conforms to the XML Schema presented in Figure 4.

The *purchaseOrder* element is defined as a complex type *PurchaseOrderType*. In defining *PurchaseOrderType*, two of the element declarations, for *shipTo* and *billTo*, associate different element names with the same complex type, namely *USAddress*.

Since our methodology for conceptual design is based on detecting many-to-one relationships, in the following we will focus on the way those relationships can be expressed in the XML Schema. Two different ways of specifying relationships exist: by sub-elements and by using *key* and *keyref* elements.

A. Modeling relationships by sub-elements

Relationships in XML Schema can be specified by sub-elements with different cardinalities. An element is required to appear in the document when the value of the *minOccurs* attribute in its declaration is 1 or more. The maximum number of times an element may appear is determined by the value of a *maxOccurs* attribute. The default value for both the *minOccurs* and the *maxOccurs* attributes is 1. On the other hand, attributes may appear once or not at all. The occurrence of an attribute can be declared by setting the value of the *use* attribute in the Schema to *required* or *optional*.

```
<xsd:element name="purchaseOrder"
       type="PurchaseOrderType"/>
...

<xsd:complexType
       name="PurchaseOrderType">
<xsd:sequence>
<xsd:element name="shipTo"
             type="USAddress"/>
<xsd:element name="billTo"
             type="USAddress"/>
<xsd:element ref="comment"
             minOccurs="0"/>
<xsd:element name="items"
             type="Items"/>
</xsd:sequence>
<xsd:attribute name="orderDate"
             type="xsd:date"/>
</xsd:complexType>
...
```

Figure 4. Purchase order schema

In the Schema graph, we use the operators from the DTD element type declarations because of their simplicity. Concerning the greatest number of times the same sub-element may appear within an element, we distinguish between two general types of relationships: -to-one relationship and -to-many relationship. On the other hand, if a sub-element is optional, it might not appear at all. Consequently, four general types of relationships are distinguished:

- -to-one (the sub-element or attribute appears exactly once within its parent element),
- optional –to-one (marked ?; the sub-element or attribute may appear once or not at all),
- -to-many (marked +, the sub-element appears once or more) and
- optional –to-many (marked *; the sub-element may appear zero or more times).

The Schema graph for the Schema describing a purchase order is shown in Figure 5.

The default cardinality is exactly one and in that case no operator is shown. Element *item* is defined in the Schema as a sub-element of the element *items* with the values of its *minOccurs* and *maxOccurs* attributes set to 0 and "unbounded", respectively. Therefore, there is a "*" operator assigned to the connection between *items* and *item* in Figure 5. If the *minOccurs* attribute of an element is, for instance, set to "2" and *maxOccurs* to "10", the useful information we get from these values is that the element must occur and it can occur more than once, so there is a non-optional -to-many relationship that will be represented by a "+" operator in the Schema graph. The *comment* element is optional within *PurchaseOrderType* because the value of the *minOccurs* attribute in its declaration is 0. Therefore, it does not have to appear in the XML document in Figure 3.

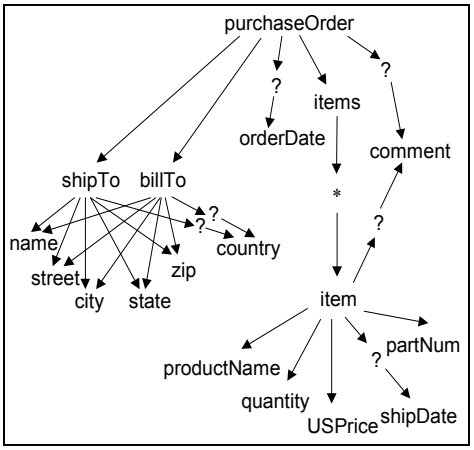

Figure 5. Schema graph for a purchase order

To derive a fact scheme and enable multidimensional analysis of data, it is necessary to find -to-one relationships. The presented classification with only four types of relationships preserves the information about those relationships and eliminates unnecessary details.

When creating a Schema graph from the Schema, only the operators indicating the relationship in the direction from the parent element to its child element can be marked. The cardinality in the opposite direction cannot be found out by exploring the Schema. Only by exploring the data that conforms to the Schema or by having some knowledge about the domain described by the Schema, it can be concluded about the cardinality in the direction from a child element to its parent element.

B. Modeling relationships by key and keyref elements

In XML Schema the *key* and *keyref* elements are 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 has to be of a simple type. By using *keyref* elements, keys can be referenced. The advantage of this mechanism is that not just attribute values, but also element content and their combinations can be declared to be keys. Further, *key* and *keyref* elements are specified to hold within the scope of particular elements.

Figure 6 presents a part of a Schema graph where the *number* attribute is defined as a key for the *part* element and, for each value of the *partNum* attribute, there must exist a *number* attribute with the same value. *part* and *partNum* attributes must be of the same simple type.

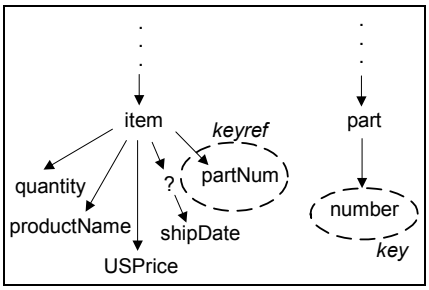

Figure 6. Schema graph with *key* and *keyref*

Figure 7 shows a part of the declaration of the parent element of the *part* element. XML Schemas allow specifying the scope for each key by means of an XPath expression [16]. In the example in Figure 7, the *key* element is named *partKey*. The *number* attribute of the *part* element is specified as the key by means of the *selector* and the *field* sub-elements of the *key* element. The *selector* element specifies an XPath expression relative to instances of the element being declared. In our example, the *selector* specifies that the key is a descendant of the *part* element. The *field* element specifies XPath expression relative to each element selected by a *selector*.

```
<key name="partKey">
  <selector xpath="part"/>
  <field xpath="@number"/>
</key>
```

Figure 7. The *key* element

The value of the *refer* attribute of a *keyref* element should be the name of the key it references. By using *selector* and *field* elements, the *partNum* attribute of the *item* element is specified in Figure 8 as a reference to the *partKey*, i.e. to the *number* attribute that is defined as a key.

```
<xsd:element name="item"
  minOccurs="0"
  maxOccurs="unbounded">
 <xsd:complexType>
   ...
   <xsd:attribute name="partNum"
    type="SKU" use="required"/>
 </xsd:complexType>
 <xsd:keyref name="part_fKey"
       refer="partKey">
   <xsd:selector xpath=".//"/>
   <xsd:field xpath="@partNum"/>
 </xsd:keyref>
</xsd:element>
```

Figure 8. The *keyref* element

In conclusion, using *key* and *keyref* elements not only enables referencing other elements and attributes, but it also provides functional dependencies. The *key*/*keyRef* mechanism may be applied to any element and attribute content, as well as their combinations, and the scope of the constraint can be precisely specified.

V. From XML Schema to multidimensional schema

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

1. Preprocessing the XML Schema.
2. Creating and transforming a Schema graph.
3. Choosing facts.
4. For each fact:
   - 4.1 Building an attribute tree from the Schema graph.
   - 4.2 Rearranging the attribute tree.
   - 4.3 Defining dimensions and measures.

The *attribute tree* is an intermediate structure used to move towards a multidimensional representation of data. After the attribute tree has been built from the Schema graph, it can be rearranged, and dimensions and measures are defined. However, this phase of conceptual design necessarily depends on the user requirements and cannot be carried out automatically. The goal of this paper is to describe only the steps of conceptual design that can be performed automatically or semi-automatically.

A. Preprocessing the Schema

The relationships in the Schema can be specified in a complicated and redundant way. Therefore, we transform some structures to simplify the Schema, similarly as DTD was simplified in [7]. There are also many Schema structures that are neither relevant in detecting relationships nor carry any data content, so they have no impact on the later steps of our algorithm and can be excluded from the Schema.

The transformations for simplifying a Schema include converting a nested definition into a flat representation. For instance, if there is a *choice* element in an element declaration, exactly one of the sub-elements declared inside the *choice* element must appear in a document conforming to that Schema. An example is shown in Figure 9**.**

```
<xsd:element name="item"
...

<choice>
  <element name="priceUSD"
  type="s:priceType"/>
  <element name="priceEUR"
  type="s:priceType"/>
</choice>
...
```

Figure 9. The *choice* element of the Schema

Using the *choice* element, it is defined that the price of an ordered item can be expressed either in US dollars or in euros. From our point of view, the important information here is only that both elements are optional, and they cannot appear more than once. The fact that exactly one of them must occur as sub-element of *item* has no significance, as it is unknown which one. The resulting simplified structure, although not being equal with the choice expression, preserves all the needed information about the cardinalities of relationships.

Figure 10 shows the same part of the schema after its simplification. The *choice* element is removed from the schema and a *minOccurs* attribute is added to each of the two prices elements: *priceUSD* and *priceEUR*, always with value "0".

```
<xsd:element name="item"
...

<xsd:element name="priceUSD"
  type="priceType"
  minOccurs="0"/>
<xsd:element name="priceEUR"
  type="priceType"
  minOccurs="0"/>
...
```

Figure 10. Schema preprocessing

As another example of Schema preprocessing, when an element contains many identical sub-elements on the same level, they are all merged into one sub-element with the *maxOccurs* value "unbounded".

B. Creating and transforming a Schema graph

After the Schema has been simplified, a Schema graph representing its structure can be created. Our prototype for conceptual design from XML sources presents the graph to the designer as shown in Figure 11. The Schema graph describing the purchase order, presented in Figure 4, is used as an example.

After the designer has seen the initial Schema graph, the next step of the algorithm consists in eliminating "containers". A container is an element that has only one sub-element of a complex type and no attributes, and the relationship between the container and its sub-element is -to-many. Since the container does not store any value itself and gives no information other than that it contains other elements, it should neither be chosen as a fact, nor be included into the dimensional hierarchy in the conceptual model of the web warehouse. Therefore, in our algorithm the containers are eliminated. In the purchase order example, the *items* element is marked as a container in Figure 11 (marked "C") and eliminated from the Schema graph, as shown in Figure 12. The parent of the container gets all the container's children and the relationship between them has the same cardinality as it was between the container and its children.

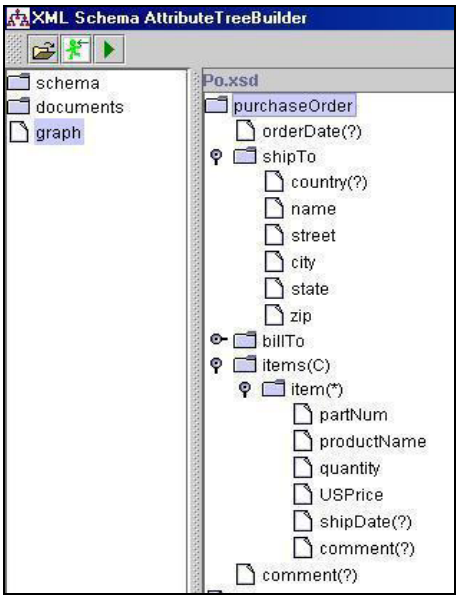

Figure 11. Prototype for conceptual design – schema graph presented

Further, all the *key* and *keyref* attributes or elements are located and the transformation of the "primary key" part of the Schema graph is done. The example is shown in Figure 13.

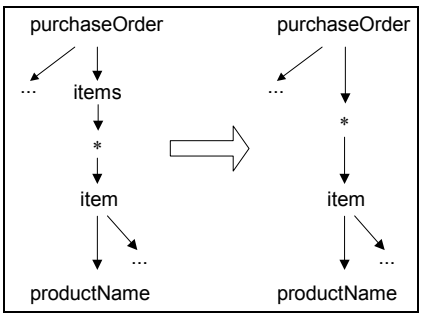

Figure 12. Container elimination

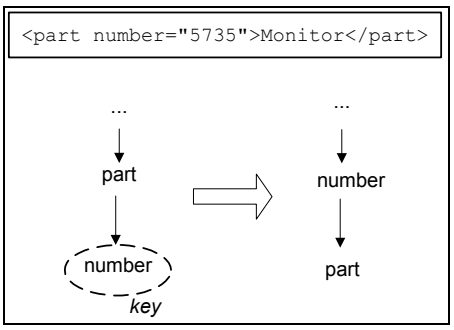

Figure 13. Key transformation

The *number* attribute is defined as the key. The *part* element has its own value too (it is "Monitor" in the line presented above). *part* and *number* are swapped after the transformation. In case the *part* element had not its own value, it would be dropped from the graph and only the *number* attribute would remain. In both cases, all the necessary information would remain in the graph.

C. *Choosing facts*

Our prototype for conceptual design of web warehouses starting from XML Schemas allows the designer to choose the fact among all the vertices and arcs of the Schema graph. In order to obtain a meaningful fact scheme, it is crucial that the fact is chosen properly. It is up to designer to decide what the event of interest for the decision making process is. Vertices or arcs representing frequently updated archives are good candidates for defining facts. When arcs are chosen as facts, they generally represent many-to-many relationships.

For the purchase order Schema graph (Figure 4), after the *items* element has been eliminated as a container (Figure 12), the many-to-many relationship between *purchaseOrder* and *item* is chosen as a fact, as shown in Figure 14.

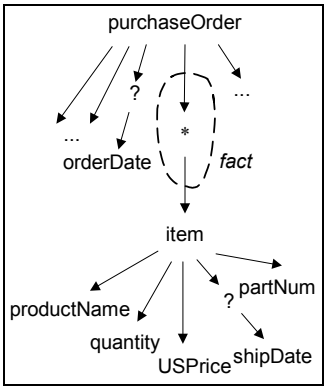

Figure 14. Choosing a fact

By choosing this relationship as a fact, once a web warehouse is made, it will be possible to find out how many items of a certain kind (products) have been ordered in a given order, what is the revenue for a product on a given date, etc.

D. *Building the attribute tree*

Given a Schema graph and a fact *F* chosen by the designer, we call *attribute tree* the tree such that:

1. the root corresponds to the vertex or arc representing the fact *F* in the Schema graph;
2. every other vertex corresponds to a vertex of the Schema graph;
3. for each vertex *v*, the corresponding attribute in the Schema graph functionally determines all the attributes corresponding to the descendants of *v*.

The vertices of the attribute tree are a subset of the element and attribute vertices of the Schema graph. The attribute tree is initialized with the fact vertex *F*; then, it is enlarged by recursively navigating the functional dependencies between

the vertices of the Schema graph. Each vertex *v* inserted in the attribute tree is expanded as follows:

1. *For each vertex w that is a child of v in the Schema graph*: When examining relationships in the same direction as in the Schema graph, 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 attribute tree as a child of *v*; if *w* is a "?" operator, its child is added to the attribute tree as a child of *v*; if *w* is a "*" or "+" operator, no vertex is added.
2. *For each vertex z that is a parent of v in the Schema graph*: When examining relationships in this direction, vertices corresponding to "?" and "*" 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. This is done by counting the number of distinct values of *z* corresponding to each value of *v* using the XQuery language [17]. If a -to-many relationship is detected, *z* is not included in the attribute tree. 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 attribute tree.

An XML Schema that describes the analysis of web site traffic can be taken as an example for examining relationships in the direction from the fact to its ancestors. In this example, the site is a hierarchical directory of web pages consisting of categories such as "World News", "Sport" etc., where a URL can belong to more than one category. For instance, the "Olympic Games" page *www.fastestnews.com/olympics* can belong to both the "Sport" category and the "World News" category. It is supposed that the web administrator wants to organize XML files containing the access data for every category separately. In the Schema graph, *category* will be parent of *url* and the chosen fact will be a descendant of *url.* When building the attribute tree in the upwards direction, the relationship from *url* to *category* should be examined by using XQuery since we have no information about the relationship cardinality. Since in our example each URL can belong to many categories, the relationship from *url* to *category* is -to-many. Therefore, the resulting attribute tree will not contain the *category* vertex.

In some cases it may happen that two attributes in the attribute tree are connected by two or more directed paths. This is called a *convergence* and in this case the attribute tree is actually becoming a graph. As an example, there can be two different hierarchies for the store dimension: *store* → *city* → *region* → *state* and *store* → *sales district* → *state*. It is supposed that there is no inclusion relationship between sales district and regions and that every district makes a part of only one state. No matter how the aggregation is done, each store always belongs to one state. Therefore, the two directed paths starting from the *store* vertex will converge in the *state* vertex.

On the other hand, it often happens that whole parts of hierarchies are replicated two or more times. For instance, there can be two or more different temporal dimensions in the same attribute tree and all of them can have a *month* → *year* hierarchy. This can be emphasized by introducing a specific graphical representation called *shared hierarchy*.

For every complex type that has more than one instance in the Schema graph, where all of the instances have a common ancestor vertex, it is necessary to understand whether this implies a convergence or a shared hierarchy. The examination is made by using the available XML documents conforming to the given Schema. All the instances of the common ancestor vertex should be found in the documents. For each of them it should be examined, by using XQuery, whether every pair of the complex type instances has the same content. If the content is always the same, we still cannot be sure that it is a convergence. It is possible that documents in which the contents of the complex type instances are not equal exist, but we do not have them. Therefore, we ask the designer if it is a convergence. If there is no convergence for that complex type, than we have a shared hierarchy.

As already mentioned, in the purchase order example the relationship between *purchaseOrder* and *item* is chosen as a fact. From the fact, following a -to-one relationship, the *purchaseOrder* vertex is added to the attribute tree. It has two children, *shipTo* and *billTo* (Figure 4), that have the same complex type *USAddress*. All the instances of the *purchaseOrder* elements have to be found in the available XML documents. For each of them the content of *shipTo* and *billTo* is compared. It is found that *shipTo* and *billTo* have different values in some cases. The two sub-trees are shown as a shared hierarchy with a special symbol in our prototype, as shown in Figure 15. The vertex is named after the complex type *USAddress*. In our presentation of the attribute tree, arcs with the line across them represent optional arcs.

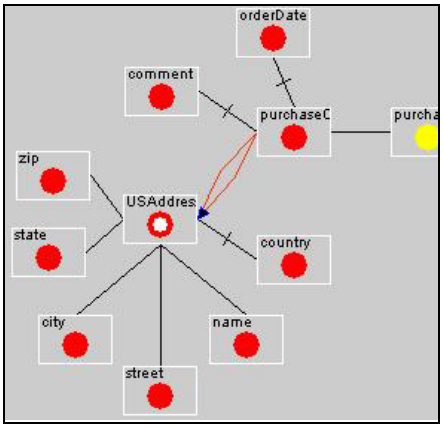

Figure 15. Prototype - shared hierarchy

Coming from the same fact in another direction, the *item* vertex is added to the attribute tree. The *partNum* vertex is a child of *item* (Figure 6) and is defined as a key reference to the *number* attribute. After the transformation presented in Figure 13, *part* and *number* are swapped. Then, during the creation of the attribute tree, *part* becomes a child of the *partNum*

attribute, since *partNum* is referencing the *number* attribute. The resulting attribute tree is presented in Figure 16. Using the relational model terminology, descendants of the primary key attribute become descendants of the foreign key (*keyref*). Without this procedure the information about part description (the *part* attribute) would be lost. This operation of replacing the foreign key attribute with the primary key attribute and its sub-tree is similar to the natural join in the relational model, and it prevents from losing the attributes that can be interesting for making useful aggregations of data.

Probably, not all of the attributes represented in the attribute tree are interesting for the web warehouse. Thus, the designer can rearrange some parts of the tree or eliminate the unnecessary details. The final steps of building a fact scheme include the definition of dimensions, measures and hierarchies as described in [5].

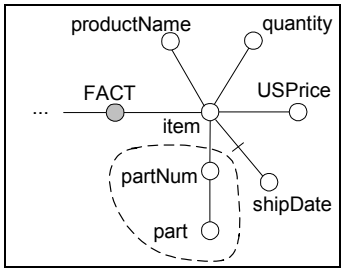

Figure 16. Replacement of keys

VI. CONCLUSIONS

In this paper we have presented a semi-automated approach for conceptual design of web warehouses from XML Schemas. After transforming the XML Schema into a Schema graph, this graph is navigated starting from a vertex/arc in order to detect the functional dependencies to be modeled within the conceptual schema for the warehouse. The algorithm proposed also takes into account the existence of attributes shared between two or more hierarchies and the presence of attributes where two or more paths of functional dependencies converge.

The algorithm has been implemented within a prototype which thus acts as a valuable support for conceptual design of web warehouses.

REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu, "Data on the Web: From Relations to Semistructured Data and XML", Morgan Kaufman Publishers, 2000.

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

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

[4] D. Florescu and D. Kossmann, "Storing and Querying XML Data using an RDBMS", *IEEE Data Engineering Bulletin* vol. 22, n. 3, 1999.

[5] 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.

[6] M. Golfarelli, D. Maio, and 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.

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

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

[9] D. Lee and W.W. Chu, "Constraints-preserving Transformation from XML Document Type Definition to Relational Schema", *Proc. 19th ER*, Salt Lake City, 2000.

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

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

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

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

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

[15] World Wide Web Consortium (W3C), "XML Schema Part 0: Primer", http://www.w3.org/TR/xmlschema-0/.

[16] World Wide Web Consortium (W3C), "Xpath Specification 1.0", http://www.w3.org/TR/xpath.

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