A Study on Data Storing Mechanism in Data Warehouse for E-Commerce Applications

Thota. Raveendranath*, Dr. Syed Umar

B.Tech. Student, ECM dept., K L University, Vaddeswaram, Guntur, AP, India
*Assoc. Professor, ECM dept., K L University, Vaddeswaram, Guntur, AP, India

Abstract: The data, which is widely used for ecommerce applications, in decision-making processes is stored in the XML format. Thus, as more organizations view the web as an integral part of their communication and business, the importance of integrating XML data in data warehousing environments is becoming increasingly high. In this paper we show how the design of a data mart can be carried out starting directly from an XML source. Two main issues arise: on the one hand, since XML models semi-structured data, not all the information needed for design can be safely derived; on the other, different approaches for representing relationships in XML DTDs and Schemas are possible, each with different expressive power. After discussing these issues, we propose a semi-automatic approach for building the conceptual schema for a data mart starting from the XML sources.

Keywords: Data warehouse design, Data warehousing and the web, XML

I. INTRODUCTION

A large amount of data needed in decision-making processes is stored in the XML (Extensible Markup Language) data format. The structure of XML, composed of nested custom-defined tags that can describe the meaning of the content itself, makes it usable as a semantic-preserving data exchange format on the web. As the Internet has evolved into a global platform for e-commerce and information exchange, the interest in XML has been growing and large volumes of XML data already exist. XML can be considered as a particular standard syntax for the exchange of semi-structured data [1]. One common feature of semi-structured data models is the lack of schema, so that the data is self-describing. However, XML documents can be associated with and validated against either a Document Type Definition (DTD) or an XML Schema, both of which allow the structure of XML documents to be described and their contents to be constrained. DTDs are defined as a part of the XML 1.0 Specification [15], while XML Schemas have recently become a W3C Recommendation [16]. XML Schemas considerably extend the capabilities of DTDs, especially from the point of view of data typing and constraining. With DTDs or Schemas, the applications exchanging data can agree about the meaning of the tags and, in that case, XML reaches its full potential. During the recent years, the enterprises have been asking for support in the process of extracting useful, concise and handy information for decision making out of the tons of data stored in their expensive and complex information systems. Consequently, a substantial effort has been made in the academic world to devise methodologies for designing data warehousing systems capable of seamlessly integrating information from different sources, in particular from heterogeneous databases. Now, as more organizations view the web as an integral part of their communication and business, the importance of integrating XML data in data warehousing environments is becoming increasingly high. Some commercial tools now support data extraction from XML sources to feed the warehouse, but both the warehouse schema and the logical mapping between the source and the target schemas must be defined by the designer. In this paper we show how multidimensional design for data warehouses can be carried out starting directly from an XML source. Two main issues arise: on the one hand, different approaches for representing relationships in XML
DTDs and Schemas are possible, each achieving a different expressive power; on the other, since XML models semi-structured data, not all the information needed for design can be safely derived. Thus, our contribution in this work is twofold: first, we propose a warehouse-oriented review and comparison of the approaches for structuring XML documents; then, we outline an algorithm in which 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. 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 can be found in the literature, both leaning on the DTD [10][13] or not [5], but insufficient emphasis is given to the problem of determining the cardinality of relationships, which instead has a primary role in multidimensional design.

II. MULTIDIMENSIONAL MODELING AND DESIGN

It is now widely recognized that an accurate conceptual design is the necessary foundation for building a data warehouse which is both well-documented and fully satisfies requirements. In order to be independent of the specific issues involved in logical and physical modeling, the approach proposed here is referred to the conceptual level, from which the logical schemas of the data marts can be easily derived.

A. Conceptual Modeling

Several conceptual models for data warehouses were devised in the literature [2][3][6][7][8][14]; they mainly differ for the graphical representation of concepts, with small differences in expressive power. In this paper we will adopt the Dimensional Fact Model (DFM) [7], which represents the data mart by means of a set of fact schemas. In the following we will briefly discuss the DFM representation of the main concepts of the multidimensional model with reference to the fact schema CLICK, which describes the analysis of the web site traffic. The reason for choosing this example is that, due to the significant role now played by the web in attracting new clients and supporting sales, analyzing the web server traffic may be crucial for improving the enterprise business. In this context, multidimensional modeling allows many unpredictable complex queries to be answered, such as: What is the trend for the most and the least accessed pages. Is there a relationship between business events (for instance, sale promotions in an e-commerce site) and the number of accesses. For such an analysis, we need information about the hostname or IP address of the computer requesting the file, the date and time of the request, and the URL of the file being requested.

In the fact schema shown in Figure 1, the fact CLICK, focus of interest for the decision-making process, is associated to the measures which describe it, i.e. no. of clicks, and to the dimensions determining its minimum granularity, namely host, date, hour, and URL. Each dimension is the root of a hierarchy which determines how the fact may be aggregated and selected significantly for the decision-making process; each hierarchy includes a set of attributes linked by functional dependencies. For instance, the URL of the file being requested determines
the file type, and the hostname or IP address of the computer requesting the file determines its domain. The values for the domain attribute can be extracted from the hostname suffix that is indicating either the category (for instance, .com for commercial companies) or the nation. Within the DFM, as in all the other conceptual models, a strong relevance is given to functional dependencies, since they represent many-to-one relationships between attributes which enable flexible aggregation of data in OLAP queries [9]. Thus, the main problem in building a conceptual data mart schema is to identify those relationships in the business domain.

B. Conceptual Design

In most approaches to design of data marts, the conceptual schema is built starting from the (logical or conceptual) schema of the operational sources [2][7][8]. The common core of these approaches consists in navigating the functional dependencies in the source schema in order to determine the hierarchies for the fact. In particular, the methodology to build a fact schema from an E/R schema proposed in [7] consists of the following steps:

1. Choosing facts.
2. For each fact:
   2.1 Building the attribute tree.
   2.2 Rearranging the attribute tree.
   2.3 Defining dimensions and measures

For briefly illustrating the methodology, we will use the E/R diagram describing the web site traffic shown in Figure 2. Facts typically correspond to events occurring dynamically in the enterprise world. On the E/R schema a fact may be represented either by an entity F or by an n-ary relationship R. In our example, the fact of primary interest is represented by entity CLICK. Given a portion of interest of a source schema and an entity F belonging to it, we call attribute tree the tree such that:

Each vertex corresponds to an attribute - simple or compound of the schema; The root corresponds to the identifier of F; For each vertex V, the corresponding attribute functionally determines all the attributes corresponding to the descendants of V.

The attribute tree for F may be constructed automatically by navigating, starting from F, the functional dependencies expressed by many-to-one relationships between entities in the source schema. Each entity E analyzed is represented in the attribute tree by:

1) a node corresponding to the identifier of E;
2) a child node for each of the non-identifier attributes of E;
3) a child node for each entity G connected to E by a many-to-one relationship.

![E/R schema for web site traffic](image-url)
The resulting attribute tree for the website traffic analysis example is shown in Figure 3. Starting from the CLICK entity, new vertices are added by following many-to-one relationships. The URL category attribute could not be included in the attribute tree because there is a many-to-many relationship between URL and URL category. The nation attribute is included twice in the attribute tree because it can be added to the attribute tree by navigating from both the host and the site entities.

![Figure 3. Attribute tree for fact CLICK](image)

Once the attribute tree has been built, some uninteresting nodes may be dropped from it. The time attribute, that represents the exact time of a click, is replaced with a coarser hour attribute. Finally, dimensions and measures must be selected among the children of the root. In our example, the attributes chosen as dimensions are host, date, hour and URL; number of clicks, determined by counting the clicks from the same host to the same URL on a given date and hour, is chosen as a measure. Some further minor arrangements must be made in order to obtain the fact schema in Figure 1; in particular, the date dimension is enriched by building a hierarchy which includes attributes month, day of week, and holiday. Besides, the host category and nation optional attributes are replaced by attribute domain, which indicates either the category or the nation of the host.

### III. Modeling Relationships in XML

An XML document consists of nested element structures, starting with a root element. Each element may contain component elements (i.e. sub-elements) and attributes. Both elements and attributes are allowed to have values. Document structures can be nested to any level of complexity; between the opening and closing tags for an element, any number of additional elements or textual data may be present. Attributes are included, with their respective values, within the elements opening declaration. An XML document that contains data about the website traffic is shown in Figure 4.

```xml
<webTraffic>
  <click>
    <host hostId=“ares.csrl.unibo.it”>
      <nation>italy</nation>
    </host>
    <date>16-JUNE-2014</date>
    <time>11:43:25</time>
    <url urlID=‘ABCD’>
      <site siteID=“www.ABCD”>
        <nation>Croatia</nation>
      </site>
      <fileType>shtml</fileType>
      <urlCategory>catalog</urlCategory>
    </url>
  </click>
</webTraffic>
```
Figure 4. An XML document describing the web site traffic

An XML document is valid if it has an associated schema, such as a DTD or an XML Schema, and if it conforms to the constraints expressed in that schema. 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 DTD and the XML Schema.

A. Relationships in DTDs

A DTD defines elements and attributes allowed in an XML document, and the nesting and occurrences of each element. The structure of an XML document is constrained using element-type and attribute-list declarations. Element-type declarations specify which sub-elements can appear as children of the element; attribute-list declarations specify the name, type, and possibly default value of each attribute associated with a given element type. Among the different attribute types, types ID, IDREF and IDREFS have particular relevance for our approach: the ID type defines a unique identifier for the element; the IDREF type means that the attributes value is some other elements identifier; IDREFS means that its value is a list of identifiers. IDREF(S) must match the value of some ID attribute in the document [1].

B. Modeling relationships by sub-elements

Relationships can be specified in DTDs by sub-elements that may have different cardinalities. The optional character following a child element name or list in the element-type declarations determines whether the element or the content particles in the list may appear one or more (+), zero or more (*), or zero or one times (?); the default cardinality is exactly one. Figure 5 presents a DTD according to which the XML document presented in Figure 4 is valid. Element webTraffic is defined as a document element, thus becomes the root of XML documents. A webTraffic element may have many click elements, while in an url element the site sub-element must occur exactly once, followed by one fileType and many urlCategory sub-elements. A host element may have either a category or a nation element.

```xml
<!DOCTYPE webTraffic [<ELEMENT webTraffic (click*)>]
<ELEMENT click (host, date, time, url)>
<ELEMENT host (category | nation)>
<ATTLIST hostHostId ID #REQUIRED>
<ELEMENT category (#PCDATA)>
<ELEMENT date (#PCDATA)>
<ELEMENT time (#PCDATA)>
<ELEMENT url (site, fileType, urlCategory+)>
<ATTLIST urlurlId ID #REQUIRED>
<ELEMENT site (nation)>
<ATTLIST sitesiteId ID #REQUIRED>
<ELEMENT nation (#PCDATA)>
<ELEMENT fileType (#PCDATA)>
<ELEMENT urlCategory (#PCDATA)>}
```

Figure 5. A DTD where relationships are specified by sub elements

If a one-to-one or one-to-many relationship must be represented in XML, sub-elements with the above mentioned cardinalities can be used without loss of information. However, given a DTD, we can follow only
one direction of a relationship. For instance, according to the DTD in Figure 5, an url element may have many urlCategory sub-elements, but it is not possible to find out, from the DTD, whether an URL category can refer to many URLs. Only by having some knowledge about the domain described by the DTD, we can conclude that the latter is the case.

C. Modeling relationships by ID and IDREF(S)

Another way to specify relationships between elements in DTDs is by means of ID and IDREF(S) attributes. The way these attributes operate resembles the key and foreign key mechanism used in relational databases, with some important differences and limitations. For instance, if we take the last part of the DTD from Figure 5 and let the relationships of the url element be defined by ID and IDREF(S) attributes rather than by sub-elements, we obtain the DTD in Figure 6.

```xml
<!ELEMENT webTraffic (click*, fileType+, urlCategory+)>  
<!ELEMENT url (site)>  
<!ATTLIST url urlId ID #REQUIRED  
fileTypeRef IDREF #REQUIRED  
urlCategoryRef IDREFS #REQUIRED>  
<!ELEMENT fileType EMPTY>  
<!ATTLIST fileType typeID ID #REQUIRED  
type Description CDATA #IMPLIED>  
<!ELEMENT urlCategory EMPTY>  
<!ATTLIST urlCategory urlCategoryId ID #REQUIRED  
urlCategoryDesc CDATA #IMPLIED>  
```

Figure 6. A DTD where relationships are specified by IDREF(S)

In this example we assume that the fileTypeRef attribute of the url element references the fileType element. On the other hand, an instance of urlCategoryRef references many instances of urlCategory. Therefore, for one URL there is exactly one file type while there may be several categories. The problem is that, using IDREF(S), the participating elements cannot be constrained to be of a certain element type. For instance, fileTypeRef could also contain a reference to an urlCategory element, while obviously we would like to constrain such references to fileType elements only. Further, though the value of an ID attribute is unique within the whole document, element types are not required to have an ID, and even if an element type has an ID, its usage may be optional. For these reasons, there is no means to actually constrain the allowed relationships by the ID/IDREF(S) mechanism.

D. Relationships in XML Schemas

XML Schemas give more accurate representation of the XML structure constraints than DTDs; in particular, the cardinality can be specified in more detail by using the minOccurs and maxOccurs attributes. 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 the sharing and reusing of simple and composite types. Further, besides ID and IDREF(S) attributes, XML Schemas support the use of key and keyRef elements for defining keys and their references. The two different ways of specifying relationships in XML Schemas, i.e. sub-elements and key/keyRef mechanisms, will be described in the following.

IV. CONCEPTUAL DESIGN FROM XML SOURCES

In the previous section, we showed different approaches for representing relationships in DTDs and XML Schemas. Three of them are suitable for specifying relationships: sub-elements in DTDs, sub-elements and key/keyRef in Schemas; though their expressive power is different, in the context of this paper they may be considered to be equivalent since with reference to multidimensional design they allow the same knowledge to
be captured. We do not consider the fourth approach, ID/IDREF(S) in DTDs, since it is not precise and useful enough in constraining relationships. In this section we propose a semi-automatic approach for building the conceptual schema of a data mart starting from the XML sources. Of the three above-mentioned approaches, we have chosen sub-elements in DTDs for the presentation of our methodology, since Schemas are still not as widely used as DTDs; however, the methodology is essentially the same when dealing with Schemas. Starting with the assumption that the XML document has a DTD and conforms to it, the methodology consists of the following steps:

1. **Simplifying the DTD.**
2. **Creating a DTD graph.**
3. **Choosing facts.**
4. **For each fact:**
   1. **Building the attribute tree from the DTD graph.**
   2. **Rearranging the attribute tree.**
   3. **Defining dimensions and measures.**

In the following paragraphs we will describe steps from (1) to (4.1) referring to the web site traffic example; once the attribute tree is built, steps 4.2 and 4.3 are identical, respectively, to steps 2.2 and 2.3 described in Section 2.3. **Simplifying the DTD**

The sub-elements in DTDs may have been declared in a complicated and redundant way. However, those details of a DTD can be simplified [13]. The transformations for simplifying a DTD include converting a nested definition into a flat representation: for instance, in the web site traffic example, `host(category?/nation)` is transformed into `host(category?, nation?)`. Further, the sub-elements having the same name are grouped, and many unary operators are reduced to a single unary operator. Finally, all `+` operators are transformed into `*` operators. **Creating a DTD graph** After simplifying the DTD, a DTD graph representing its structure can be created as described in [10] and [13]; its vertices correspond to elements, attributes and operators in the DTD. Attributes and sub-elements are not distinguished in the graph since, in our methodology, they are considered as equivalent nesting mechanisms. The DTD graph for the DTD in Figure 5 is given in Figure 10.

**Defining facts**

The designer chooses one or more vertices of the DTD graph as facts; each of them becomes the root of a fact schema. In our example, we choose the `click` vertex as the only interesting fact.

**Building the attribute tree**

The vertices of the attribute tree are a subset of the element and attribute vertices of the DTD graph. The algorithm to build the attribute tree is sketched in Figure 11.
root=newVertex(F);
// newVertex(<vertex>) returns a new vertex
// of the attribute tree
// corresponding to <vertex>
// of the DTD graph
expand(F,root);
expand(E,V):
// E is the current DTD vertex,
// V is the current attribute tree vertex
{ for each child W of E do
  if W is element or attribute do
    { next=newVertex(W);
      addChild(V,next);
      // adds child W to V
      expand(W,next);
    }
  else
    if W="?" do
      expand(W,V);
    for each parent Z of E such that
    Z is not a document element do
    if Z="?" or Z="*" do
      expand(Z,V);
    else
      if not toMany(E,Z) do
        if askDesignerToOne(E,Z) do
          { next=newVertex(Z);
            addChild(V,next);
            expand(Z,next);
          }
      }
  }
}

Figure 11. Algorithm for building the attribute tree

Figure 12. Attribute tree derived from the DTD graph

In our example, no uncertain relationships are navigated. Vertex urlCategory is not added to the attribute tree because it is a child of a "*" vertex in the DTD graph. The resulting attribute tree for the web site traffic analysis example is given in Figure 12. Some further arrangements should be made to this tree: for instance, since there
is no need for the existence of both host and hosted vertices, only host should be left; the same logic should be applied for purled and siteId. Finally, the time attribute is replaced with the hour attribute. In the following some general considerations on the proposed approach are reported.

The problem of checking cardinalities in XML documents is related to that of discovering functional dependencies in relational databases, which was widely investigated in the literature on relational theory and data mining [11][12]. In our case, the situation is much simpler since no inference is necessary, so it comes down to properly querying the data with an XML query language supporting aggregate queries. For instance, in W3C XQuery [18] the use of the distinct function is proposed for that purpose, while the use of the group-by function is proposed in [4]. The main question arising is how many XML documents we must see to reasonably confirm our presumption that the cardinality is 1-to-one. Clearly, the semi-structured nature of XML sources increases the level of uncertainty on the structure of data as compared to E/R sources, thus making recourses to the designers experience more frequent. In our algorithm, we chose to ask questions interactively during the tree building phase in order to avoid unnecessary queries on the documents. An alternative solution consists in first building the tree by emphasizing all the uncertain relationships, then handing on the complete tree to the designer in order to have it rearranged (step 4.2) with specific attention to uncertain relationships, which could be dropped together with their sub tree. While this solution allows the designer to have a wider point of view on the tree, it may be less efficient since a vertex, dropped later, might have been (uselessly) expanded by querying the XML documents.

On the other hand, in the relational case this issue is much less relevant than in the XML case. In fact, while the designer of an XML document chooses the direction of each link without considering the cardinality of the relationship to be modeled, the designer of a relational schema is constrained (by the need to satisfy the first normal form) to represent each relationship in the 1-to-one direction. Thus, in general, the relationship from S to R is one-to-many, hence, not interesting for multidimensional modeling; the only case in which it might be interesting is when the foreign key mechanism has been used by the designer to model a one-to-one relationship D but this is not very frequent. As seen in Section 3, several DTDs representing the same subject may be designed; for each of them, the resulting attribute tree may look different. For instance, the attribute tree for the DTD graph in Figure 13 is presented in Figure 14. Having click as a fact, navigating from hostId to host and from urlId to url entails analyzing the data to check the uncertain relationship. After inverting hosted with host and urlId with url (it can be done since they are related by a one-to-one relationship), the resulting attribute tree becomes the same as the one in Figure 12.

![Figure 13. Another possible DTD graph describing the web site traffic](image-url)
V. CONCLUSIONS

In this paper we described a semi-automatic approach to conceptual design of a data mart from an XML source. We showed how the semi-structured nature of the source increases the level of uncertainty on the structure of data as compared to structured sources such as database schemas, thus requiring access to the source documents and, possibly, the designers help in order to detect -to-one relationships. The approach was described with reference to the case in which the sources are constrained by a DTD using sub-elements, but it can be adopted equivalently when XML Schemas are considered. Using XML sources for feeding data warehouse systems will become a standard in the next few years. Adopting a technique to derive the data mart schema directly from the XML sources is not the only possible approach: the data mart schema may also be designed “manually”, meaning that facts, measures and hierarchies are determined starting from the user requirements and the logical connection with the source schemas is established only a posteriori. On the other hand, the main problem with this solution is that, very often, the requirements expressed by the users cannot be fully supported by the existing data; besides, the process of mapping each requirement back to the source schema may be very complex.

REFERENCES


