INTEGRATING XML SOURCES INTO A DATA WAREHOUSE ENVIRONMENT

Matteo Golfarelli, Stefano Rizzi
University of Bologna, DEIS
Viale Risorgimento 2, 40136 Bologna, Italy
E-mail: mgolfarelli@deis.unibo.it, srizzi@deis.unibo.it

Boris Vrdoljak
University of Zagreb, FER
Unska 3, 10000 Zagreb, Croatia
E-mail: boris.vrdoljak@fer.hr

Abstract: *A data warehousing system is a collection of technologies and tools which enables knowledge workers to acquire, integrate and flexibly analyze information from different sources aimed at improving the knowledge assets of the enterprise. The importance of integrating XML data in data warehousing environments is becoming increasingly high as more organizations view the web as an integral part of their communication and business. In this paper we propose a semi-automatic approach for building the conceptual schema for a data mart starting from the DTDs describing the XML sources. The main issue arising is that, since XML models semi-structured data, not all the information needed for design can be safely derived. In our approach, this issue is addressed by querying the source XML documents and, if necessary, by asking the designer's help.*

KEYWORDS: data warehousing and the web, data warehouse design, XML

INTRODUCTION

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. A *data warehousing* system is a collection of technologies and tools which addresses this issue by enabling knowledge workers (executives, managers, analysts) to acquire, integrate and flexibly analyze information from different sources aimed at improving the knowledge assets of the enterprise.

The core of this architecture is a *data warehouse* (DW), i.e. a data repository oriented to subjects, integrated and consistent, regularly refreshed to represent temporal evolution. Though the DW is logically centralized, it often consists of different *data marts* oriented to specific areas of the enterprise. From the designer's point of view, data marts are typically used as building blocks when creating the warehouse. At the conceptual level, each data mart is organized according to the multidimensional model and accessed by OLAP (On-Line Analytical Processing) queries [4].

Most approaches to data mart design devised in the literature are based on the schemas of the operational sources (e.g., [3][6]). Now, as more organizations view the web as an integral part of their communication and business, and since a large amount of data needed in decision-making processes is stored in the XML (Extensible Markup Language) data format, the importance of integrating XML data in data warehousing environments is becoming increasingly high.

XML is used 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 [8], while XML Schemas have recently become a W3C Recommendation [9]. 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.

In this paper we show how multidimensional design for data warehouses can be carried out starting directly from an XML source. Different approaches for representing relationships in XML DTDs are possible, each achieving a different expressive power; on the other hand, 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 comparison of the approaches for structuring XML documents by DTDs and Schemas; 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.

The paper is structured as follows. In Section 1 the basics of multidimensional modeling are given, while in Section 2 the design alternatives for modeling relationships in DTDs and XML Schemas are discussed. In Section 3 our approach to multidimensional design is presented with reference to a case study, and in Section 4 the conclusions are drawn.

1. MULTIDIMENSIONAL MODELING

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.

Several conceptual models for data warehouses were devised in the literature [6]; 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) [3], 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?

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*. Facts typically correspond to events occurring dynamically in the enterprise world. 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.

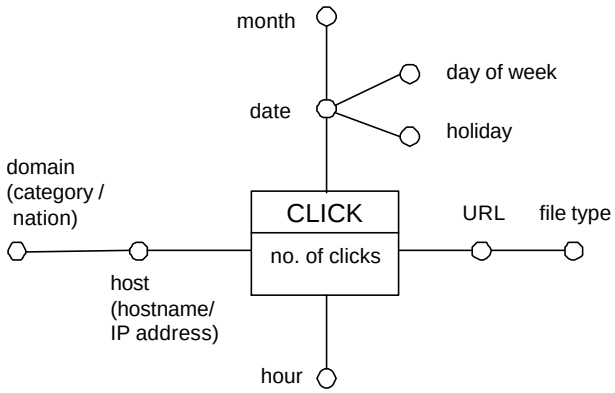

Figure 1. Fact schema for click-stream analysis

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 [4]. Thus, the main problem in building a conceptual data mart schema is to identify those relationships in the business domain.

2. REPRESENTING 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. 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 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 attribute's value is some other element's 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].

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.

An XML document that contains data about the web site traffic is shown in Figure 2, and a DTD where relationships are specified by sub-elements is included in the document. 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.

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 2, 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.

```
<!DOCTYPE webTraffic [
  <!ELEMENT webTraffic (click*)>
  <!ELEMENT click (host, date, time, url)>
  <!ELEMENT host (category | nation)>
  <!ATTLIST host
     hostId ID #REQUIRED>
  <!ELEMENT category (#PCDATA)>
  <!ELEMENT date (#PCDATA)>
  <!ELEMENT time (#PCDATA)>
  <!ELEMENT url (site, fileType, urlCategory+)>
  <!ATTLIST url
     urlId ID #REQUIRED>
  <!ELEMENT site (nation)>
  <!ATTLIST site
     siteId ID #REQUIRED>
  <!ELEMENT nation (#PCDATA)>
  <!ELEMENT fileType (#PCDATA)>
  <!ELEMENT urlCategory (#PCDATA)>
]>
<webTraffic>
  <click>
     <host hostId=”ares.csr.unibo.it”>
       <nation>italy</nation>
     </host>
     <date>23-MAY-2001</date>
            …
</webTraffic>
```

Figure 2. An XML document with a DTD where relationships are specified by sub-elements

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. Using IDREF(S), the participating elements cannot be constrained to be of a certain element type. 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.

XML Schemas give more accurate representation of the XML structure constraints than DTDs; in particular, the cardinality can be specified in more detail. Further, XML Schemas introduce more powerful and flexible mechanisms for modeling inter-concept relationships, similar to the relational

concept of foreign key. We will not discuss further Schemas since, though their expressive power is different, with reference to multidimensional design they allow the same knowledge to be captured as sub-elements in DTDs.

3. CONCEPTUAL DESIGN FROM XML SOURCES

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 XML approaches for representing relationships, 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. We do not consider the ID/IDREF(S) approach in DTDs, since it is not precise and useful enough in constraining relationships.

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:
   4.1 Building the attribute tree from the DTD graph.
   4.2 Rearranging the attribute tree.
   4.3 Defining dimensions and measures.

In the following paragraphs we will describe the steps referring to the web site traffic example.

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 [7]. 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.

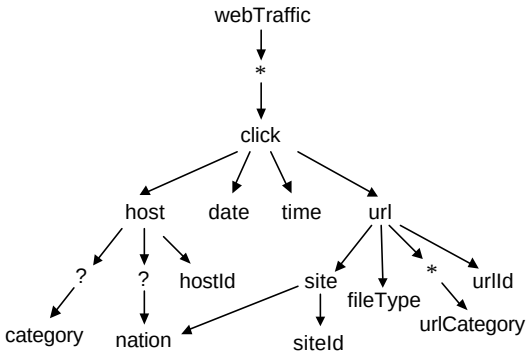

Figure 3. DTD graph for web site traffic analysis

Creating a DTD graph

After simplifying the DTD, a DTD graph representing its structure can be created as described in [5] and [7]; 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 2 is given in Figure 3.

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 4.

```
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 4. Algorithm for building the attribute tree

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 DTD graph. Each vertex V inserted in the attribute tree is expanded as follows (procedure `expand`):

1. *For each vertex W that is a child of V in the DTD graph*:
   When examining relationships in the same direction as in the DTD 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 DTD, 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 "*" operator, no vertex is added.

2. *For each vertex Z that is a parent of V in the DTD graph*:
   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 DTD yields no further information about the relationship cardinality, it is necessary to examine the actual data by querying the XML documents conforming to the DTD. This is done by procedure `checkToMany`, which counts the number of distinct values of Z corresponding to each value of E. 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 E 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 (procedure `askDesignerToOne`). Only in the first case, Z is added to the attribute tree. The reason why document elements are not considered is that they have only one instance within XML documents, thus they have no interest for aggregation and should not be modeled in the data mart.

As to the problem of checking cardinalities in XML documents, XML query languages supporting aggregate queries can be used. For instance, in W3C XQuery [10] the use of the *distinct* function is proposed for that purpose, while the use of the *group-by* function is proposed in [2]. The main question arising is how many XML documents we must see to reasonably confirm our presumption that the cardinality is -to-one.

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 5.

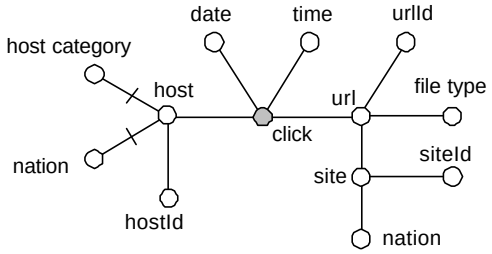

Figure 5. Attribute tree derived from the DTD graph

Rearranging the attribute tree

Some further arrangements should be made to this tree: for instance, since there is no need for the existence of both *host* and *hostId* vertices, only *host* should be left; the same logic should be applied for *urlId* and *siteId*. Finally, the *time* attribute is replaced with the *hour* attribute.

Defining dimensions and measures.

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.

4. 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 designer's 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.

We believe that using XML sources for feeding data warehouse systems will become a standard in the next few years. Thus, designing the data warehouse directly from the XML sources may reduce the risk of losing relevant information during the translation from XML to relational.

REFERENCES

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

[2] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu. A Query Language for XML. Proc. 8th World Wide Web Conference (1999).

[3] M. Golfarelli, D. Maio, S. Rizzi. The Dimensional Fact Model: a conceptual model for data warehouses. Int. Jour. of Cooperative Inf. Systems 7, 2&3 (1998).

[4] R. Kimball. The data warehouse toolkit. John Wiley & Sons (1996).

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

[6] M. Blaschka, C. Sapia, G. Höfling, B. Dinter. Finding Your Way through Multidimensional Data Models. DEXA Workshop (1998).

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

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

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

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