XML Source Preparation for Building Data Warehouses

Yasser Hachaichi, Jamel Feki, Hanene Ben-Abdallah
Laboratoire MIRACL, Faculté des Sciences Economiques et de Gestion de Sfax,
Route de l’Aérodrome km 4, B.P. 1088 - 3018 Sfax, Tunisie
Yasserhfr@yahoo.fr, {Jamel.Feki, Hanene.BenAbdallah}@fsegs.rnu.tn

Abstract

Faced with the high economic competition, today’s enterprises are forced to rely on decision support systems to assist them in the analysis of large data volumes. Traditionally, the analyzed data are mainly issued from the enterprise’s operational information system. However, due to the international nature of the competition, enterprises are increasingly pressed to explore other, external data sources, mainly issued from the web. However, despite the convergence of XML as a standard data format on the web, a main difficulty in exploiting these data is the lack of data warehouse design approaches for XML data sources. The few proposed approaches suppose that the designer must first manually identify the XML element that represents the analyzed fact; such a step requires a high expertise and domain knowledge. The main contribution of this paper is to automate this step.

1. Introduction

Faced with the high economic competition, today’s enterprises are constrained to rely on decision support systems to assist them in the analysis of large data volumes. Data warehousing technologies intervened to allow managers both to acquire and integrate information from different sources, and to query/analyze efficiently very large databases. In deed, these technologies encompass tools to regroup large data volumes into a single repository, called a data warehouse (DW), suitable for direct querying and analysis. In addition, a DW can be used as a source for building data marts (DM) that are oriented to specific topics.

Traditionally, the data loaded into a DW/DM is mainly issued from the enterprise’s operational information system. However, due to the international nature of the competition, enterprises are increasingly forced to explore other, external data sources mainly issued from the Web. In this information universe, despite the convergence of XML as a standard data format, a main difficulty in exploiting the data from the web remains the lack of data warehouse design approaches for XML data sources.

Over all, the few proposed approaches for the design of XML-based DW/DM (cf. [1] [2] [3] [4] [5]) suppose that the designer must first manually identify the XML element that represents an analyzed fact. However, this identification requires a high expertise in both the OLAP domain and the XML document domain.

The main contribution of this paper is to prepare XML data sources to automate the fact identification step in the design of XML-based DM/DW. More specifically, this paper presents a generic set of rules that identify the XML element representing potential facts and dimensions. Our rules are generic in the sense that they rely on the structure of the XML sources without any domain knowledge. They are inspired from our approach for multidimensional schema construction from relational data sources [6]; this approach is also independent of the data source semantics.

We note that, once an XML element is identified as a fact, any approach for determining the remaining multidimensional schema components can be applied, cf. [1] and [2].

The remainder of this paper is organized as follows. Section 2 presents pertinent work on DW/DM schema construction for XML data sources. Section 3, first, presents an interpretation of the XML elements from a multidimensional perspective; secondly, it describes an example used throughout the paper. Section 4 details our data preparation for fact and dimension identification. Finally, Section 5 summarizes the presented work and outlines our ongoing efforts.

2. Related work

Several researchers have studied how to construct a multidimensional schema either from a data source model (e.g., E/R diagrams [7] [8] [9] and relational databases [6]), or from real world entities (i.e., any information artifact) [10], and XML documents [1],[2],[3],[4],[5]. We next overview only those proposals that dealt with XML sources.

Pokorny [11] and Hümmer [12] propose to represent a multidimensional model as an XML document. However, they do not address how to build the multidimensional schema represented by the proposed XML model. On the other hand, Golfarelli et al., propose in [1] a method for the design of an XML data warehouse from XML sources. Their method relies on two assumptions: the
existence of a document type definition (DTD) for the XML sources, and the conformity of these sources to their corresponding DTD. This method consists of three steps: DTD simplification mainly to flatten nested elements; DTD graph creation in order to represent graphically the source structure; and for each fact manually chosen in the graph, an attribute tree is built where the dimensions and measures are found among the nodes immediately linked to the chosen fact. In this method, the selection of facts and measures is manual and requires the intervention of an expert in the domain of the XML documents that will load the future data warehouse.

In an attempt to improve their previous method, Vrdoljak et al., [2] developed a semi-automated process to design XML data warehouses from XML schemas. Here, again, in this process, facts and measures are chosen manually. For each chosen fact, this design process is performed according to the following three steps: building the dependency graph from the XML schema graph; rearranging the dependency graph to define dimensions and measures; and creating a logical schema. One major drawback of this method is that it requires an intensive intervention of the designer. In addition to manually identifying the fact, dimensions and measures, the designer must also identify the many-to-many relationships among elements; these relationships are needed to construct the dependency graph and are explicit in the XML documents.

Furthermore, the authors of [3] and [13] propose a generic method for building an XML data warehouse for XML documents. This method first applies a set of cleaning and integration operations in order to minimize the number of occurrences of dirty data, XML structural errors, duplications or inconsistencies. Secondly, it summarizes data to extract only useful and valuable information in order to create another XML document(s) used for the construction of the dimensions. Thirdly, the method creates intermediate XML documents from the initial documents; this step focuses on determining the main activity data (data involved in queries, calculations etc.). Thus, each intermediate document linked to other documents represents a fact document. Finally, the method updates/links all intermediate XML documents (fact and dimensions documents), in such a way that relationships between keys are established, and an XML data warehouse is created. In this method, several sub-steps have to be accomplished manually by an expert in the XML documents domain.

On the other hand, Jensen et al. [4] studied how an OLAP cube can be obtained from XML data. To build a cube, the DTD of the XML documents is transformed into an UML (Unified Modeling Language) class diagram using a set of transformation rules; these rules are informally described in [14]. After that, the designer uses the obtained class diagram to specify an OLAP DB model (named a UML snowflake diagram) using a graphical user interface. Finally, the UML snowflake diagram is transformed into relational structures to prepare the implementation of the OLAP system. This approach is also used by Ouaret et al. in [5] who starts from XML schemas instead of DTD.

Overall, the proposed methods for the design of XML data warehouse require assistance in the identification of XML elements representing facts and/or dimensions. Such assistance should be automatic and independent of the XML source semantics. This is the main contribution of the paper.

3. Interpreting XML structures

An XML document represents two types of information: the data structure and content. XML provides a means for separating one from the other in the electronic document. The structure of a document is given by opening and closing, matching tag pairs (each called an element) and the content is given by the information between matching tags. In addition, an element can have attributes whose values are assigned in the opening tag of the element.

To define the structure of a set of XML documents, a DTD document can be used. A DTD is a context free grammar defining, in terms of element content specifications, all allowable elements, their attributes, and the element nesting structure. Given one DTD, it can be verified whether an XML document conforms to/respects the DTD, and if so, the XML document is said to be valid.

Note that there exist other formalisms for describing XML document structure, e.g., XML Schema. However, the DTD is a formalism recommended by the World Wide Web Consortium (W3C) [15] [16]. For this, we assume here that the structure of an XML document is described by a DTD and that the XML source documents are valid.

A DTD is composed of element types, sub-element types, attributes, and terminal strings such as ENTITY, PCDATA and CDATA; the DTD types are however very limited since all of the types are considered as strings. Furthermore, a DTD can constrain the occurrences of an element and a sub-element type through the symbols: “*” (set with zero or more elements), “+” (set with one or more elements), “?” (optional elements), and “|” (alternative elements). More details about DTD can be found in [15] [16].

Figure 1 depicts an example of DTD describing the structure of Sales documents. A sales document describes the list of items sold by a named retailer, at a given date, and for a given customer.

An example of an XML document, valid according to this DTD, is shown in Figure 2. This sample document
partially describes a sale performed on the 1st of January 2008, for the customer with id 00001-08 and name “extraterrestrial”, and whose address is in “New Buildings Jupiter planet El2 JPP”. This customer bought “one part of chip of a shuttle” with id “CS276512” for the price of “$6.25”.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE sales SYSTEM "sales.dtd">
<sales>
<date>2008-01-01</date>
<customer customerID="00001-08" name="extraterrestrial">
<address>
<locality>New Buildings. Jupiter planet</locality>
<ZIP>El2 JPP</ZIP>
</address>
</customer>
<item>
<itemID>cs276512</itemID>
<price>6.25</price>
<quantities>1</quantities>
<description>chip of a shuttle</description>
</item>
...
</sales>
```

**Figure 1.** An example of DTD for Sales documents.

Our DM construction approach starts from a DTD to build a multidimensional schema where facts, dimensions and dimensional attributes are extracted, mainly, from non-PCDATA elements. The intuition behind our approach is that: a fact represents a collection of data; hence, an element representing a fact should have a complex structure. In addition, a dimension supplies a fact with the data; hence, a dimension should also come from an element with a complex structure and nested in another (fact) element.

In our running example, the *sales* element is a potential fact with the *customer* element being one of its dimensions. The next section prepares for fact and dimension identification from a DTD document.

### 4. Data Preparation for fact and dimension identification

As mentioned in the introduction, our goal, in this paper, is to complement the proposed approaches for DM schema construction from XML sources by automatically identifying the XML elements representing potential facts and dimensions. In our approach, similar to Golfarelli et al. [1], we suppose that the XML sources are valid with respect to an existing DTD.

Figure 3 illustrates the three, automatic steps of our approach. The DTD is first simplified to reduce/eliminate redundancies. Secondly, the simplified DTD is split into substructures reorganized as trees that we call transition trees; each root node of a transition tree represents either a potential fact or dimension. Finally, in addition to the DTD, our approach explores the XML documents themselves in order to extract typing information that is absent in the DTD. The typing information is used to assist in identifying measures of the facts.

```xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE sales SYSTEM "sales.dtd">
<sales>
<date>2008-01-01</date>
<customer customerID="00001-08" name="extraterrestrial">
<address>
<locality>New Buildings. Jupiter planet</locality>
</address>
</customer>
<item>
<itemID>cs276512</itemID>
<price>6.25</price>
<quantities>1</quantities>
<description>chip of a shuttle</description>
</item>
...
</sales>
```

**Figure 2.** A sample XML Sales document.

#### 4.1. DTD simplification

The simplification of a DTD removes empty elements, substitutes and transforms other elements. Empty element removal is applied to every element that is tagged EMPTY and that does not declare an ATTLIST. Such an element has no content in the valid XML documents. For example, the DTD declaration:

```xml
<!ELEMENT test EMPTY>
```

can be encountered in XML documents in one of two void forms:

```xml
<test></test> or <test/>
```

These empty elements are neither useful for the decision process, nor for the DM/DW construction.

The substitution step first replaces each reference to an ENTITY type with the text corresponding to that entity, and then it removes the ENTITY declaration. Note that an ENTITY is a variable used to define shortcuts for either commonly used text, or text that is difficult to type.

Finally, the third simplification step applies a set of transformations to the DTD to reduce the components of its ELEMENT declarations. Our transformation rules slightly differ from those presented in [17][18] and resemble the transformations in [14]. They are of the three types depicted in Figure 4: (a) flattening which converts a nested definition into a flat representation where the binary operators “,” and “|” do not appear inside any parentheses; (b) reduction which reduces several consecutive unary operators to a single unary
4.2. Transition tree construction

The DTD simplification step is followed by the construction of transition trees for the simplified DTD. Each transition tree describes an entity in the domain of the XML documents. Thus, the transition trees will facilitate fact and dimension identification since, in DW design approaches, facts and dimensions are constructed on entities [7] [9].

Each transition tree has a root node, intermediate nodes, terminal nodes (leaves) all of which are connected by arcs. The root node and intermediate nodes refer to one element in the DTD. On the other hand, a leaf denotes either a PCDATA element, an attribute, or an element identified as the root node of a transition tree (the same tree if the DTD contains a recursion). In addition, the arcs of the transition tree are labeled with either a cardinality symbol (?, * or +), or an attribute type (ID, IDREF, I: IMPLIED, R: REQUIRED, F: FIXED).
4.2.1. Root determination. In this section, we define four rules that determine the root node (element) of a transition tree.

**RD1:** Each element that does not appear in the declaration of any other element is a root of a transition tree.

In general, each XML document can be seen as one root element that contains all the elements in the document. Thus, the rule RD1 will extract this top-most element as a root of one transition tree. Informally, a root of a transition tree is an element that is not contained in any other element. However, for some XML documents, their DTD can be recursive; that is, all elements in the DTD are sub-elements of other elements [18]. We treat the case of a recursive DTD in the RD4 rule.

**RD2:** Each element that contains at least one non-PCDATA element is a root for a transition tree.

Rule RD2 excludes transition trees composed of the root connected to only leaf nodes (PCDATA). Such a tree, in the XML document, can be considered as a basic data entity, in the sense that it cannot represent a fact to be analyzed. In addition, by imposing that a transition tree contains at least one complex element, rule RD2 ensures that the transition tree represents a candidate fact. In this tree, the complex element will be identified as the root of a transition tree, referenced by the transition tree identified by RD2. The transition tree of the complex element will therefore correspond to a candidate dimension of the fact transition tree identified by RD2.

**RD3:** Each element contained in the declaration of $n$ elements ($n \geq 2$) is the root of a transition tree.

This rule avoids the redundancy of the elements in a transition tree and identifies the elements shared by two or several trees. This identification is useful for the construction of constellation schemas.

**RD4:** Each element directly or transitively containing its own declaration is the root of a transition tree.

This rule treats the case of recursive DTD. Informally, if an element refers to itself in its declaration (directly or indirectly), then this element contains a non-PCDATA element, and thus, in accordance with rule RD2, this element should be the root of a transition tree.

For our running example, we find two roots: the element sales (via RD1) and the element customer (via RD2).

4.2.2. Transition tree construction. To construct the transition trees, we start from the identified roots and scan the simplified DTD. Let $R$ be a root; the transition tree stemming from $R$ is constructed automatically by applying the algorithm Create_tree ($R, DTD$), defined as follows:

Create_tree ($E, DTD$) // $E$ is the current node in DTD

```plaintext
for each element $e$ in the declaration of $E$
do {
    addChildNode($E, e$) //add a child $e$ to the node $E$
    markArcCard($E, e$) // mark the arc from $E$ to $e$ by the cardinality of $e$
    if ($e$ is determined as a root)
        endif identified by RD1 to RD4
    then annotateNode($e, #$)
    else if ($e$ contains other elements or attributes)
        then CreateTree($e, DTD$)
    }
for each attribute $a$ in the declaration of $E$
do {
    addChildNode($E, a$)
    markArcType($E, a$) //mark arc from $E$ to $a$
    by the type of $a$
}
```

In the above transition tree construction algorithm:

- The function markArcCard($E, e$) annotates the arc from $E$ to $e$ with the number of occurrences of $e$ in $E$. The annotation symbol can be either $?, *, +,$ or the default value of $1$. It is helpful in identifying the kind of relationship between $E$ and $e$: the x-to-one ($? \text{ and } 1$) and x-to-many ($+ \text { and } *$) relationships will help us to identify the dimension and hierarchy parameters [7][8]; in addition, a minimal occurrence ($? \text { and } 1$) will help us to classify identified concepts according to their analytical potential. For more details about the classification process, the reader is referred to [6].

- The function annotateNode($e, #$) marks any sub-element that is the root of a transition tree with the symbol $#$. This annotation is borrowed from the concept of foreign keys in relational databases; it is useful to link transition trees of the same DTD and to construct parameter hierarchies. Note that with this annotation, the constructed transition trees will have at most a depth of four. Such a limited depth accelerates the traversal during the DW/DM schema construction.

- The function markArcType($E, a$) annotates the arc from $E$ to $a$ with the type of the attribute $a$; the attribute types are found inside any attribute declaration. The annotation symbol can be: $I$ for an optional attribute (defined by #IMPLIED in the DTD), $R$ for a mandatory attribute (#REQUIRED), $F$ for a constant valued attribute (#FIXED), $ID$ for a unique identifier, and $IDREF$ for an attribute that points to an ID attribute. These attribute types will enable us to reduce candidate measures: Optional, fixed and identifier attributes cannot be significant measures because they are artificial and redundant information that does not trace an activity of the enterprise.
4.3. Transition tree enrichment

In addition to facts and dimensions, other multidimensional concepts (i.e., measures, non dimensional attributes and some dimensions) can be further identified among the attributes and PCDATA elements. This identification relies on precise data types (e.g., number, date...). However, such typing information is totally absent in the DTD and XML documents. In fact, a DTD schema declares all data as string.

To assign a data type to the attributes and PCDATA elements in a transition tree, we query a representative set of XML documents valid with respect to the source DTD. For each leaf not marked with # (i.e., each attribute and PCDATA element), we consult the data contained in its corresponding XML tag. Then, we determine a type by scanning the text value and cast it into one of three appropriate types: date, number or string. To assist us with this analysis of the XML documents, there are several semi-structured query languages for XML documents, including XML-QL, Lorel, UnQL, XQL (from Microsoft), XQuery (from w3c). All these languages have the notion of path expressions to navigate in the nested structure of XML documents. In our example, we used the XQuery language.

The application of the enrichment step on the transition trees of our running example (Figure 6) produced the transition trees depicted in Figure 7. The data types are added to leaf nodes not annotated with the symbol #.

Once enriched, the transition trees can be scanned first to identify automatically facts and dimensions. The fact is any root containing a numeric leaf that is not annotated with # and that is connected via an arc either not annotated or annotated with R. In our running example, the root Sales is identified as a fact: The leaf Price satisfies the above three conditions. Once the fact roots are identified, the remaining roots are dimensions.

Finally, within the transition trees rooted by a fact, we can find the measures by applying the rules proposed in [1]. Further more, within the trees (and sub-trees) rooted by a dimension node, we can determine the hierarchies by adapting the rules proposed in [1] and [6].

For our running example, we obtained the star schema shown in Figure 8 from the Sales transition tree shown in Figure 7.

5. Conclusion

For data internal to an enterprise, several DW design approaches have been proposed for data sources organized often as relational data models, cf. [6]. On the other hand, when external data sources are used, the relational models are replaced by structured models defined through DTD. This data model replacement motivated some proposals for design approaches of DW schemas from XML documents. However, all of the proposed approaches suppose that the designer can identify the XML element corresponding to facts and dimensions. That is, they have the necessary expertise in the data source and multidimensional modeling. However, such an expertise is often not evident to acquire.

The work presented in this paper aims at complementing the proposed design approaches of XML-based DW by automating the fact and dimension
identification step. More specifically, it proposes a three-step identification approach: The first step simplifies the DTD of the sources by eliminating redundant information. The second step applies a set of rules to reorganize the simplified DTD into sub-trees; the root of each sub-tree is can be either a fact or a dimension. Experimental evaluation of our identification approach on examples from the literature identifies all the manually picked fact elements.

A second contribution of the paper is the proposal of an enrichment step that annotates the leaves of each sub-tree with their types extracted from XML documents conforming to the source DTD. The annotation with types allows the identification of the date dimensions and the dimension hierarchies. Hence, the final output of the three steps can be used by a design approach to construct the DM schema of the source DTD.

The presented work is currently being integrated into a complete automated DW schema design approach. In this approach, we will consider multiple and heterogeneous data sources (XML documents and relational databases). For this, we will define integration rules to apply either at the source level, or at the DM schema level.

6. References


