X-Warehousing: an XML-Based Approach for Warehousing Complex Data

Omar Boussaid[♭], Riadh Ben Messaoud[‡], Rémy Choquet[†], Stéphane Anthoard[†]
[♭]omar.boussaid@univ-lyon2.fr, [‡]rbenmessaoud@eric.univ-lyon2.fr,
[†]{remy.choquet|stephanea}@gmail.com

Laboratory ERIC - University of Lyon 2
5 avenue Pierre Mendès-France
69676, Bron Cedex – France
http://eric.univ-lyon2.fr

Abstract. XML is suitable for structuring complex data coming from different sources and supported by heterogeneous formats. It allows a flexible formalism capable to represent and store all types of data. Therefore, the importance of integrating XML documents in data warehouses is becoming increasingly high. We propose an XML-based methodology, called *X-Warehousing*, which designs warehouses at a logical level, and feeds them with XML documents at a physical level. Our approach is mainly oriented to users analysis objectives expressed according to an *XML Schema* and merged with XML data sources. The resulted *XML Schema* represents the logical model of a data warehouse. Whereas, XML documents validated against the analysis objectives define the physical model of the data warehouse, called the *XML cube*.

1 Introduction

With the recent popularity of Internet and new ways of communication, enterprises are collecting huge amount of heterogeneous data. These data are quite complex since they concern different types of information, coming from different sources, and presented on different supports. For instance, in medicine, a case study of a patient may contain general information about the patient (age, sexe, etc.) as well as scanned images, recorded interviews and expert's annotations. Since enterprises aim at integrating these data in their Decision Support Systems (DSS), some efforts are needed to structure them and to make them homogeneous as well as possible. The XML (eXtensible Markup Language) formalism has emerged as a dominant W3C[1] standard in describing and exchanging data among heterogeneous data sources in a semi-structured way. Its self-describing hierarchical structure enables a manipulative power to accommodate complex, disconnected, and heterogeneous data. Further, XML documents may be validated against an *XML Schema*. It allows to describe the structure of a document and to constraint its contents. Nowadays, in most organizations, XML

[1] http://www.w3.org

documents are becoming a usual way to represent and to store data. Therefore, new efforts are needed to integrate XML in classical business applications. Feeding data warehouses with XML documents is also becoming a challenging issue since we know that multidimensional organization [1] of data is quite different from semi-structured organization. The difficulty consists in carrying out a multidimensional design within a semi-structured formalism like XML.

In this paper, we propose an XML based approach, called *X-Warehousing*, to warehouse complex data. We include a methodology that enables the use of XML as a logical modelling formalism of data warehouses. This methodology starts from analysis objectives defined by users according to a multidimensional conceptual model (MCM). We use MCM in order to easily represent multidimensional structures of a data warehouse through what users can express future analysis objectives at a conceptual level. The data warehouse is then modelled at a logical level with an *XML Schemas*, which defines a *reference data cube model*. Our approach is also able to feed the designed data warehouse with XML documents that reflect the latter analysis needs over complex data. In fact, the reference data cube model is matched with complex data presented under XML documents. Note that, we focus on analysis needs rather then data themselves. In order to match the reference model with XML documents, they are both presented by *XML Schemas*. Then, we transform them into *attribute trees* [2] to make them comparable. Therefore, these *attribute trees* will be merged according to a fusion function by *pruning* and *grafting* [3]. Finally, our approach outputs XML documents valid as well as possible against the reference cube model. Each output XML document respects the user constraints required on its data content and represents a real OLAP (On-Line Analytical Processing) fact. The whole set of warehoused documents correspond to the physical model of the data warehouse denoted *XML Cube*.

The rest of the paper is organized as follows. We dress a survey of related work in Section 2. In Section 3 an overview and the context of our approach are given. Section 4 provides a formal background needed for our *X-Warehousing* proposal. Section 5 details how we build *XML Cubes* from initial XML sources. We present in Section 6 a *Java* application we implemented. A case study on a real complex data is illustrated in Section 7. Finally, we conclude and propose future work in Section 8.

2 Related work

Some proposals regarded multidimensional modelling by using XML as a base language for describing data warehouses. Krill [4] affirms that vendors such as *Microsoft*, *IBM*, and *Oracle* will largely employ XML in their database systems for interoperability between data warehouses and tool repositories. Nevertheless, we distinguish two separate approaches in the field.

The first approach focuses on physical storage of XML documents in data warehouses systems. XML feeds warehouses since it is considered an efficient technology to support data within structures well suited for interoperability and information exchange. Baril and Bellahsène introduced the *View Model* [5],

which is a method capable of querying XML databases. A data model is defined for each view to organize semi-structured data. An XML warehouse, called *DAWAX* (*DAta WArehouse for XML*), based on the *View Model* was also proposed. In [6], Hümmer *et al.* proposed an approach, called *XCube*, to model data cubes with XML. *XCube* consists of three kinds of *XML Schemas*: (1) *XCubeSchema* to hold the multidimensional schema; (2) *XCubeDimension* to describe hierarchical structure of the dimensions involved; and (3) *XCubeFact* to describe facts. Nevertheless, this approach focuses on the exchange and the transportation of data cubes over networks rather than multidimensional modeling with XML.

The second approach aims at using XML to design data warehouses according to classical multidimensional models such as *star schemes* and *snow flake schemes*. *XML-star schema* [7] uses *Document Type Definitions* (DTDs) to explicit dimension hierarchies. A dimension is modelled as a sequence of DTDs that are logically associated similarly as the referential integrity does in the relational databases. Golfarelli *et al.* introduced a *Dimensional Fact Model* [8] represented via *Attribute Trees* [2]. They also use *XML Schemas* to express multidimensional models by including relationships with *sub-elements*. Nevertheless, Trujillo *et al.* think that this approach focuses on the presentation of the *multidimensional XML* rather than on the presentation of the structure of the MCM itself [9]. They claim that an *Object Oriented* (OO) standard model is rather needed to cope all multidimensional modeling proprieties at both structural and dynamic levels. Trujillo *et al.* provide a DTD model from which valid XML documents are generated to represent multidimensional models at a conceptual level. Nassis *et al.* propose a similar approach where OO is used to develop a conceptual model for *XML Document Warehouses* (XDW) [10]. An XML repository, called *xFACT*, is built by integrating OO concepts with *XML Schemas*. Nassis *et al.* also define *Virtual dimensions* by using XML and UML package diagrams in order to help the construction of hierarchical *conceptual views*.

The aim of our proposal is to cover the two previous approaches. The *X-Warehousing* process is entirely based on XML: it designs warehouses with *XML Schemas* at a logical level, and then feeds them with valid XML documents at a physical level. Further, since it uses XML, our approach can also be considered a real solution for warehousing heterogenous and complex data in order to prepare them for future OLAP analysis.

3 Overview and context of our approach

Since we need to prepare XML documents to future OLAP analysis, storing them in a data repository is not a sufficient solution. We rather need to express through these documents a more interesting abstraction level completely oriented to analysis objectives. *X-Warehousing* builds a collection of homogeneous XML documents. Each document corresponds to an OLAP fact where the XML formalism structures data according to a multidimensional model. In order to do so, we propose to match and validate XML documents against a MCM (*star* or *snow flake* schema) modelled via a reference *XML Schema*.

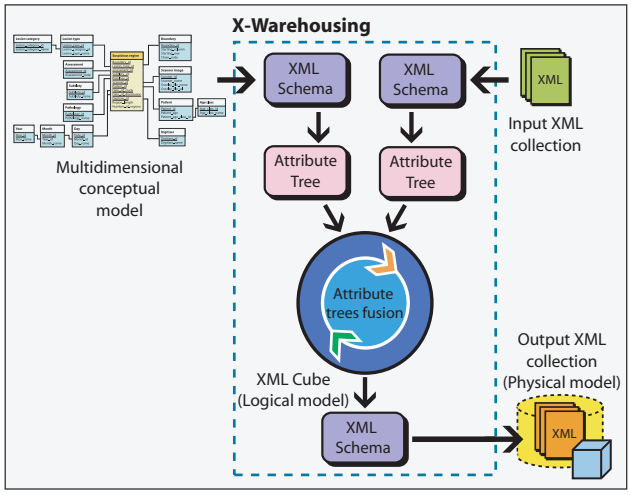

Fig. 1. Overview of the *X-Warehousing* approach

As presented in Figure 1, the *X-Warehousing* approach accepts a reference MCM and XML documents in input. In fact, through the reference MCM a user may design a data warehouse by defining facts, dimensions, and hierarchies. This MCM reflects analysis objectives needed by the user. This model is then transformed to a logical model via an *XML Schema* (XSD file). In a second step, an *attribute tree* [2] is automatically generated from the previous XSD file. Once the reference model is defined, we can submit XML documents to feed the designed warehouse. *XML Schemas* and *attribute trees* are also extracted from input XML documents. We transform the reference model and XML documents into *attribute trees* in order to make them comparable. In fact, two *attribute trees* can easily be merged together through fusion based on *pruning* and *grafting* functions [3]. At this stage, two cases are possible: (1) if an input document contains a minimum information required in the reference MCM, the document is accepted and merged with the MCM. An instance of the XML documents is created and validated against the resulted *XML schema*. This new *XML Schema* represents the logical model of the final *XML Cube*; (2) else, if a submitted document does not contain enough information to represent an OLAP fact according to the reference MCM, the document will be rejected and no output is provided. The goal of this condition is to obtain an homogeneous collection of data with minimum information capable to feed the final *XML Cube*.

The interest of our approach is quite important since organizations are treating domains of complex applications. In these application, a special consideration is given to the integration of heterogenous and complex information in DSS. For example, in *breast cancer researches*, experts require efficient representations of mammographic exams. Note that information about a mammogram come from different sources like texts, experts annotations, and radio scanners. We think that structuring such a set of heterogenous data within XML format is an inter-

esting solution for warehousing them. Nevertheless, this solution is not sufficient for driving future analysis. We propose to structure these data in XML format with respect to a multidimensional reference model of a data warehouse. Output XML documents of the *X-Warehousing* process represent the physical model of the data warehouse. Each output document corresponds to a multidimensional structured information of an OLAP fact.

In the following, we base our study on a running example about the *breast cancer* domain. A collection of input XML documents describing suspicious regions of cancer tumors is already created from the *Digital Database for Screening Mammography*[2] [11].

4 Formal background

In this section, we provide a formalization for our *X-Warehousing* approach. We recall conceptual aspects of typical data warehouse models, i.e., *star schema* and *snow flake schema*. Then, we propose a logical model of data warehouses extracted from the conceptual model, and represented by both *XML Schemas* and *attribute trees* [3].

4.1 Conceptual warehouse models

In general, the conceptual model of a data warehouse is a description of dimension and fact tables. *star schema* and *snow flake schema* are two main variants of this approach. From a relational point of view, a *star schema* consists in one fact table surrounded by independent r dimension tables, i.e., there is no particular relations between dimension tables.

Definition 1. *(Star schema)*
Let $\mathcal{D} = \{D_s, 1 \leq s \leq r\}$ be a set of r independent dimension tables. Each table D_s has $D_s.PK$ as a primary key. F is a fact table with d multi-part keys. A "star schema" is defined by the couple $(F, \mathcal{D})$ that satisfies the following conditions :

- $\forall t \in \{1, \ldots, d\}$, *it exists exactly one* $s \in \{1, \ldots, r\}$ *such as* $F.K_t = D_s.PK$;
- $\forall s \in \{1, \ldots, r\}$, *it exists one or many* $t \in \{1, \ldots, d\}$ *such as* $F.K_t = D_s.PK$.

According to the previous definition, each multi-part key from a fact table is linked to exactly one dimension table. Whereas, a dimension can be linked to one or many multi-part keys in the fact table. This situation can be encountered in many real world modeling problems. For instance, a *Sale* fact can be characterized by an *Origin Country* and an *Destination Country*.

In OLAP analysis, we usually need more than a single granularity level of information in one dimension. For example, to learn about the detailed *Origin* of a *Product*, a multidimensional is supposed to cope information as well about the *State* of the *Product* as its *Country*, its *District*, and its *Office*. In order to do so, a dimension may be expressed through a multi-level hierarchy. From a

[2] http://marathon.csee.usf.edu/Mammography/Database.html

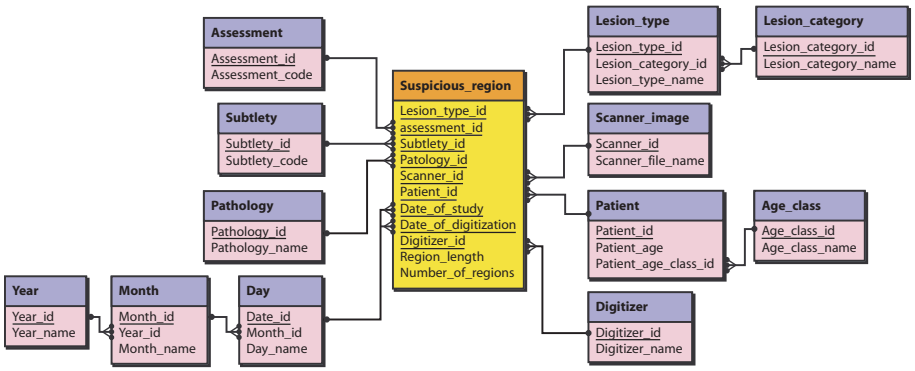

Fig. 2. Conceptual model of "Suspicious Region" data cube

conceptual point of view, a hierarchy with l levels is generally represented by a set of l tables $D_1, \ldots, D_t, \ldots, D_l$, where $\forall t \in \{2, \ldots, l\}$ the primary key $D_t.PK$ of D_t is an attribute (foreign key) in D_{t-1}. In other terms, tables of a hierarchy are linked by a semantic inclusion: $D_1 \subset \cdots \subset D_{t-1} \subset D_t \subset \cdots \subset D_l$. For example, one tuple from table *Office* is semantically included to another tuple from table *District*. In the same way, a *District* is semantically included to a *Country*, and so on. We assume that the primary key of a hierarchy corresponds to the the primary key of its first table D_1, which represents the finest granularity level of the dimension.

Definition 2. *(Snow flake schema)*
Let $\mathcal{H} = \{H_s, 1 \leq s \leq r\}$ be a set of r independent hierarchies. Each hierarchy H_s has $H_s.PK$ as a primary key. F is a fact table with d multi-part keys. A "snow flake schema" is defined by the couple $(F, \mathcal{H})$ that satisfies the following conditions:

- *$\forall t \in \{1, \ldots, d\}$, it exists exactly one $s \in \{1, \ldots, r\}$ such as $F.K_t = H_s.PK$;*
- *$\forall s \in \{1, \ldots, r\}$, it exists one or many $t \in \{1, \ldots, d\}$ such as $F.K_t = H_s.PK$.*

A *snow flake schema* is quite similar to a *star schema*. It consists in one fact table surrounded by a set of dimensions, where each dimension is represented by a hierarchy instead of a single table. For example, the MCM of the Figure 2 displays a data cube of *suspicious regions* (tumors detected on mammographic screens) organized according to a *snow flake schema*. The conceptual representation of data warehouses is a way through what users can easily define future analysis objectives. We emphasize that the relational formalism as used here aims at representing both multidimensional data structure and analysis objectives.

4.2 Modelling a warehouse with XML

An XML document consists in nested element structures, starting with a root element. Each element can contain sub-elements and attributes. Both elements

and attributes are allowed to have values. Attributes are included, with their respective values, within the element's opening declaration (tag). Between an opening and a closing tag of an element, any number of sub-elements can be present. According to these properties, we propose to represent the above conceptual models (*star schema* and *snow flake schema*) of data warehouses with XML. More precisely, we use *XML schemas* to define the structure of a data warehouse.

To write a *star schema* of a data warehouse within XML, we define the notion of an *XML star schema* as follows:

Definition 3. *(XML star schema)*
Let $(F, \mathcal{D})$ be a star schema, where F is a fact table having m measure attributes $\{F.M_q, 1 \leq q \leq m\}$ and $\mathcal{D} = \{D_s, 1 \leq s \leq r\}$ is a set of r independent dimension tables where each D_s contains a set of n_s attributes $\{D_s.A_i, 1 \leq i \leq n_s\}$. The "XML star schema" of $(F, \mathcal{D})$ is an XML schema where:

- *F defines the XML root element in the XML schema;*
- *$\forall q \in \{1, \ldots, m\}$, $F.M_q$ defines an XML attribute included in the the XML root element;*
- *$\forall s \in \{1, \ldots, r\}$, D_s defines as many XML sub-elements of the XML root element as times it is linked to the fact table F;*
- *$\forall s \in \{1, \ldots, r\}$ and $\forall i \in \{1, \ldots, n_s\}$, $D_s.A_i$ defines an XML attribute included in the XML element D_s.*

Knowing that the XML formalism allows to embed multi-level sub-elements in one XML tag, we use this property to represent XML hierarchies of dimensions. Let $H = \{D_1, \ldots, D_t, \ldots, D_l\}$ be a dimension hierarchy. We can represent this hierarchy by writing D_1 as an XML element and $\forall t \in \{2, \ldots, l\}$, D_t is writing as an XML sub-elements of the XML element D_{t-1}. The attributes of each tables D_t are defined as XML attributes included in the XML element D_t. Therefore, we can also define the notion of *XML snow flake schema*, which is the XML equivalent of a conceptual *snow flake schema*:

Definition 4. *(XML snow flake schema)*
Let $(F, \mathcal{H})$ be a star schema, where F is a fact table having m measure attributes $\{F.M_q, 1 \leq q \leq m\}$ and $\mathcal{H} = \{H_s, 1 \leq s \leq r\}$ is a set of r independent hierarchies. The "XML snow flake schema" of $(F, \mathcal{H})$ is an XML schema where:

- *F defines the XML root element in the XML schema;*
- *$\forall q \in \{1, \ldots, m\}$, $F.M_q$ defines an XML attribute included in the the XML root element;*
- *$\forall s \in \{1, \ldots, r\}$, H_s defines as many XML dimension hierarchies, like sub-elements of the XML root element, as times it is linked to the fact table F;*

Based on properties of XML formalism, *XML Schemas* enable to write a logical model of a data warehouse from its conceptual model. Our approach does not only use the XML formalism to design data warehouses (or data cubes),

but also feeds them with data. We use XML documents to support information relative to the designed facts. As an XML document supports values of elements and attributes, we assume that it contains information about a single OLAP fact. We say that an XML document supports an *XML fact* when it is valid against an *XML star schema* or an *XML snow flake schema* representing a logical model of a warehouse. For instance, Figure 3 shows an example of an *XML fact* associated to the conceptual model of the "Suspicious Region" data cube presented in Figure 2. Note that at a physical level, the *XML Cube*, introduced in the section 3, corresponds to a set of *XML facts*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Suspicious_region Region_length="287" Number_of_regions="6">
    <Patient Patient_age="60" >
        <Age_class Age_class_name="Between 60 and 69 years old" />
    </Patient>
    <Lesion_type Lesion_type_name="calcification type round_and_regular distribution n/a">
        <Lesion_category Lesion_category_name="calcification type round_and_regular" />
    </Lesion_type>
    <Assessment Assessment_code="2" />
    <Subtlety Subtlety_code="4" />
    <Pathology Pathology_name="benign_without_callback" />
    <Date_of_study Date="1998-06-04">
        <Day Day_name="June 4, 1998">
            <Month Month_name="June, 1998">
                <Year Year_name="1998" />
            </Month>
        </Day>
    </Date_of_study>
    <Date_of_digitization Date="1998-07-20">
        <Day Day_name="July 20, 1998">
            <Month Month_name="July, 1998">
                <Year Year_name="1998" />
            </Month>
        </Day>
    </Date_of_digitization>
    <Digitizer Digitizer_name="lumisys laser" />
    <Scanner_image Scanner_file_name="B_3162_1.RIGHT_CC.LJPEG" />
</Suspicious_region>
```

Fig. 3. An example of an XML fact

4.3 Attribute trees

The concept of *Attribute trees* was first introduced in [2] by Golfarelli *et al.*. An *attribute tree* is a directed, acyclic and weakly connected graph that represents a warehouse schema. In [3], Golfarelli and Rizzi have also proposed a general framework for data warehouses design. They developed a data warehouse model called *Dimensional Fact Model*. This model is obtained by a semi-automated technique that starts from the E/R schemas and from the logical schemas describing it. A *Dimensional Fact Model* is represented by an *attribute tree* on which it is possible to apply algorithms in order to transform it.

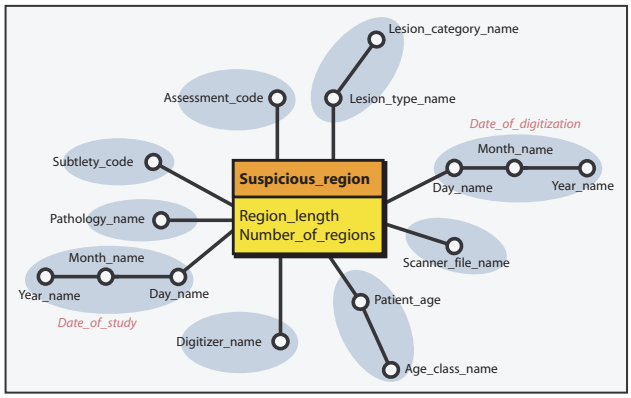

Fig. 4. Attribute tree associated to the "Suspicious Region" data cube

In order to handel data warehouses and to be able to transform their schemas, we also represent their logical model via *attribute trees*. For example, Figure 4 shows the *attribute tree* associated to the multidimensional model of the "Suspicious Region" data cube presented in Figure 2.

5 Building XML Cubes

Recall that our approach starts from a reference MCM corresponding to future analysis objectives. The reference MCM will be matched with complex data presented in XML documents. In order to make them comparable, both MCM and XML documents are transformed to *attribute trees*. As explained in Subsection 5.1, the comparison of *attribute trees* is realized by fusion operations according to *pruning*, and *grafting* functions [3]. Nevertheless, an XML document which does not contains sufficient information about defined analysis objectives is naturally rejected from the final warehouse. Thus, we introduce in Subsection 5.2 the notion of *Minimal XML document content*.

5.1 Fusion of attribute trees

The *pruning* and the *grafting* functions provide from two input *attribute trees* a merged *attribute tree* which contains the maximum of common attributes by respect to their relative relationships.

The fusion by *pruning* is carried out by dropping any uncommon subtree starting from the root vertex. The attributes dropped are not included in the merged tree. For example, in Figure 5(a), only common vertexes (black circles) in the two input trees are kept in the resulting tree. All other uncommon vertexes (white circles) are therefore dropped with their subtrees.

The fusion by *grafting* is used when common subtrees do not have a same structure of relationships in two input trees. In this case we need to pick up common attributes by preserving their general relationships. When an uncommon vertex is dropped, the grafting function checks wether its descendants contain

common vertex or not. The common descendants are therefore preserved in the merged tree. For example, in Figure 5(b), uncommon vertexes `x` and `y` are dropped, but since their respective descendants (`d`, `e` and `b`) are common, they are kept in the merged tree.

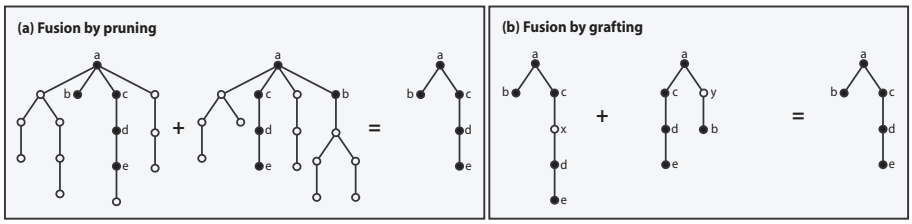

Fig. 5. Examples of fusion of two attribute trees by (a) pruning and by (b) grafting

5.2 Minimal XML document content

In some cases, when an input XML document does not contain enough information required by the analysis objectives, the fusion provides a poor output XML document, which represents an OLAP fact with missing data. It is naturally useless to feed the warehouse with such a document. In order to check wether an input XML document contains enough information to feed the warehouse or not, we introduce the *Minimal XML document content*. The *minimal XML document content* is an information threshold entirely defined by users when submitting the MCM to express analysis objectives. At this stage, a user can declare for each measure, dimension, and dimension hierarchy wether it is mandatory or optional according to his objectives and to the information he needs to see in the final *XML Cube*. The *minimal XML document content* corresponds to the *attribute tree* associated to mandatory elements declared by the user when submitting the data cube model.

Recall that our approach aims at building a data cube with XML sources that allows future OLAP analysis. It is naturally not possible to decide with an automatic process which element in a future analysis context may be optional or not. It is entirely up to the user to define the *minimal XML document content*. Nevertheless, by default, we suppose that all measures and dimensions attributes of a submitted data cube model are mandatory in the final *XML Cube*. We also suppose that not all measures can be optional elements in the data cube. Indeed, in an analysis context, OLAP facts without a measure could not be exploited by OLAP operators such as aggregation. For this, users are not allowed to set all the measures to optional elements. At least one measure in the submitted data cube model must be mandatory.

At the fusion step, the *attribute tree* of an input XML document is controlled. If it contains all mandatory elements required by the user, it will be merged with the *attribute tree* of the data cube model. Else, it will be rejected, the fusion process will be canceled, and therefore no output document will be created.

6 Implementation

The core programm of the *X-Warehousing* application is developed with *Java* and runs on all Java-enabled platforms (Figure 6(b)). The application contains two main modules: the *Model Loader Module* and the *Model Merger Module*.

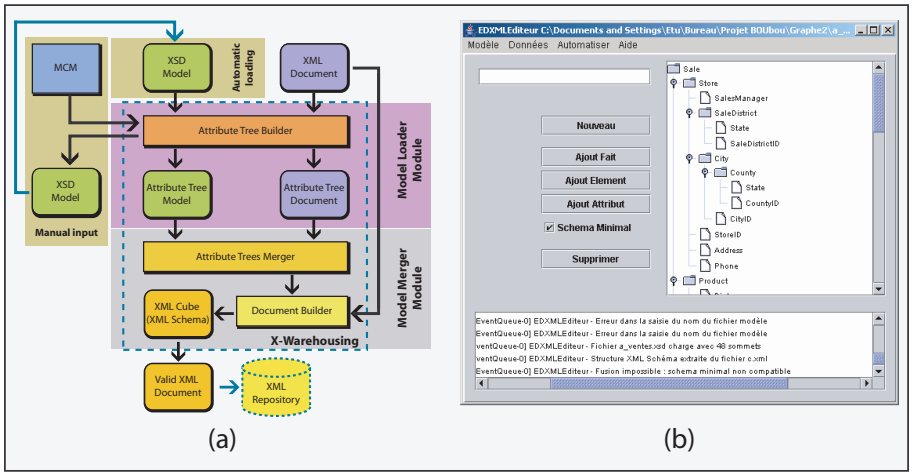

Fig. 6. (a) Architecture (b) Interface of the X-Warehousing application

6.1 Model Loader Module

A reference data cube model can be submitted by a manual input or by loading an XSD file associated to a MCM. In the case of a manual submission, the loader module transforms the data cube model to an *XML Schema* and then to an *attribute tree*. The *attribute tree* is saved into an XSD file, which will be displayed within a hierarchical tree (Figure 6(b)) via a *JTree Object*. If a user loads an XSD file, an algorithm parses it and feeds an internal *attribute tree* object structure. We consider each XSD file as a *JDOM* document type. Then, we use the *JDOM API* to scan the document and build *attribute trees*. On the other hand, the *Model Loader Module* loads input XML documents containing data and their underlying structure. It also extracts the XSD file and the *attribute tree* corresponding to an input XML document.

6.2 Model Merger Module

Once a data cube reference model and input XML documents are loaded, the *Model Merger Module* can run according to an *automatic* or a *manual* mode. The two modes use the same core algorithms. Nevertheless, the automatic mode picks up XML documents from a specified directory, validates them against the reference model and saves them in an XML repository automatically within a

looping mode. The *Model Merger Module* works with the help of fusion functions presented in Section 5. Figure 7 shows function MergeTree which merges two *attribute trees*. This function goes through each branch of the tree, reads the tree of the data cube model and the tree of an input XML document and feeds a new XML document with the resulted model structure. When a vertex from the reference tree does not match with the document tree, MergeTree sets the arc value to zero. Then, it re-writes the tree with only non-null arcs.

```
Function MergeTree(tree1,tree2)
    tree3=DuplicateTree(tree1)
    While Not(end(nodeList(tree3)))
        vertex1=GetVertex(tree3)
        While Not(end(nodeList(tree2)))
            vertex2=GetVertex(tree2)
            If vertex2=vertex1 Then vertex1.arc = 0
        End While
    End While
    Tree3=WriteTree(tree3)
End Function
```

Fig. 7. The function MergeTree

7 A case study

We run our *X-Warehousing* approach on a real world application domain. We consider the *screening mammography* data cube presented in Figure 2 as a reference MCM. We use a collection of 4 686 XML documents as input data to be warehoused [3]. All these documents have the same structure and are valid against the same *XML Schema*. Therefore they have the same *attribute tree*. Figure 8 shows the *attribute tree* associated to these input XML documents. Once the reference MCM and the input XML documents are submitted, our application achieve the fusion of *attribute trees* displayed in Figures 8 and 4. The result of this step closely depends on the *minimal XML document content* defined at the submission of the reference MCM.

For example, let set to mandatory all the dimensions and all the measures of the reference model. In this case, all the input XML documents will be rejected and no output will be obtained. In fact, note that the *Assessment_code* attribute is absent in the *attribute tree* of input XML documents. Therefore, as this attribute is mandatory in the reference model, the *Attribute Tree Merger* will reject each XML document that does not include it.

Suppose now that we define a more flexible *minimal XML document content* by setting *Assessment* dimension to optional. In this case, the absence of the *Assessment_code* attribute in input XML document would not prevent the fusion step of attribute trees. Therefore, the *Model Merger Module* provides a logical model of an *XML Cube* represented by the *XML schema* of Figure 9. Further, for each input document, an *XML fact* (XML document) is generated.

[3] The collection of XML documents is available at:
http://eric.univ-lyon2.fr/∼rbenmessaoud/?page=donnees&section=3

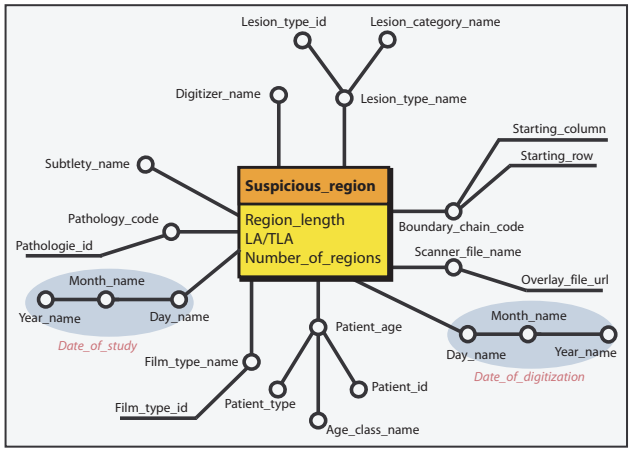

Fig. 8. Attribute tree of input XML documents

Note that, in this case study, as all the input XML documents have the same structure, all the generated *XML facts* are valid against the same *XML Cube* of Figure 9. Note also that uncommon attributes in *attribute trees* of Figures 8 and 4 are pruned by the *Attribute Trees Merger* component of our application. For instance, the attributes *Lesion_type_id*, *Boundary_chain_code*, and *Patient_id* are dropped, and therefore do not exist in the *XML Cube*.

Finally, through this case study, we prove the capability of our approach to use XML both to design and to store complex data according to a multidimensional structure that reflects analysis objectives required by users. XML can therefore be considered as a logical and physical description platform for future analysis tasks on complex data.

8 Conclusion and future work

In this paper, we proposed a methodology entirely based on the XML formalism to warehouse complex data. Our approach, called *X-Warehousing*, does not simply feed a repository with XML documents, but also expresses an interesting abstraction level by preparing XML documents to future analysis. In fact, it consists in validating documents against an *XML Schema* which designs a data warehouse. We defined a general formalization for modelling *star* and *snow flake schemas* within XML. We also use the concept of *attribute trees* [2] in order to help the creation and the warehousing of homogeneous XML documents by merging initial XML sources with a reference multidimensional model. Constraints on the created XML documents can be required and expressed by users. To validate our *X-Warehousing* approach, we implemented a Java application which loads in input a reference multidimensional model and XML documents. It provides a logical and a physical model of an *XML cube* composed of homogeneous XML documents where each document corresponds to an OLAP fact

that respects data required constraints. A case study on *breast cancer* domain is provided to show the interest of employing our approach in a real world field for designing and warehousing complex data by using XML.

For future work, a lot of issues need to be addressed to our *X-Warehousing* approach. The first is devoted to a performance study of OLAP queries in order to achieve analysis on XML documents as provided in the *XML Cube* of our present approach. The second issue should deal with experimental tests on the reliability of the developed application. This includes studies on complexity and time processing of loading input XML documents, building attribute trees, fusion of attribute trees, and creation of output XML documents. Third, we should solve the problem of updating the *XML Cube* when we need to modify the reference MCM in order to change analysis objectives. Finally, some optimization are also needed on the *Model Loader Module* architecture. For instance, when we submit a collection of XML documents having the same structure, the application does not need to generate an *XML Schema* and an *attribute tree* for each input document.

References

1. Kimball, R.: The Data Warehouse Toolkit. John Wiley & Sons (1996)
2. Golfarelli, M., Maio, D., Rizzi, S.: Conceptual Design of Data Warehouses from E/R Schema. In: HICSS'98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 7, IEEE Computer Society (1998) 334
3. Golfarelli, M., Rizzi, S.: Designing the data warehouse: key steps and crucial issues. Journal of Computer Science and Information Management **2**(3) (1999) 88–100
4. Krill, P.: XML builds momentun as repository standard. InfoWorld **20**(25) (1998) 6
5. Baril, X., Bellahsène, Z.: Designing and Managing an XML Warehouse. In: XML Data Management: Native XML and XML-Enabled Database Systems. First edn. Addison Wesley Professional (2003) 455–474
6. Hümmer, W., Bauer, A., Harde, G.: XCube: XML for data warehouses. In: DOLAP, ACM (2003) 33–40
7. Pokorný, J.: Modelling stars using XML. In: DOLAP'01: Proceedings of the 4th ACM international workshop on Data warehousing and OLAP, Atlanta, Georgia, USA, ACM Press (2001) 24–31
8. Golfarelli, M., Rizzi, S., Vrdoljak, B.: Data Warehouse Design from XML Sources. In: DOLAP'01: Proceedings of the 4th ACM international workshop on Data warehousing and OLAP, Atlanta, Georgia, USA (2001)
9. Trujillo, J., Lujàn-Mora, S., Song, I.Y.: Applying UML and XML for Designing and Interchanging Information for Data Warehouses and OLAP Applications. Journal of Database Management **15**(1) (2004) 41–72
10. Nassis, V., Rajugan, R., Dillon, T.S., Rahayu, J.W.: Conceptual Design of XML Document Warehouses. In Kambayashi, Y., Mohania, M.K., Wöß, W., eds.: DaWaK. Volume 3181 of Lecture Notes in Computer Science., Springer (2004) 1–14
11. Heath, M., Bowyer, K., Kopans, D., Moore, R., Jr, P.K.: The Digital Database for Screening Mammography. In: The Proceedings of the 5th International Workshop on Digital Mammography, Toronto, Canada, Medical Physics Publishing (Madison, WI) (2000)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns="http://www.w3schools.com">
  <xs:element name="Suspicious_region">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Patient" type="Patient_Type" />
        <xs:element name="Lesion_Type" type="Lesion_Type_Type" />
        <xs:element name="Subtlety" type="Subtlety_Type" />
        <xs:element name="Pathology" type="Pathology_Type" />
        <xs:element name="Date_of_study" type="Date_Type" />
        <xs:element name="Date_of_digitization" type="Date_Type" />
        <xs:element name="Digitizer" type="Digitizer_Type" />
        <xs:element name="Scanner_image" type="Scanner_Type" />
      </xs:sequence>
      <xs:attribute name="Region_length" type="xs:integer" />
      <xs:attribute name="Number_of_regions" type="xs:integer" />
    </xs:complexType>
  </xs:element>
  <xs:complexType name="Patient_Type">
    <xs:sequence>
      <xs:element name="Age_class">
        <xs:complexType>
          <xs:attribute name="Age_class_name" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Patient_age" type="xs:integer"/>
  </xs:complexType>
  <xs:complexType name="Lesion_Type_Type">
    <xs:sequence>
      <xs:element name="Lesion_category">
        <xs:complexType>
          <xs:attribute name="Lesion_category_name" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Lesion_type_name" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Subtlety_Type">
    <xs:attribute name="Subtlety_code" type="xs:integer"/>
  </xs:complexType>
  <xs:complexType name="Pathology_Type">
    <xs:attribute name="Pathology_name" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Date_Type">
    <xs:sequence>
      <xs:element name="Day">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Month">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Year">
                    <xs:complexType>
                      <xs:attribute name="Year_name" type="xs:integer"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="Month_name" type="xs:string"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="Day_name" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Date" type="xs:date"/>
  </xs:complexType>
  <xs:complexType name="Scanner_Type">
    <xs:attribute name="Scanner_file_name" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

Fig. 9. Logical model of the "Suspicious Region" XML Cube