# Conception et construction d'entrepôts en XML

Omar Boussaid, Riadh Ben Messaoud, Rémy Choquet, Stéphane Anthoard

Laboratoire ERIC, Université Lyon 2
Campus Porte des Alpes, 69676 Bron Cedex
Omar.Boussaid@univ-lyon2.fr , rbenmessaoud@eric.univ-lyon2.fr,
(remy.choquet,stephanea)@gmail.com,
http://eric.univ-lyon2.fr

**Résumé.** Les entreprises sont de plus en plus concernées par des données dites complexes, se présentant sous une forme autre que numérique ou symbolique, issues de sources différentes et ayant des formats hétérogènes. Pour une exploitation à des fins décisionnelles, ces données complexes nécessitent un travail préparatoire pour les structurer et les homogénéiser.
La prolifération des données sous forme de documents XML incite à une solution d'entreposage. Nous proposons dans ce papier une approche basée sur XML, d'entreposage de données complexes contenues dans des documents XML, appelée *X-Warehousing*. Celle-ci définit une méthodologie pour concevoir des entrepôts de données complexes à l'aide du formalisme XML. Pour valider notre approche, nous avons implémenté une application *Java* et nous avons réalisé une étude de cas sur des données complexes concernant les régions suspectes sur des mammographies.

## 1 Introduction

Avec l'avènement des nouvelles technologies de communication et plus précisément l'internet, les entreprises recueillent des masses de données de plus en plus importantes. Ces données étant généralement hétérogènes car elles proviennent de différentes sources. Elles sont dites complexes car elles sont de formats différents et sont sur des supports différents. En médecine, le dossier d'un patient contient des informations générales sur le patient (age, sexe, etc) ; ainsi que des images de scanner, des interrogatoires sous forme d'enregistrements sonores ou des compte-rendus manuscrits de médecins. Pour exploiter de telles données à des fins décisionnelles, il est nécessaire de les structurer et de les homogénéiser. Le langage XML (eXtensible Markup Language) s'avère comme une solution appropriée à ce travail préparatoire sur les données complexes. XML est une norme de W3C[1] et est considéré comme un standard dans la description et l'échange des données. Il représente les données de façon semi-structurée. Sa capacité d'auto-description et sa structure arborescente donne à ce formalisme une grande flexibilité et une puissance suffisante pour décrire des données complexes, hétérogènes et provenant de sources éparpillées. Les données sont alors stockées dans des documents

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

XML formés conformément à une grammaire associée, exprimée sous forme de DTD (*Document Type Definition*) ou de *Schéma XML*. Si les DTDs n'offrent qu'un seul type de données (chaînes de caractères), les schémas XML utilisent davantage de types de données et permettent également d'en définir des types complexes.

Le document XML est perçu, incontestablement, comme un moyen efficace pour représenter et stocker des données. Cependant, dans le cadre des applications décisionnelles, des efforts sont nécessaires, plus précisément d'un point de vue méthodologique, pour construire des solutions d'entreposage de documents XML. Cette difficulté est due à l'organisation multidimensionnelle des entrepôts de données qui diffère de celle semi-structurée des documents XML. Un entrepôt de données a une architecture intégrée, centralisée, orientée sujets et nécessite des rafraîchissements périodiques des données pour garantir leur historisation, Kimball et Ross (2002); Inmon (2005). A partir de l'entrepôt, il est possible de construire des cubes de données représentant des contextes d'analyse multidimensionnels. Ainsi, la difficulté est comment peut-on concevoir un modèle multidimensionnel à l'aide d'un formalisme semi-structuré tel que XML ?

Dans ce papier, nous proposons une approche d'entreposage de données complexes entièrement basée sur XML. Celle-ci, appelée *X-Warehousing*, propose une démarche méthodologique pour construire des entrepôts ou des magasins de données XML en XML. Plus précisément, elle permet de concevoir un entrepôt (ou un magasin), de représenter son schéma conceptuel à l'aide de schémas XML et enfin d'alimenter la structure multidimensionnelle à l'aide de données initialement stockées dans des documents XML. Il est possible de représenter avec XML, des données complexes comme une information entière plutôt qu'un ensemble d'informations séparées comme c'est le cas dans un dossier de patient par exemple. Une collection de documents XML -où chaque document est une entité informationnelle- peut alors être considérée comme une base décisionnelle représentant un entrepôt (ou un magasin) de données complexes. Le choix de construire des structures multidimensionnelles en XML est dicté par le fait de la prolifération des documents XML d'une part, et par le fait que les données complexes sont décrites sous forme de documents XML, d'autre part. La structure classique d'un cube de données n'est plus appropriée dans ce cas.

En fait, *X-Warehousing* est une approche pilotée par les besoins d'analyse. Elle part des objectifs d'analyse d'un utilisateur représentés par un modèle conceptuel multidimensionnel (*MCM*). Elle utilise un ensemble de données complexes structurées dans des documents XML pour générer une base organisée de façon multidimensionnelle exprimant un contexte d'analyse. Il n'est pas utile de chercher au départ une structure commune multidimensionnelle dans les documents XML sources, car elle n'existe pas à priori. Ainsi, nous nous focalisons plutôt sur les besoins d'analyse de l'utilisateur que nous représentons par un schéma en étoile, Kimball (1996); Chaudhuri et Dayal (1997). Ce dernier exprime comment représenter conceptuellement les données pour les orienter vers l'analyse et cela indépendamment de la manière de les organiser aux niveaux logiques ou physiques (c.f. sous-section 4.1).

Le MCM et les documents XML sources sont initialement exprimés à l'aide de schémas XML pour être appariés. Cependant pour les rendre comparables, ils sont alors traduits sous forme d'arbre d'attributs, Golfarelli et al. (1998); Golfarelli et Rizzi (1999). Les deux arbres d'attributs représentant respectivement le MCM et les documents XML sources sont fusionnés selon une certaine stratégie à l'aide d'opérations de fusion par élagage (*pruning*) et/ou par greffe (*grafting*), Golfarelli et al. (2001a,b).

De plus, dans notre approche, nous n'extrayons dans les documents XML sources que les données utiles pour les besoins d'analyse, pour alimenter le cube à construire. Nous employons volontairement le terme de *cube* au lieu d'entrepôt (ou magasin) pour désigner la structure multidimensionnelle décrite en XML et alimentée par des données se trouvant dans des documents XML. Et cela, pour privilégier la véritable vocation de celle-ci qui est l'analyse des données complexes constituées en fait OLAP (*On-Line Analytical Processing*), plutôt que leur stockage. Le cube sera composé de documents XML valides et de plus conformes au MCM de départ, c'est à dire aux objectifs d'analyse de l'utilisateur. Chaque document XML de ce cube représente un fait OLAP constitué d'un ou plusieurs indicateurs (mesures) à observer à travers des axes d'analyse (dimensions et hiérarchies de dimensions). L'ensemble des documents XML entreposés correspond au modèle physique du cube de données que nous désignons par *cube XML*.

Le reste du papier est organisé comme suit. Après l'état de l'art présenté dans la section 2, nous exposerons le contexte de notre approche dans la section 3. La section 4 indiquera le formalisme de *X-Warehousing*. Alors que dans la section 5, nous montrons comment nous construisons un cube XML à partir de documents XML sources. Nous validons notre approche par une implémentation en java présentée dans la section 6 ; et nous l'illustrons par une étude de cas sur des dossiers de patientes atteintes de cancer des seins dans la section 7. Nous terminons dans la section 8, par une conclusion et les perspectives qu'ouvre le travail présenté dans le cadre de ce papier.

## 2 État de l'art

Dans Krill (1998), Krill affirme que les fournisseurs tels que *Microsoft,IBM*, et *Oracle* emploieront largement XML dans leurs systèmes de base de données pour l'interopérabilité entre les entrepôts de données. Des travaux de recherche s'intéressent déjà à la modélisation multidimensionnelle et au langage XML. Ce dernier est utilisé comme langue de base pour décrire des magasins de données. Les travaux de Rusu, Rusu et al. (2004), abordent la problématique des entrepôts XML et aboutissent déjà sur des solutions pour les différentes tâches du processus d'entreposage. D'autre part, Pokorny propose dans Pokorný (2001) une structure de données, appelée *XMLStar schéma* avec des hiérarchies de dimensions explicites utilisant des DTDs pour décrire la structure des objets. Une dimension est modélisée comme une séquence de DTDs logiquement associées, telle l'intégrité référentielle dans les bases de données relationnelles. Pokorny définit également la notion *d'intégrité référentielle XML* et ses propriétés requises pour caractériser des collections de XML. Dans Golfarelli et al. (2001a), XML est devenu une solution intéressante pour l'intégration des sources d'information dans des systèmes décisionnels. Néanmoins, ils notent que le mapping logique entre les documents XML sources et le schéma cible doit être défini par le concepteur. Ainsi, ils utilisent leur *Modèle Dimensionnel de Faits* basé sur la représentation *d'arbre d'attributs*, Golfarelli et al. (1998), et proposent une approche semi-automatique pour construire le schéma conceptuel d'un data mart directement à partir des sources XML. Dans Golfarelli et al. (2001a), les auteurs montrent la possibilité de recourir aux schémas XML pour représenter des modèles multidimensionnels en exprimant les relations à l'aide de *sous-éléments* et de clef *REF/IDREF*. Cependant, Trujillo *et al*. pensent que cette approche se focalise plus sur la présentation du ce que peut être le *XML multidimensionnel*, que sur la structure du modèle conceptuel multidimensionnel lui-même,

Trujillo et al. (2004). Les auteurs affirment qu'à des niveaux structurel et dynamique, un modèle conceptuel standard doit considérer toutes les propriétés multidimensionnelles. Pour cela, l'approche *orientée objet*(OO) avec UML (*Unified Modeling Language*) est utilisée. Pour faciliter l'échange du modèle conceptuel multidimensionnel, les auteurs produisent une DTD à partir de laquelle des documents XML valides sont générés pour représenter des modèles multidimensionnels à un niveau conceptuel.

Nassis *et al.* proposent une approche OO similaire pour développer un modèle conceptuel d'un entrepôt de documents XML (*XDW)*, Nassis et al. (2004). Elle permet de concevoir et de construire un *repository* (une collection) de documents XML appelé *xFACT*. Ils utilisent des concepts objets et des schémas XML pour intégrer les données. Les auteurs définissent également des *dimensions virtuelles* des vues conceptuelles XML pour traiter les besoins d'analyse de l'utilisateur. Ils utilisent des diagrammes de packages UML pour construire des vues conceptuelles hiérarchiques, Rajugan et al. (2003), et rehausser la sémantique et l'expressivité de XDW. Baril et Bellahsène proposent une approche de manipulation d'une base de données XML, *appelée Modèle de Vues*, Baril et Bellahsène (2000). Ils ont défini un mécanisme de vues pour restructurer les données contenues dans des documents XML et répondre aux besoins d'analyse de l'utilisateur. Ils ont défini un modèle de données pour chaque vue pour représenter des données semi structurées. Ce modèle de données peut être utilisé pour améliorer les performances des requêtes. Dans Baril et Bellahsène (2003), les auteurs proposent *DAWAX*, un entrepôt de documents XML basé sur le modèle des vues. Leur système consiste en un gestionnaire de vues (*View Manager*), une interface entre les sources XML et les clients. L'entrepôt XML se compose de fragments de vues créées par le gestionnaire de vues et d'un modèle générique. Chaque vue peut être créée par une agrégation de plusieurs sources. Cette approche est puissante dans un environnement multi-sources. Une fois tous les fragments sont créés, ils sont associés aux identificateurs *IDREF* et stockés dans une base de données relationnelle. Hümmer *et al.* ont présenté *XCube*, une famille de documents XML pour l'échange des cubes de données dans Hümmer et al. (2003). *XCube* supporte un modèle multidimensionnel sur divers sources de données. Il se compose de trois schémas XML décrivant entièrement un cube de données : (1) *XCubeSchema* pour décrire le schéma multidimensional ; (2) *XCubeDimension* pour décrire la structure hiérarchique des dimensions ; (3) *XCubeFact* qui contient l'ensemble des faits (les cellules du cube). Néanmoins, cette approche utilise XML pour l'échange des cubes de données via l'Internet, plutôt que la modélisation multidimensionnelle avec XML.
Park et al., dans Park et al. (2005), ont proposé une démarche, nommée XML-OLAP, assez proche de X-Warehousing. Elle consiste à créer un entrepôt XML composé d'un ensemble de collections de documents XML représentant les faits et les dimensions. Les auteurs utilisent XQuery et le langage MDX de Microsoft qu'ils ont adapté à XML pour construire des cubes XML.

En conclusion, notons que l'utilisation de XML dans des entrepôts de données nous amène à deux approches différentes.

- La première porte sur le stockage physique des documents XML dans des entrepôts de données. Les documents XML alimentent alors les entrepôts et XML est considéré comme une technologie efficace supportant des données faiblement structurées et adaptée à l'interopérabilité et à l'échange des données.
- La deuxième approche utilise le formalisme XML pour concevoir des entrepôts ou des

magasins de données selon les modèles multidimensionnels classiques tels que les schémas en étoile ou en flocons de neige.

Notre approche *X-Warehousing* est entièrement basée sur XML. Elle permet en même temps, de concevoir des entrepôts avec des schémas XML et de les alimenter avec des documents XML valides. XML est perçu comme une solution potentielle pour l'entreposage des données complexes.

## 3 Contexte général de notre approche

Notre approche ne consiste pas seulement à stocker des documents XML dans une base cible, mais elle apporte un niveau d'abstraction pertinent pour préparer ces derniers à l'analyse. Elle permet de construire des cubes XML. Ces derniers sont composés chacun d'une collection de documents XML. Chaque document correspond alors à un fait OLAP et doit satisfaire certaines contraintes, comme respecter une information minimale pour que le fait à observer soit consistant. Pour cela, nous proposons de valider les documents par un schéma XML. Ce dernier représente le modèle conceptuel du cube qui généralement consiste en un schéma en étoile ou en flocons de neige.

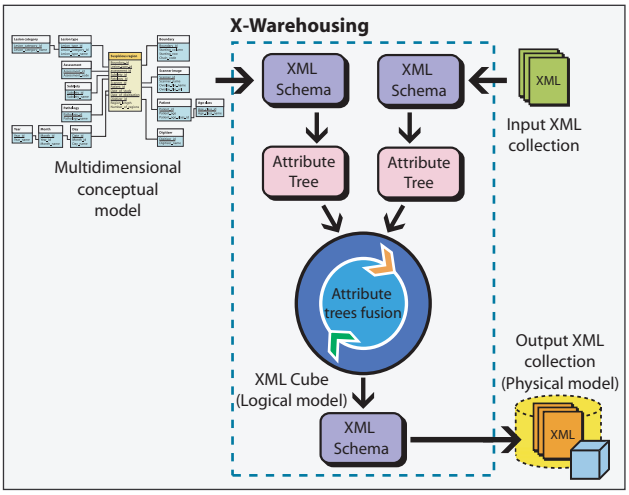


FIG. 1 – *Les étapes de l'approche* X-Warehousing

La figure 1 résume les différentes étapes de notre approche *X-Warehousing*. Au départ, l'utilisateur déclare ses objectifs d'analyse sous la forme d'un modèle conceptuel multidimensionnel (MCM). Ce modèle est exprimé par un schéma XML puis transformé en un arbre d'attributs également représenté par un schéma XML (sauvegardé dans un fichier .XSD). L'utilisateur soumet alors les documents qu'il veut analyser. Ces derniers sont traduits en arbres d'attributs. Chaque arbre d'attributs d'un document XML en entrée est alors comparé à celui du MCM. Deux cas sont alors possibles. (1) Si le document en entrée contient une information minimale requise dans le MCM : il est alors accepté. Une instance de ce document sera

créée et validée en accord avec le schéma XML du cube XML que génère l'application *X-Warehousing*. (2) Si le document soumis en entrée ne contient pas une information suffisante pour un fait OLAP, il est alors rejeté et ne fera donc pas partie du cube XML. cette approche est d'obtenir un ensemble homogène de données avec des contraintes strictes sur leurs contenus. Ces données sont structurées dans des documents XML et orientées vers des analyses en ligne. Cette collection de documents XML représente un cube OLAP que nous désignons par *cube XML*.

# 4 Formalisation

Nous proposons ici des définitions des *schémas en étoile* et *en flocon de neige*. Nous présentons un modèle logique de cubes de données utilisant un schémas XML. Quant aux arbres d'attributs, le lecteur pourra trouvé une formalisation détaillée dans Golfarelli et al. (1998).

## 4.1 Modèle conceptuel multidimensionnel

Un MCM est une description d'objectifs d'analyse. La finalité de cette dernière est de décrire un contexte d'analyse. Il s'agit de représenter un ou plusieurs indicateurs et un ensemble d'axes d'observation et cela d'un point de vue strictement conceptuel, indépendamment de toute considération logique ou physique. Pour représenter ces indicateurs et leurs descripteurs, ceux qui sont à l'origine des schémas en étoile ont proposé d'utiliser des concepts empruntés au formalisme relationnel : des tables pour regrouper les indicateurs d'une part et leurs descripteurs d'autre part. De plus, pour relier ces différentes tables, on a utilisé le mécanisme des clefs (principales, étrangères). Ce dernier se révèle très pratique pour expliquer la configuration en étoile du schéma et cela dans un but d'assurer les performances d'interrogation de ces tables qui sont globalement très volumineuses. Ainsi, l'évocation d'un schéma en étoile est souvent systématiquement rapprochée à un environnement relationnel. Pourtant, ceci n'est toujours pas vrai. Les tableaux mutidimensionnels, les classes objets et aujourd'hui les schémas XML sont d'autres possibilités d'exprimer un schéma en étoile à un niveau logique autrement qu'en relationnel. On gagnerait en clarté à considérer le schéma en étoile comme un modèle strictement conceptuel et cela en l'absence d'un véritable langage de modélisation multidimensionnelle des entrepôts de données. Ainsi, nous utiliserons les concepts de tables, de clefs principales et étrangères pour définir notre MCM.

Le schéma en étoile consiste donc en une table de faits entourée par $r$ tables de dimensions notées $\mathcal{D} = \{D_s, 1 \leq s \leq r\}$ (notons qu'il n'y a pas de relations particulières entre les tables de dimensions). Nous supposons qu'une table de dimension $D_s$ a $D_s.PK$ comme clé primaire et $n_s$ attributs $\{D_s.A_i, 1 \leq i \leq n_s\}$. Une table de faits sera notée $F$ et contient un jeu de clés $d$ ($d \geq 1$) composites tel que $\{F.K_t, 1 \leq t \leq d\}$. Les autres attributs non clés de $F$ seront appelés "***faits***" ou "***mesures***". Enfin, une table de faits $F$ contient $m$ mesures notées $\{F.M_q, 1 \leq q \leq m\}$.

**Definition 1** (*schéma en étoile*)
Soit $\mathcal{D} = \{D_s, 1 \leq s \leq r\}$ un ensemble de $r$ tables de dimensions indépendantes. Chaque table de dimension $D_s$ contient une clé primaire $D_s.PK$. $F$ est une table de faits contenant $d$ clés composites. Un *schéma en étoile* est alors défini par le couple $(F, \mathcal{D})$ vérifiant les conditions suivantes :

– $\forall t \in \{1, \ldots, d\}$, il existe exactement un $s \in \{1, \ldots, r\}$ tel que $F.K_t = D_s.PK$ ;

– $\forall s \in \{1, \ldots, r\}$, , il existe un ou plusieurs $t \in \{1, \ldots, d\}$ tel que $F.K_t = D_s.PK$.

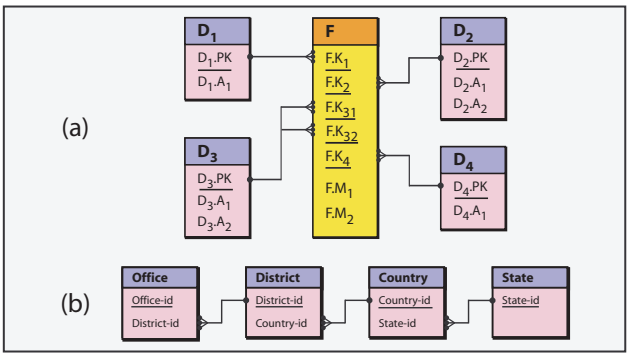


FIG. 2 – *Un exemple d'un schéma en étoile (a) et d'une hiérarchie de dimension (b)*

Cette définition signifie que chaque clé composite de la table des faits est liée à une et une seule table de dimension via sa clé primaire. Alors qu'une table de dimension peut être liée à une ou plusieurs clés composites de la table des faits. Notons que sur la figure 2 (a), la clé primaire $D_3.PK$ est liée à deux clés composites ($F.K_{31}$ et $F.K_{32}$) dans la table des faits. Cette situation peut être rencontrée dans beaucoup de problèmes réels. Par exemple, un fait *Sales* peut être caractérisé par un *Origin Country* et un *Destination Country*. Généralement, dans un contexte d'analyse, l'utilisateur a besoin de différents niveaux de détail pour une dimension donnée. Par exemple, il peut vouloir se renseigner sur l'origine (*Origin*) d'un produit *Product*. Cependant, le modèle conceptuel multidimensionnel devrait aussi pouvoir le renseigner sur le pays (*Country*), le département d'origine (*State*), le district (*District*), l'agence (*Office)* du produit (*Product*). Ceci nous amène à définir une notion de *hiérarchie* dans une dimension.

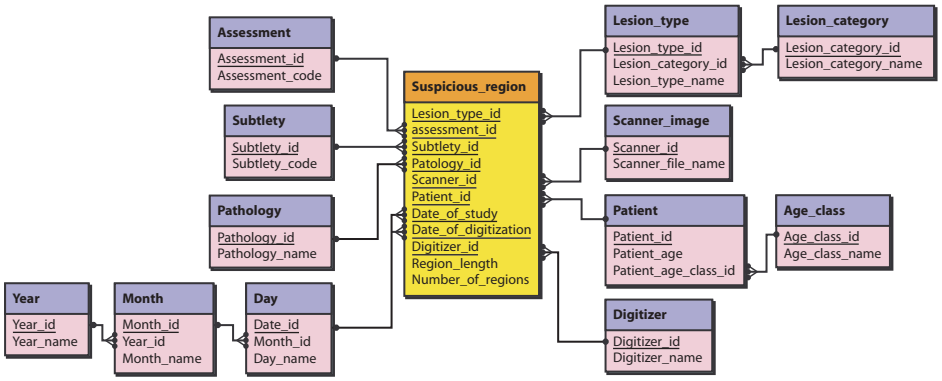


FIG. 3 – *modèle conceptual du cube de données des "Régions suspectées"*

**Definition 2** (*Hiérarchie d'une dimension*)
Soit $H$ un jeu de tables $l$ notées $H = \{D_1, \ldots, D_t, \ldots, D_l\}$. $H$ étant une *hiérarchie de dimensions* avec $l$ niveaux de détails, si $\forall t \in \{2, \ldots, l\}$ la clé primaire $D_t.PK$ de $D_t$ est un attribut (clé étrangère) dans $D_{t-1}$.

En d'autres termes, les $l$ tables de la hiérarchie sont liées par inclusion sémantique : $D_1 \subset \ldots \subset D_{t-1} \subset D_t \subset \ldots \subset D_l$. Dans la dimension hiérarchique de la figure 2 (b), un n-uplet de la table *Office* est sémantiquement inclus dans la table *District*. Définissons maintenant la *clé primaire* d'une dimension hiérarchisée d'une part, et la modélisation en *flocon de neige* d'autre part.

**Definition 3** (*Clé primaire d'une dimension hiérarchique*)
Soit $H = \{D_1, \ldots, D_t, \ldots, D_l\}$ une dimension hiérarchisée avec $l$ niveaux de détail où $D_1 \subset \ldots \subset D_{t-1} \subset D_t \subset \ldots \subset D_l$. La *clef primaires* de la dimension hiérarchisée $H$ notée $H.PK$ correspond à la clef primaire de la première table de $H$ ($H.PK = D_1.PK$).

**Definition 4** (*modèle en flocon de neige*)
Soit $\mathcal{H} = \{H_s, 1 \leq s \leq r\}$ un ensemble de $r$ hiérarchies indépendantes. Chaque hiérarchie $H_s$ a $H_s.PK$ comme clé primaire. La table de faits $F$ contient $d$ clés composites. Une *modèle en flocon de neige* est défini par le couple $(F, \mathcal{H})$ vérifiant les conditions suivantes :

- $\forall t \in \{1, \ldots, d\}$, il existe exactement un $s \in \{1, \ldots, r\}$ tel que $F.K_t = H_s.PK$ ;
- $\forall s \in \{1, \ldots, r\}$,, il existe un ou plusieurs $t \in \{1, \ldots, d\}$ tel que $F.K_t = H_s.PK$.

Un modèle en flocons de neige prend donc en compte les dimensions hiérarchisées. Par exemple, le model MCM de la figure 3 représente un cube des *régions suspectes* (tumeurs détectées sur des mammographies) organisé en flocon de neige.

## 4.2 Modélisation des cubes de données avec XML

Un document XML est une structure composée d'imbrications d'éléments, débutant par une racine. Chaque élément pouvant contenir des sous éléments et des attributs. Les attributs sont inclus, avec leurs valeurs respectives, entre les déclarations d'ouverture et de fermeture de l'élément (balises). Entre les balises d'ouverture et de fermeture, plusieurs sous éléments peuvent être insérés. Nous proposons une formalisation pour définir un MCM d'un cube XML. Pour cela, nous définissons la notion de *schéma en étoile XML* :

**Définition 5** (*Schéma en étoile XML*)
Soit $(F, \mathcal{D})$ un schéma en étoile, où $F$ est une table de faits ayant $m$ attributs de mesure $\{F.M_q, 1 \leq q \leq m\}$ et $\mathcal{D} = \{D_s, 1 \leq s \leq r\}$ un ensemble de $r$ tables de dimension où chaque $D_s$ contient un ensemble de $n_s$ attributs $\{D_s.A_i, 1 \leq i \leq n_s\}$. Le *Schéma en étoile XML* de $(F, \mathcal{D})$ est un schéma XML tel que :

- $F$ définit l'élément racine dans le schéma XML ;
- $\forall q \in \{1, \ldots, m\}$, $F.M_q$ définit un attribut XML inclus dans l'élément racine ;
- $\forall s \in \{1, \ldots, r\}$, $D_s$ est un sous éléments XML de l'élément racine XML. Il y a autant de sous éléments XML que de dimensions liées à la tables des faits $F$ ;
- $\forall s \in \{1, \ldots, r\}$ et $\forall i \in \{1, \ldots, n_s\}$, $D_s.A_i$ définit un attribut XML inclut dans l'élément XML $D_s$.

La figure 4 montre une partie du schéma XML équivalent au modèle conceptuel de la Figure 2(a). Nous proposons maintenant une solution pour écrire un modèle en flocons de neige en XML Schéma. Pour ce faire, nous définissons la notion de *dimension hiérarchisée XML* .

**Définition 6** (*Dimension Hiérarchisée XML*)
Soit $H = \{D_1, \ldots, D_t, \ldots, D_l\}$ une dimension hiérarchisée. La *dimension hiérarchisée XML* est une partie d'un schéma XML tel que :

- $D_1$ définit un élément XML ;

```
<xs:element name="F">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="D1" type="D1_Type" />
                  <xs:element name="D2" type="D2_Type" />
                  <xs:element name="D3" type="D3_Type" />
                  <xs:element name="D3" type="D3_Type" />
                  <xs:element name="D4" type="D4_Type" />
            </xs:sequence>
            <xs:attribute name="F.M1" type="xs:integer" />
            <xs:attribute name="F.M2" type="xs:integer" />
      </xs:complexType>
</xs:element>

<xs:complexType name="D1_Type">
      <xs:attribute name="D1.A1" type="xs:string" />
</xs:complexType>

<xs:complexType name="D2_Type">
      <xs:attribute name="D2.A1" type="xs:string" />
      <xs:attribute name="D2.A2" type="xs:string" />
</xs:complexType>

<xs:complexType name="D3_Type">
      <xs:attribute name="D3.A1" type="xs:string" />
      <xs:attribute name="D3.A2" type="xs:string" />
</xs:complexType>

<xs:complexType name="D4_Type">
      <xs:attribute name="D4.A1" type="xs:string" />
</xs:complexType>
```

FIG. 4 – *Un exemple d'un schéma en étoile XML*

– $\forall t \in \{2, \ldots, l\}$, $D_t$ t définit un sous élément XML de l'élément XML $D_{t-1}$ ;
– $\forall t \in \{1, \ldots, l\}$, chaque attribut dans $D_t$ définit un attribut XML inclus dans l'élément XML $D_t$.

**Définition 7** (*modèle en flocons de neige XML*)
Soit $(F, \mathcal{H})$, un modèle en flocons de neige où $F$ est une table de faits ayant $m$ mesures $\{F.M_q, 1 \leq q \leq m\}$ et $\mathcal{H} = \{H_s, 1 \leq s \leq r\}$ est un ensemble de $r$ hiérarchies indépendantes. Le *modèle en flocons de neige XML* de $(F, \mathcal{H})$ est un schéma XML tel que :

– $F$ définit l'élément XML racine du schéma XML ;
– $\forall q \in \{1, \ldots, m\}$, $F.M_q$ définit un sous élément XML inclut dans l'élément racine XML ;
– $\forall s \in \{1, \ldots, r\}$, $H_s$ définit autant de fois des dimensions hiérarchisées XML, comme des sous éléments de l'élément XML racine qu'elle est liée à la table de faits $F$.

En conclusion, en utilisant le typage des données et les relations exprimées par les sous éléments, les schéma XML permettent d'écrire un modèle logique d'un cube de données. Comme les document XML contiennent les valeurs des éléments et des attributs, nous en déduisons qu'ils contiennent l'information nécessaire pour définir des faits OLAP. Nous dirons qu'un document XML supporte un *fait XML* tel que :

**Definition 8** (*Un Fait XML*)
Un *fait XML* est un document XML valide conformément au *modèle en étoile* ou au *modèle XML en flocon de neige* correspondant au modèle conceptuel du cube de données (MCM).

La figure 5 montre un exemple d'un fait XML associé au MCM "Régions suspectes" présenté sur la figure 3.

# 5 Construction des cubes XML

Rappelons que dans notre approche le MCM est un référentiel qui exprime les objectifs d'analyse de l'utilisateur. Il doit être apparié avec les données complexes contenues dans des

```
<?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. 5 – *Exemple d'un fait XML*

documents XML afin de construire une base décisionnelle. Le MCM et les documents XML sources sont exprimés à l'aide de schémas XML puis transformés en arbres d'attributs pour être comparés. Des algorithmes d'appariement permettent grâce à des opérateurs de fusion par *élagage* ou par *greffe*, proposés par Golfarelli et al. dans Golfarelli et al. (1998); Golfarelli et Rizzi (1999), de traiter les arbres d'attributs pour générer le schéma XML du cube XML ainsi que les documents XML conformes Golfarelli et al. (2001a,b). Il est possible que certains documents XML sources risquent d'être rejetés. Pour cela, nous introduisons la notion du contenu minimal d'un document XML.

## 5.1 Fusion des arbres d'attributs

L'objectif de notre approche est de fusionner, à l'aide des opérations de *fusion*, deux arbres d'attributs correspondant au MCM et aux documents XML sources en un seul contenant les attributs communs et en conservant leurs relations de départ. La fusion par élagage (*pruning*) consiste à supprimer des parties des deux arbres. Dans l'exemple de la figure 7(a), seuls les sommets communs (cercles noirs) dans les deux arbres d'attributs de départ sont conservés dans l'arbre fusionné. Tous les autres sommets (cercles blancs) sont élagués avec les sous-arbres associés.

Quant à la fusion par greffe, elle s'effectue lorsque des sous-arbres communs n'ont pas la même structure de relations dans les deux arbres en entrée. Dans ce cas, certains sommets non communs (cercles blancs) sont éliminés et seuls les sommets identiques et leurs relations (cercles noirs) sont maintenus dans l'arbre fusionné. Lorsqu'un sommet est éliminé, ses descendants sont conservés dans l'arbre fusionné. Dans la figure 7(b), les sommets non communs x et y sont éliminés et leurs descendants respectifs sont greffés aux sommets maintenus dans l'arbre fusionné.

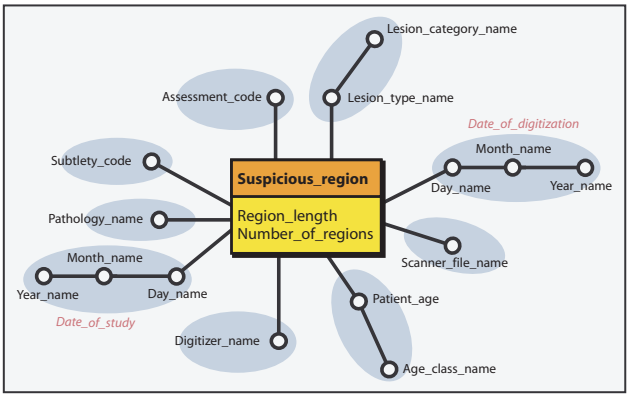


FIG. 6 – *Arbre d'attributs associé au MCM des "Régions suspectes"*

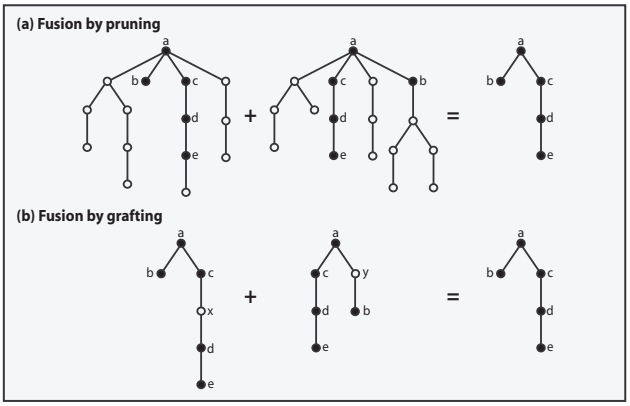


FIG. 7 – *Exemples de fusion de deux arbres d'attributs par élagage (a) et par greffe (b)*

## 5.2 Contenu minimal d'un document XML

Dans certains cas, lorsque le document XML en entrée ne contient pas assez d'information requise par le MCM, le résultat de la fusion peut s'avérer un document XML relativement démuni d'un point de vue informationnel. Il risque de représenter un fait OLAP avec trop de valeurs absentes. Il est donc non utile d'alimenter l'entrepôt avec de tels documents. Ainsi, il est important d'introduire un contrôle sur l'arbre d'attributs correspondant aux documents XML sources. Le but de ce contrôle est de vérifier si un document XML en entrée contient les informations requises par l'utilisateur ou non. Pour cela, nous introduisons la notion du *contenu minimal d'un document XML*.

Le contenu minimal d'un document XML est entièrement défini par l'utilisateur lorsqu'il soumet le modèle du cube pour exprimer ses objectifs d'analyse (MCM). Lors de la phase de saisie ou de chargement du MCM, l'utilisateur peut ainsi déclarer pour chaque élément

de celui-ci, s'il est obligatoire ou optionnel dans le cube final. Ces éléments portent sur les mesures, les dimensions ou les hiérarchies de dimensions et leurs attributs. Le contenu minimal d'un document XML corresponds donc à la partie obligatoire de l'arbre d'attributs associé au modèle du cube soumis. Rappelons que l'objectif de notre approche est de construire des cubes de données avec des documents XML sources pour faire des analyses en ligne (OLAP). Il n'est pas possible de décider dans un processus complètement automatisé si des éléments d'un documents XML sont optionnels ou non. C'est donc à l'utilisateur de définir le contenu minimal d'un document XML. Ceci dit, nous supposons que par défaut, toutes les mesures et tous les attributs des dimensions et des hiérarchies seront présents dans le cube final, c'est à dire sont obligatoires. Nous supposons également que les mesures ne peuvent pas être toutes optionnelles, au moins une mesure doit être obligatoire. Car un fait OLAP sans mesure n'a pas de sens et ne peut donc pas être exploité par des opérateurs OLAP tel l'agrégation. Lors de la phase de fusion des arbres d'attributs, l'arbre des documents XML en entrée est vérifié s'il contient tous les éléments obligatoires imposés par l'utilisateur. Il est alors fusionné avec l'arbre du MCM si ce contrôle est satisfaisant. Autrement, la fusion n'a pas lieu, l'arbre est rejeté et le document XML en sortie n'est pas généré.

# 6 Implémentation

Nous avons choisi *Java* pour implémenter notre approche dans l'optique d'une plate-forme Java dédiée à *X-Warehousing*. L'application contient deux modules : *de chargement* et *de fusion* (Figure 8).

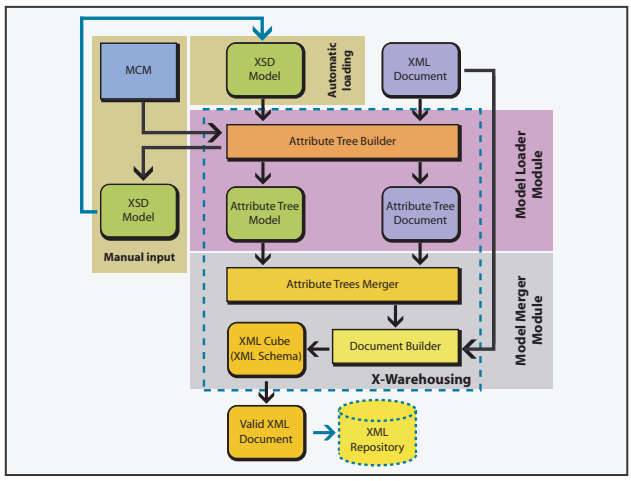


FIG. 8 – *Architecture de l'application*

## 6.1 Module de chargement

Pour définir un contexte d'analyse, l'utilisateur doit définir un modèle de cube comme référence. Il peut alors saisir manuellement un MCM (c.f. sous-section **??**) ; ou charger un fichier XSD contenant le schéma XML d'un MCM déjà existant (c.f. sous-section 4.2). Lorsque le MCM est saisi manuellement, il est automatiquement décrit en schéma XML, stocké dans un fichier XSD et ensuite transformé en arbre d'attributs. Lorsque l'utilisateur charge un fichier XSD, ce dernier est considéré comme un type document *JDOM*. Il est alors analysé par un algorithme de l'interface *JDOM API* et l'arbre d'attributs correspondant est construit. La figure 9 présente les fonctions recursives *WriteTreeDeep* et *ReadTreeDeep* que nous avons créées pour manipuler les arbres d'attributs. Le *module de chargement* charge respectivement le MCM et les documents XML en entrée. Ces derniers , le MCM et leurs arbres d'attributs sont tous décrits en schéma XML pour faciliter leur manipulation par les algorithmes de traitements. L'arbre d'attributs est un arbre orienté, acyclique et a un centre. Il est entièrement compatible avec la structure XML des documents en entrée.

```
Function WriteTreeDeep(document,tree)
    root=GetRootElement(document)
    nodeList=GetNodes(tree,root)
    While Not(end(nodeList))
        Graphe.AddVertex(nodeList.name)
        Call Function ReadTreeDeep(nodeList.name,tree)
    End While
End Function
```

```
Function ReadTreeDeep(root,tree)
    nodeList=GetNodes(tree,root)
    While Not(end(nodeList))
        Graphe.AddVertex(nodeList.name)
        Call Function ReadTreeDeep(nodeList.name,tree)
    End While
End Function
```

FIG. 9 – *Functions récursives WriteTreeDeep et ReadTreeDeep*

## 6.2 Module de fusion

Une fois le MCM et les documents XML en entrée sont chargés, l'application peut fonctionner selon deux modes : *automatique* ou *manuel*. Ces modes emploient le même noyau d'algorithmes avec une seule différence dans le cas du mode automatique. En effet dans ce denier cas, l'application extrait des documents XML partir d'une collection stockée dans un répertoire donné, les valident conformément au modèle de référence, et les stockent dans un répository XML et cela de façon automatique et itérative. Le *module de fusion* utilise les opérations de fusion par *élagage* et par *greffe* présentées dans la section 5. La figure 10 montre la fonction *MergeTree* que nous avons développée pour fusionner deux arbres d'attributs. Cette fonction lit les deux arbres respectifs du MCM et du document XML source et génère l'arbre fusionné. Celui-ci correspond au schéma du cube XML à partir duquel les documents XML valides générés vont alimenter le cube XML. Quand un sommet de l'arbre du MCM ne correspond pas à un nœud de l'arbre du document source, nous plaçons la valeur d'arc à zéro. Nous réécrivons ensuite l'arbre seulement avec les arcs non nuls.

```
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. 10 – *La fonction MergeTree*

# 7 Étude de cas

Pour illustrer notre approche *X-Warehousing*, nous l'avons appliquée sur un cas réel. Considérons le cube de données des "régions suspectes dans des mammographies" présenté dans la figure 3 en tant que MCM de référence. Nous disposons de 4 686 documents XML [2] en entrée que nous souhaitons entreposer. L'ensemble de ces documents en entrée ont la même structure et sont valides par rapport à un schéma XML. Par conséquent, ils ont le même arbre d'attributs indiqué dans la figure 11. L'application procède à la fusion des arbres d'attributs respectifs (Figures 11 et 6). Le résultat de la fusion dépend du *contenu minimal des documents XML* défini par l'utilisateur lors de la déclaration du MCM.

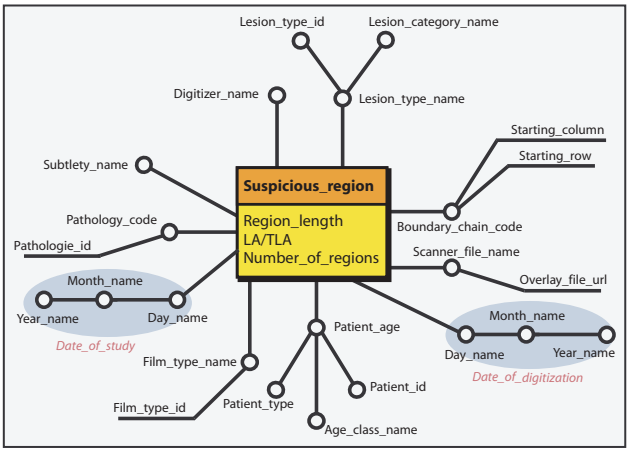


FIG. 11 – *Arbre d'attributs des documents XML en entrée*

Supposons que toutes les dimensions et les mesures soient marquées "obligatoires" sur le MCM par l'utilisateur. Lors de la fusion des arbres d'attributs du MCM et des documents XML en entrée, le *module de fusion* rejettera tous ces derniers. En effet, nous constatons que l'attribut *Assessment_code* est absent de l'arbre d'attributs des documents XML en entrée (figure11).

[2] http ://eric.univ-lyon2.fr/.../

Comme il est marqué "obligatoire" sur le MCM, aucun document XML en entrée ne sera apparié avec ce dernier et aucun document XML en sortie ne sea généré.

Supposons maintenant que l'utilisateur définit un *contenu minimal des documents XML* plus souple et qu'il considère que la dimension *Assessment* n'est pas pertinente pour son analyse. Il va donc la définir comme "optionnelle" sur son MCM de référence. Dans ce cas, l'absence de l'attribut *Assessment_code* dans les documents XML en entrée ne perturbera pas la fusion des arbres d'attributs correspondants et *le module de fusion* générera le modèle logique du cube XML sous la forme d'un schéma XML, comme le montre la figure 12.

De plus, pour chaque document en entrée, un *fait XML* (document XML) est créé. Comme tous les documents XML en entrée ont la même structure, tous les faits XML générés sont valides conformément au schéma XML de la figure 12. Notons également que tous les attributs non-concernés dans les arbres d'attributs (figures 11 et 6) sont élagués par le *module de fusion*. Par exemple, D'autres attributs (*Lesion_type_id*, *Boundary_chain_code* et *Patient_id*...) ne figureront pas dans le *cube XML*.

Finalement à travers cette étude de cas nous avons validé notre approche *X-Warehousing*. Elle permet de concevoir des entrepôts XML (ou des magasins XML) et de les alimenter avec des données complexes contenues dans des documents XML ; et cela conformément aux besoins d'analyses d'un utilisateur définis dans un MCM. Par conséquent XML peut être considéré comme une description logique d'un entrepôt de données complexes.

# 8 Conclusion et perspectives

Dans ce papier, nous avons proposé une méthodologie basée sur le formalisme XML pour entreposer des données complexes. Notre approche ne consiste pas seulement à alimenter une structure multidimensionnelle à l'aide de documents XML mais permet également d'exprimer un niveau d'abstraction intéressant pour préparer ces derniers à l'analyse. En effet, cela consiste à valider les documents XML selon une grammaire XML (schéma XML) représentant un modèle de cube de données. Nous avons défini une formalisation des *schémas en étoile ou en flocons de neige* en XML. Nous avons utilisé le concept *d'arbre d'attributs*, Golfarelli et al. (2001a,b), pour apparier des besoins d'analyse exprimés par un modèle conceptuel multidimensionnel et des documents XML et cela dans le but d'entreposer des données complexes. Pour valider notre approche, nous avons développé une application Java qui charge en entrée un MCM et des documents XML. Elle produit en sortie un modèle logique et un modèle physique pour un cube composé de documents XML homogènes correspondant chacun à un fait OLAP que nous désignons par *cube XML*. Une étude de cas sur les régions suspectes sur des mammographies a montré l'intérêt de notre approche sur des applications réelles pour concevoir et construire des entrepôts de données complexes en utilisant XML.

Plusieurs perspectives semblent se dessiner pour notre approche *X-Warehousing*. La première porte sur l'interrogation du cube XML. Une extension du langage XQuery est nécessaire pour permettre de réaliser l'opération du *Group-by*. Dans l'exemple de notre étude de cas, les mesures étaient numériques, il n'est pas difficile de les agréger à l'aide d'opérateurs classiques quoique dans notre cas nous avons opté pour une moyenne statistique plus élaborée que la moyenne empirique. Cependant, dans le cas où les mesures ne seraient pas numériques, le recours à des opérateurs appropriés est requis. L'exemple de l'opérateur *OpAC*, Messaoud et al. (2004), peut s'avérer fort utile. D'autres perspectives sont inscrites dans la continuation de nos

travaux. Une étude de performance des requêtes OLAP dans le cadre de cube XML pemettra de valider cette approche de ce point de vue. La fiabilité de l'application *X-Warehousing* passe par une étude de complexité et des temps de réponse des chargements des documents XML en entrée, de la construction et de la fusion des arbres d'attributs, de la création des documents XML en sortie. Une autre perspective concerne le problème de mise à jour du *cube XML* lors des changements dans les données sources, Pedersen et Pedersen (2003); Yin et Pedersen (2004), ou lorsqu'un utilisateur désire modifier le MCM de référence pour prendre en compte une évolution de ses objectifs d'analyse.

## Références

Baril, X. et Z. Bellahsène (2000). A View Model for XML Documents. In *OOIS*, pp. 429–441.

Baril, X. et Z. Bellahsène (2003). *XML Data Management : Native XML and XML-Enabled Database Systems* (First ed.)., Chapter Designing and Managing an XML Warehouse, pp. 455–474. Addison Wesley Professional.

Chaudhuri, S. et U. Dayal (1997). An overview of data warehousing and olap technology. *SIGMOD Record 26*(1), 65–74.

Golfarelli, M., D. Maio, et S. Rizzi (1998). 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*, Washington, DC, USA, pp. 334. IEEE Computer Society.

Golfarelli, M. et S. Rizzi (1999). Designing the data warehouse : key steps and crucial issues. *Journal of Computer Science and Information Management 2*(3), 88–100.

Golfarelli, M., S. Rizzi, et B. Vrdoljak (2001a). Data Warehouse Design from XML Sources. In *DOLAP'01 : Proc. of the 4th ACM int. workshop on Data warehousing and OLAP*.

Golfarelli, M., S. Rizzi, et B. Vrdoljak (2001b). Integrating XML Sources into a Data Warehouse Environment. In *Proc. of IEEE Int. Conf. on Soft., Telecomm. and Computer Networks (SoftCOM 2001)*, pp. 49–56.

Hümmer, W., A. Bauer, et G. Harde (2003). XCube : XML for data warehouses. In *DOLAP 2003, ACM Sixth International Workshop on Data Warehousing and OLAP, New Orleans, Louisiana, USA, November 7, 2003,Proceedings*, pp. 33–40. ACM.

Inmon, W. (2005). *Building the Data Warehouse, Fourth Edition*. Wiley and Sons.

Kimball, R. (1996). *The data warehouse toolkit*. Wiley.

Kimball, R. et M. Ross (2002). *The Data Warehouse Toolkit*. John Wiley and Sons.

Krill, P. (1998). XML builds momentun as repository standard. *InfoWorld 20*(25), 6.

Messaoud, R. B., O. Boussaid, et S. Rabaséda (2004). A new olap aggregation based on the ahc technique. In *DOLAP*, pp. 65–72.

Nassis, V., R. Rajugan, T. S. Dillon, et J. Rahayu (2004). Conceptual Design of XML Document Warehouses. In Y. Kambayashi, M. Mohania, et W. Wöß (Eds.), *DaWaK*, Volume 3181 of *Lecture Notes in Computer Science*, pp. 1–14. Springer.

Park, B.-K., H. Han, et I.-Y. Song (2005). Xml-olap : A multidimensional analysis framework for xml warehouses. In *DaWaK*, pp. 32–42.

Pedersen, D. et T. B. Pedersen (2003). Achieving adaptivity for olap-xml federations. In *DOLAP 2003, ACM Sixth International Workshop on Data Warehousing and OLAP, New Orleans, Louisiana, USA, November 7, 2003, Proceedings*, pp. 25–32. ACM.

Pokorný, J. (2001). Modelling stars using XML. In *DOLAP'01 : Proc. of the 4th ACM int. workshop on Data warehousing and OLAP*, pp. 24–31.

Rajugan, R., E. Chang, T. Dillon, et L. Feng (2003). XML Views : Part 1. In V. Marík, W. Retschitzegger, et O. Stepánková (Eds.), *DEXA*, Volume 2736 of *Lecture Notes in Computer Science*, pp. 148–159. Springer.

Rusu, L. I., J. W. Rahayu, et D. Taniar (2004). On building xml data warehouses. In *IDEAL*, pp. 293–299.

Trujillo, J., S. Luján-Mora, et I. Song (2004). Applying UML and XML for Designing and Interchanging Information for Data Warehouses and OLAP Applications. *Journal of Database Management 15*(1), 41–72.

Yin, X. et T. B. Pedersen (2004). Evaluating xml-extended olap queries based on a physical algebra. In *DOLAP*, pp. 73–82.

## Summary

Complex data proliferate more and more and entreprises are concerned by this kind of data that are neither numeric nor symbolic. These complex data are coming from different and heterogeneous sources. Nowadays, data are more and more supported on XML documents. To efficiently explore the complex data it is necessary to integrate them into a decision support system. In this paper, we propose to warehouse complex data supported on XML documents. Our approach, called *X-Warehousing*, is a methodology to design and to build an XML warehouse using the XML formalism. To validate it, a java application was implemented and a case study of complex data taken from the breast cancer domain was achieved.

```
< ?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. 12 – *Logical model of the “Suspicious Region” XML Cube*