Un formalisme pour l'intégration de données hétérogènes

Sana Hamdoun*, Faouzi Boufares*

*Institut Galilée, Université Paris Nord Av. J. B. Clément 93430 Villetaneuse France
Sh@lipn.univ-paris13.fr
Boufares@lipn.univ-paris13.fr

Résumé. Dans ce papier nous proposons, un formalisme d'intégration de données hétérogènes. Nous définissons, d'une manière générale, une source de données comme un ensemble de composants muni des relations et fonctions qui relient ces composants, et un environnement d'intégration comme un ensemble de sources associé à un ensemble de ″liens d'intégration″ entre ces dernières.
L'approche générale d'intégration que nous proposons s'inscrit dans le cadre de la construction d'entrepôts de données hétérogènes basés sur des sources de catégories différentes, structurées, semi-structurées et non structurées telles que relationnelles, objet-relationnelles et XML. Le processus d'intégration est composé de trois étapes : le filtrage de l'ensemble des composants de l'entrepôt, la génération de son schéma global et la construction des vues qui le composent.

1 Introduction

Très généralement parlant, une base de données (BD) est une grande quantité d'informations stockées sur support informatique de telle sorte que son exploitation (mise-à-jour, recherche, extraction d'information, analyse et fouille de données (Benabdeslem et al., 2001) soit facilitée.

Dans une BD les ″éléments″ d'information sont reliés logiquement et physiquement. Ces éléments et liens sont organisés selon un modèle de données. Une *donnée* est une sorte de *composition* des éléments d'information via les liens envisagés, et dans ce cas *une BD est un ensemble de données*.

Dans ce papier, nous faisons l'abstraction des concepts de données et de bases de données, aussi bien au niveau logique qu'au niveau sémantique, et ceci indépendamment de tout processus de modélisation.

Une BD sera vue comme un ensemble de ″composants″ muni des liens pour les relier.

Sémantiquement, les données et leurs liens seront interprétés dans un *système de types*. Bien que ce dernier puisse dépendre du modèle envisagé, nous faisons aussi l'abstraction d'un tel système pour BDs, indépendamment du modèle.

Gérer plusieurs BDs revient donc à gérer plusieurs ensembles et plusieurs relations (au sens général du terme) sur ces derniers.

L'*intégration* des données est l'ensemble des transformations que subissent certaines données et leurs organisations afin d'être fusionnées et présentées éventuellement sous une nouvelle forme (entre-

pôt de données, bases de données temporelles, versions…). On parle d'intégration quand les données proviennent des BDs différentes (éventuellement avec des modélisations différentes). Dans ce cas, chaque BD est appelée une *source de données (SD)*. Fusionner les sources nécessite d'envisager aussi des liens entre les données provenant de ces dernières. De tels liens peuvent permettre, par exemple au niveau logique, d'identifier deux données ou négliger une donnée en faveur d'une autre ″mieux définie″. Ces sources et ces liens forment un *environnement d'intégration*. Donc, un environnement d'intégration est un ensemble de SDs munies de liens entre les données provenant de ces sources. Le processus d'intégration, proprement, dit devient alors une fonction qui prend en entrée un environnement d'intégration et rend un nouvel ensemble de données tel qu'un *entrepôt de données* (ED) ou une BD temporelle. Notre démarche n'est pas applicable exclusivement pour la construction d' EDs. Elle peut être utilisée pour d'autres environnements d'intégration.

Un *ED* est une collection de données orientées sujet, intégrées, non volatiles et variant dans le temps organisées pour le support d'un processus d'aide à la décision (Inmon, 1995). L'environnement informationnel actuel se caractérise par des données fortement distribuées. Ces dernières, surabondantes, sont généralement éparpillées et hétérogènes. En effet, avec l'apparition d'Internet et le développement des différentes représentations et formats des documents, les données peuvent être de *plusieurs catégories* : *structurées* (données relationnelles et objet), *semi-structurées* (HTML, XML et graphes) ou même *non structurées* (texte, images et son).

L'hétérogénéité n'est pas un problème récent (Pontieri et al., 2003). Cependant, dans la littérature, il n'existe pas de consensus sur la signification de l'hétérogénéité. En effet, selon le domaine et le type d'application, le traitement et l'interprétation de l'hétérogénéité ont été fait de plusieurs façons. Dans certains travaux, l'hétérogénéité des données réside dans l'utilisation de sources contenant des données différentes alors qu'elles sont de même catégorie et de même modélisation (Da Silva et al., 2002). D'autres travaux évoquent l'hétérogénéité quand il s'agit de l'utilisation de données de même catégorie mais avec des modélisations différentes (Saccol et Heuser, 2002). La différence des modèles a un impact sur la représentation et les types des données ce qui les rend hétérogènes. D'autres auteurs présentent l'hétérogénéité dans le fait que les données appartiennent à des catégories différentes (Kim et Park, 2003) (Beneventano et al., 2002). Ainsi, nous classons les données en trois catégories (structurées, semi-structurées et non-structurées). L'utilisation de données de structures différentes revient nécessairement à traiter l'hétérogénéité.

Vu les différentes interprétations de l'hétérogénéité, nous adoptons, dans le cadre de notre travail d'intégration, la définition ci-dessous. Ce qui va nous permettre de traiter toutes les catégories à la fois :

Des *sources de données* sont dites *hétérogènes* si elles vérifient l'une des deux propriétés suivantes : 1) Elles appartiennent à la *même catégorie* de données mais elles ont des *modélisations différentes* ; 2) Elles appartiennent à des *catégories de données différentes*. Ainsi le traitement d'une BD relationnelle et d'une BD objet-relationnelle revient à gérer des sources de données hétérogènes. Il en est de même pour une BD relationnelle et une BD XML.

De nos jours, les applications complexes telles que l'extraction de connaissances, les fouilles de données, l'apprentissage (Bennani, 2006) et les applications web utilisent des données hétérogènes et distribuées. Dans un tel contexte, le besoin d'intégration se fait de plus en plus ressentir. Cependant, pour répondre à ce besoin, le développement des applications d'intégration se voit contraint de composer avec la répartition des sources et leur hétérogénéité (Structures et Données). L'utilisation de données non structurées ou semi-structurées telles que XML dans le cadre de ces applications rend également leur maintenance et leur interrogation non évidente (Wang et al.,2006) (Prakash et al., 2006).

Dans ce papier, nous nous intéressons au domaine d'*intégration de données* dans le but de construire des entrepôts dont les sources sont totalement hétérogènes. Il est structuré comme suit. Le paragraphe deux traite les BD classiques (XML, relationnelles, et objet-relationnelles) revisitées ainsi qu'une proposition d'une modélisation générale de ces bases. L'environnement d'intégration ainsi que les différents liens sont présentés dans la section trois. Les étapes du processus d'intégration proposé sont présentées dans le paragraphe quatre. Nos travaux futurs sont donnés en guise de conclusion.

2 Modélisation formelle d'une source de données

Afin d'uniformiser la modélisation des SDs hétérogènes, nous présentons dans cette section, dans un premier temps, les concepts de système de types et de source de données. Dans un deuxième temps, nous revisitons les modélisations des BDs structurées et semi-structurées. Les données non structurées sont gérées dans des BDs relationnelles étendues avec les types complexes tels que BLOB et CLOB.

2.1 Système de types pour BD

Dans un système de types, les types sont construits à partir des types de base (prédéfinis) et de constructeurs de types. Un constructeur de type prend une séquence de types et construit un nouveau type. Donc, d'une manière générale, les types sont construits par la grammaire ci-dessous :

$TYPES ::= BASE \mid CONST\ TYPES^*$ où $BASE$ est l'ensemble des types prédéfinis, $CONST$ est l'ensemble de constructeurs et $TYPES^*$ est l'ensemble de séquences de $TYPES$.

Exemple 1 (Système de types)

BASE ::=integer | char(n) | varchar(n) | BLOB | CLOB |...
CONST ::=set | list | tuple | ...

Dans un système de types pour les BDs, les valeurs des types de base ont par exemple un certain format et les formats peuvent être étendus d'une manière naturelle. Par exemple, si **`A:char(5)`** et **`B:char(10)`**, les A-valeurs peuvent être utilisées comme B-valeurs. Ceci ressemble à une relation de réécriture. Selon les constructeurs retenus, cette relation peut être étendue aux types construits de même structure.

Utilisons la notation $t1 \rightarrow t2$ pour dire que tout x : t1 peut être vue comme x : t2.

Une telle relation parmi les types est transitive. Toutefois, il ne faut pas voir cette relation comme une relation d'héritage. Inspirés de la théorie de réécriture, nous notons par $\rightarrow^*$ la fermeture transitive de $\rightarrow$.

Nous supposons que les types d'un Système de Gestion de BD (SGBD) sont munis d'une telle relation et vérifient la propriété suivante, dite de *confluence :* Si $t_1 \rightarrow^* t_2$ et $t_1 \rightarrow^* t_3$ alors il existe t_4 tels que $t_2 \rightarrow^* t_4$ et $t_3 \rightarrow^* t_4$. Dans un tel système, définissons la compatibilité de la manière suivante :

Définition 1 (Compatibilité) : Deux types t_1 et t_2 sont compatibles s'il existe un type t_3 tel que $t_1 \rightarrow^* t_3$ et $t_2 \rightarrow^* t_3$■

Dans un système satisfaisant la propriété de confluence, la compatibilité est une relation d'équivalence que nous notons par $\approx$. La preuve est donnée dans (Hamdoun, 2006). Dans la suite, nous supposerons que les systèmes de types envisagés vérifient la propriété de confluence.

2.2 Sources de données

Rappelons d'abord que si X est un ensemble. |X| désigne sa cardinalité et $\mathcal{P}(X)$ l'ensemble de ses parties. Les ensembles envisagés seront des ensembles finis, donc |X| est un entier naturel et $\mathcal{P}(X)$ est aussi fini.

D'une manière générale, nous appelons *source de données* un ensemble accompagné d'une ou plusieurs relations. Toutefois, dans ce papier, nous nous restreignons dans un cadre plus restreint et nous modélisons une source de données comme suit :

Définition2 (Source de données) : *Nous appelons source de données un triplet BD = (C, ref, comp) où :*

C est un ensemble,
ref : $\mathcal{P}(C) \rightarrow \mathcal{P}(C)$ est une relation binaire tel que $X\ ref\ Y \Rightarrow |X| = |Y|$,
comp : $C \rightarrow \mathcal{P}(C)$ est une fonction telle que $c \notin comp(c)$ pour tout $c \in C$ ■

Un élément de C sera appelé un *composant* de BD. Un composant c est dit simple si comp(c)=∅. Toute source de données est accompagnée d'un système de types dans lequel elle est interprétée. Chaque composant simple est interprété par un type de base. Les constructeurs de types doivent permettre d'interpréter tous les composants.

Revisitons ci-dessous les modèles relationnel, objet-relationnel et XML afin de démontrer que ces derniers peuvent être modélisés de la même manière indépendamment du modèle et de la catégorie des données

2.3 Modélisation d'une SD structurée (Modèles relationnel et objet-relationnel revisités)

Dans la suite de notre travail nous désignerons par BD relationnelle-étendue soit une BD relationnelles soit une BD objet-relationnelles. La définition d'une BD se fait en prenant en compte la description de la structure des objets du monde réel que l'on souhaite stocker (schéma de la BD) et leur définition (instance de la BD).

Soit REL, ATT et DOM trois ensembles disjoints.

- Un *schéma* de base de données relationnelle-étendue sur (REL, ATT, DOM) est un triplet (Voir Hamdoun, 2006) :

$S=(R_s, ATT_s, DOM_s)$ accompagné de trois fonctions :

$dom_s : ATT_s \longrightarrow DOM$

$att_s : R_s \longrightarrow \mathcal{P}_f(ATT)\backslash\{\emptyset\}$

$[[\ ]]_s : DOM_s \longrightarrow TYPES$

où TYPES désigne les différents types générés par un système de types préalablement envisagé, et $ATT_S=\cup_{R\in R_s} att_s(R)$.

- Une *instance* de S est une fonction δ_s :

$\delta_s : R_s \longrightarrow \cup_{R\in Rs}(Inst_s(R))$

telle que $\delta_s(R) \in Inst_s(R)$ pour tout $R \in R_s$, où $Inst_s(R)=\mathcal{P}_f(dom_s(R))$ et $dom_s(R)= \prod_{A\in att_S(R)}([[dom_s(A)]]_s)$.

- Une *base de données* est un triplet (S, K_s, δ_s) où S est un schéma, K_s est un ensemble de contraintes et δ_s est une *instance* de S satisfaisant les *contraintes*. L'état δ_s s'appelle l'état courant.

Une base relationnelle-étendue, peut ainsi être vue comme une source de données (Cf Définition 2).

Définition 3 (Source de données relationnelles étendues) : Une source de données relationnelles étendues est alors un triplet *BD = (C, ref, comp)* avec

+ L'ensemble de composants **C** = $ATT_s \cup R_s$. Un composant c de C a donc l'une des deux formes suivantes. Soit c=DB.R.A ou bien c=DB.R avec $R\in R_s$ et $A\in ATT_s$.

+ Le lien ***ref***, permet d'identifier les références externes dans la BD, il est défini comme suit :

X ref Y $\Leftrightarrow X \subset ATT_s$, $Y\subset ATT_s$ et X est une référence externe de Y.

+La fonction ***comp*** est définie de la manière suivante : $comp : ATT_s\cup R_s \rightarrow \mathcal{P}_f(ATT_s\cup R_s)$

$comp(X)=Y \Leftrightarrow X \in R_s$, $Y \subset ATT_s$ et $Y = att_s(X)$

ou bien, $X \in ATT_s$, X est un attribut composé, et $Y \subset ATT_s$, Y est l'ensemble des attributs qui compsent X ■

2.4 Modélisation d'une SD semi-structurée (Modèle XML revisité)

La définition d'une BD XML se fait aussi en deux étapes. La description des objets du monde réel (schéma de la BD) et la définition de ces derniers (instance de la BD). La description du schéma de la BD peut se faire soit à l'aide d'une DTD ou bien à l'aide d'un schéma XML (Cluet et Siméon, 2000).
Soit ELEM et DOM deux ensembles disjoints.

- Un *schéma* de BD XML sur (ELEM, DOM) est un couple **S=(E_S, DOM_S)** accompagné de trois fonctions (Voir Hamdoun, 2006) :

  dom_s : $E_S^s \longrightarrow$ DOM
  att_s : $E_s \longrightarrow \mathcal{P}_f(E_s)$
  $[[\ ]]_s$: $DOM_s \longrightarrow$ TYPES

  où TYPES désigne les types générés par un système de types auparavant envisagé.
- Une instance de S est une fonction $\delta_s : E_S^s \longrightarrow \cup_{R\in Rs}([[dom_s(E)]]_s)$ telle que $\delta_s(E) \in [[dom_s(E)]]_s$ pour tout $E \in E_S^s$.
- Une *base de données* est un triplet (S, K_s, δ_s) où S est un schéma, K_s est un ensemble de contraintes et δ_s est une instance de S satisfaisant les contraintes. L'état δ_s est appelé état courant.

Dans la vision du modèle XML présentée ci-dessus, nous n'avons pas tenu en compte la notion d'attributs des éléments XML. Nous considérons que les attributs d'un élément peuvent être représentés comme étant des sous-élements de celui-ci. Nous n'avons pas également considéré les entités car elles n'interviennent pas dans la structure du document XML.
Une base XML peut ainsi être vue comme source de données (Cf Définition 2).

Définition 4 (Source de données XML) : Une source de données XML est alors un triplet *BD=(C, ref, comp)* avec :

+ L'ensemble de composants $C = E_s$
+ Le lien ***ref***, permet d'identifier les références externes dans la BD. Il est défini comme suit : X ref Y $\Leftrightarrow X \subset E_S^s$, $Y \subset E_S^s$ et X est une référence externe de Y.
+ La fonction ***comp*** est la fonction att_s■

Remarque 1
Cette formalisation montre que les modèles des sources n'interviennent pas. En effet, nous avons démontré que toute source de données d'un des types étudiés peut être vue comme un triplet *BD = (C, ref, comp)* quel que soit le modèle sous-jacent (cf définitions 2, 3 et 4). Ainsi, les sources peuvent être totalement hétérogènes, structurées, semi-structurées et éventuellement non structurées.

3 Environnement et liens d'intégration

D'une manière générale, nous définissons un environnement d'intégration de la manière suivante.

Définition 5 (Environnement d'intégration, définition informelle**)** : Un environnement d'intégration est défini par *un ensemble de sources de données* et *un ensemble de "liens d'intégration"* entre les composants simples de toutes ces sources ■

Ces liens vont permettre de définir le processus d'intégration. Insistons sur le fait que la définition, que nous avons faite, d'un environnement d'intégration est indépendante de toute modélisation des sources de données. Ces dernières peuvent alors être de catégories différentes et donc hétérogènes.

3.1 Liens d'intégration

Nous proposons deux catégories de liens, *un ensemble SS de relations d'équivalence et un ensemble II de relations d'ordre strict*. Ces liens permettent de comparer les composants en faisant abstraction de tous les critères liés aux schémas des sources, à leurs contraintes, aux données, ainsi que toute autre information auxiliaire. Ils permettent également de couvrir la plupart des liens qui peuvent exister naturellement entre composants (Doan et al., 2002) (Serafini et al., 2003) (Xu et Embley, 2003) (Nassis et al., 2004) (Magnani et al., 2005) (Zerdazi et Lamolle, 2006). Dans notre approche, nous n'avons considéré qu'un seul lien S d'équivalence (en faisant allusion à la synonymie) et un seul lien I d'ordre strict (rappelant l'inclusion). Ces liens, que nous définissons ci-dessous, sont utilisés dans les différentes étapes du processus d'extraction et d'intégration des données.

Définition 6 (Liens d'intégration) : l'ensemble des liens d'intégration est formé d'*une relation S d'équivalence sur l'ensemble* $C = \cup_{k \in D}(C_k)$ *et d'une relation I d'ordre stricte sur C* ■

Afin d'intégrer des sources hétérogènes, nous avons à préciser les liens d'intégration. Ceci consiste donc à définir une relation d'équivalence et une relation d'ordre sur l'ensemble $C = (C_1 \cup C_2)$. Dans la suite, nous allons décrire ces deux relations dans le cadre de l'application aux données structurées et semi-structurées. Elles seront appelées **SYN***onymie* et **INC***lusion* et seront notées par **SYN** et **INC**, respectivement.

Etant données deux BDs définies par : $BD_i = (C_i, ref_i, comp_i)$, i=1,2. En pratique, c'est l'expert (l'administrateur) qui définit ces relations ou tout au moins un « noyau » de ces relations. Nous faisons l'abstraction de ce travail d'expert en considérant deux parties SY et IN de C×C vérifiant un certain nombre de propriétés de « bons sens ».

Plus précisément, considérons les « noyaux » suivants :

- Soit **SY** $\subset$ C×C tel que pour tout $(c_1, c_2) \in$ **SY,** $c_1 \in C_1$, $c_2 \in C_2$ sont des composants simples, $dom_{s_1}(c_1) \approx dom_{s_2}(c_2)$, et l'ensemble des contraintes de domaine de c_1 est logiquement équivalent à l'ensemble des contraintes de domaine de c_2.
- Soit **IN** $\subset$ C×C tel que pour tout $(c_1, c_2) \in$ **IN,** $c_1 \in C_1$, $c_2 \in C_2$ sont des composants simples, $dom_{s_1}(c_1) \approx dom_{s_2}(c_2)$, et toute contrainte de l'ensemble des contraintes de domaine de c_2 est impliquée par l'ensemble des contraintes de domaine de c_1.

Etant donnés ces noyaux, donnons, ci-dessous, les deux définitions de synonymie et d'inclusion.

Définition7 (Synonymie) : ***SYN*** *est la fermeture de* ***SY*** *construite récursivement par les règles ci-dessous :*

- pour tout $c \in C$, c ***SYN*** c *et si* c_1 ***SY*** c_2 *alors* c_1 ***SYN*** c_2 *(base de la récursion)*
- si c_1 ***SYN*** c_2 *alors* c_2 ***SYN*** c_1 *(symétrie),*
- si c_1 ***SYN*** c_2*, et* c_2 ***SYN*** c_3 *alors* c_1 ***SYN*** c_3 *(transitivité),*
- si $c_1 \in C_1$ *et* $c_2 \in C_2$ *sont composés et* $|comp_1(c_1)| = |comp_2(c_2)|$ *et il existe une bijection* $b : comp_1(c_1) \rightarrow comp_2(c_2)$ *telle que* c ***SYN*** $b(c)$ *pour tout* $c \in comp_1(c_1)$*, alors* c_1 ***SYN*** c_2 *,*
- ***SYN*** *est construite seulement par les clauses ci-dessus* ■

La relation **SYN** définie ci-dessus est une relation d'Equivalence (Voir Hamdoun, 2006).

Définition 8 (Inclusion) : ***INC*** *est la fermeture de* ***IN*** *construite récursivement par les règles ci-dessous :*

*- pour tout $c \in C$, $\neg (c$ **INC** $c)$ et si c_1 **IN** c_2 alors c_1 **INC** c_2 (base de la récursion)*
*- si c_1 **INC** c_2, alors $\neg (c_2$ **INC** $c_1)$ (antisymétrie),*
*- si c_1 **INC** c_2, et c_2 **INC** c_3 alors c_1 **INC** c_3 (transitivité),*
*- si $c_1 \in C_1$ et $c_2 \in C_2$ sont composés et $|comp_1(c_1)| \geq |comp_2(c_2)|$ et il existe $P_1 \subset \mathcal{P}(comp_1(c_1))$ et une bijection $b : P_1 \rightarrow comp_2(c_2)$ telle que c **INC** $b(c)$ pour tout $c \in P_1$, alors c_1 **INC** c_2 ,*
*- **INC** est construite seulement par les clauses ci-dessus* ■

La relation **INC** définie ci-dessus est une relation d'ordre strict (Voir Hamdoun, 2006).

En se basant sur les définitions ci-dessus, on pourra présenter formellement un environnement d'intégration.

3.2 Environnement d'intégration

Ayant défini informellement un environnement d'intégration comme *un ensemble de sources de données* et *un ensemble de "liens d'intégration"* entre les composants simples de ces sources. Nous donnons dans la suite sa définition formelle.

Définition 9 (Environnement d'intégration, définition formelle**)** : Un environnement d'intégration est un triplet (E,S,I) où $E = \{BD_k, k \in D\}$ est un ensemble de sources de données où $BD_k = (C_k, ref_k, comp_k)$ et D est un ensemble fini dit d'indexage, S est une relation d'équivalence sur l'ensemble $C = \cup_{k \in D}(C_k)$ et I est une relation d'ordre stricte sur C ■

Notons que cette définition, comme celles précédemment citées, est indépendante du modèle de chaque source. Par conséquent, ces sources peuvent être hétérogènes (modélisées différemment) (cf figure 1).

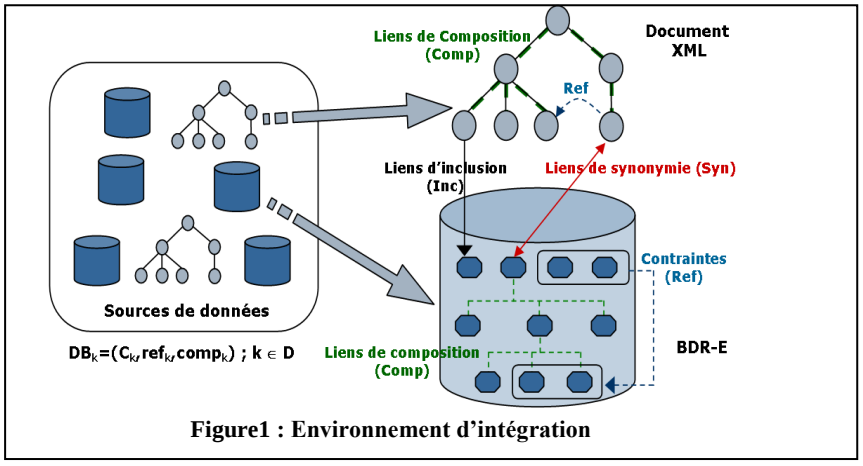

Figure1 : Environnement d'intégration

Remarque 2

Chaque BD_k désignant une source différente, nous pouvons supposer que les différents BD_k n'ont pas d'ingrédients communs. Autrement dit, il n'y a pas de composants de même nom dans deux sources différentes. En fait, $k \in D$ peut être vu comme un nom pour la source BD_k et par conséquent k simule le préfixage des composants de BD_k par le nom de la source.

Exemple 1 (Environnement d'intégration, Exemple Formel)

Un exemple d'environnement d'intégration peut être le triplet (**E, S, I**) tels que :

- $E=\{BD_1, BD_2\}$ avec $BD_1=(C_1, ref_1, comp_1)$ et $BD_2=(C_2, ref_2, comp_2)$.
- $C_1=\{x_1, x_2, x_3, x_4, x_5, x_6\}$, $ref_1 = \emptyset$ et $comp_1(x_1)=\{x_2,x_3\}$, $comp_1(x_4)=\{x_5,x_6\}$ et $comp_1(x) =\emptyset$ pour tout $x\notin\{x_1,x_4\}$
- $C_2=\{y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}\}$, $\{y_3\}ref_2\{y_1\}$ et $\{y_3\}ref_2\{y_8\}$ et $comp_2(y_2)=\{y_4,y_3\}$, $comp_2(y_5)=\{y_1,y_6\}$, $comp_2(y_7)=\{y_8,y_9, y_{10}\}$ et $comp_2(y) = \emptyset$ pour tout $y\notin\{ y_2, y_5, y_7\}$
- x_2 **S** y_3 et y_3 **S** x_2 et x **S** x pour tout $x \in C_1\cup C_2$
- x_3 **I** y_4

Cet exemple montre que les modèles des sources n'interviennent pas.

4 Processus d'intégration de données hétérogènes

Notre approche d'intégration prend en entrée un environnement d'intégration (contenant un ensemble de SDs hétérogènes de données) et rend, en sortie par exemple, un ED. Ce dernier est constitué d'un ensemble de vues sur les données sources. Deux architectures sont possibles grâce au méta-modèle présenté dans (Badri, 2008). Une architecture classique telles que les modèles en étoiles ou en flocons et une architecture originale plate.

Soit **(E, S, I)** un **environnement d'intégration** où $E =\{BD_k, k\in D\}$ et chaque $BD_k=(C_k, ref_k, comp_k)$ est une SD. Notre approche d'intégration est composée principalement de trois étapes : le filtrage des composants de l'ED, la génération du schéma global et la construction des vues.

4.1 Filtrage des composants de l'ED

Considérons l'ensemble **L** des *composant*s sélectionnés. Cette étape permet de filtrer l'ensemble L en appliquant tout d'abord une opération de décomposition suivie d'une opération d'élimination des composants inutiles. L constitue l'ensemble des composants des sources de données qui doivent figurer dans l'entrepôt. Le choix de **L** peut être fait par des critères prenant en compte la nature des données ou les objectifs de l'intégration. Nous faisons l'abstraction de ce choix en supposant que l'ensemble des composants de l'entrepôt est un sous-ensemble de tous les composants des différentes sources de données. Autrement dit : $L\subseteq\cup_{k\in D}(C_k)$ avec $L=\cup_{k\in D}(L_k)$ où $L_k=L \cap (C_k)$. On supposera que L n'est pas vide mais cela n'empêche pas que certains des L_k soient vides.

Les C_k étant disjoints il en est de même pour les L_k. Donc pour tout $x \in L$ il existe un unique $k\in D$ tel que $x\in L_k$. Dans la suite, nous notons par skl la fonction suivante : $skl : L \longrightarrow D$ $\forall x \in L$, $skl(x)=k \Leftrightarrow x \in L_k$. Cette fonction permet d'identifier la source correspondant à un composant de l'entrepôt.

Tous les composants non simples de L doivent être remplacés par les ″sous-composants″ qui les constituent. Nous avons défini comme composant simple tout composant c tel que $comp(c)=\emptyset$. Il s'agit donc, dans cette étape, de remplacer récursivement tout composant non simple c de L ($comp(c)\neq\emptyset$) par tous les éléments de comp(c), jusqu'à qu'il n'y ait que des composants simples.

Soit BD = (C, ref, comp) une source de données. Notons par C^S l'ensemble des composants simples. Afin de réaliser la décomposition de l'ensemble L, nous définissons les fonctions decomp et decomp* de la manière suivante.

Définition 10 (Les fonctions decomp et decomp*) : La fonction $decomp : C \longrightarrow \mathcal{P}(C^S)$ est définie récursivement comme suit : $decomp(c)$ = **if** $comp(c) = \emptyset$ **then** $\{c\}$ **else** $\cup_{c'\in comp(c)}$ $(decomp(c'))$

La fonction $decomp^* : \mathcal{P}(C) \longrightarrow \mathcal{P}(C^S)$ est défini par $decomp^*(A) = \cup_{a\in A} (decomp(a))$ ■

Notons par $decomp_k$* la fonction ainsi définie pour la source BD_k
decomp(c) rend l'ensemble de tous les composants simples qui ont participé à la construction de c. Puisque $L=\cup_{k\in D}(L_k)$ et $L_k \subseteq C_k$ on a $decomp_k^*(L_k) \subseteq C^s_k$. Le résultat de la décomposition de l'ensemble des composants de l'entrepôt s'obtient donc par $L'=\cup_{k\in D}(decomp_k^*(L_k))$. Notons que L' n'est formé que de composants simples. L'ensemble L' est le résultat de la décomposition.

L'ensemble des composants de l'entrepôt est ensuite raffiné en utilisant les deux relations S et I de l'environnement retenu. En effet, deux étapes de filtrage sont effectuées sur l'ensemble L selon ces relations. L'ordre est important dans la réalisation de ces deux opérations de filtrage. Un premier filtrage est réalisé en utilisant la relation d'ordre I. Cette opération consiste à laisser les composants correspondant *aux éléments maximaux* de la relation d'ordre I. Un deuxième Filtrage est ensuite effectué en utilisant la relation d'équivalence S. L' est alors filtré en laissant *un seul représentant de chaque classe d'équivalence* de la relation S. Le choix du représentant est aléatoire et il n'intervient pas dans le processus d'intégration.
L'ensemble L'' résultat du filtrage est donc formé : $L'' \subseteq L'$. L'' peut être écrit de la manière suivante : $L''=\cup_{k\in D}(L''_k)$ avec $L''_k=L'' \cap (L_k)$.

Exemple 2 (Ensemble Filtré des composants de l'ED)
Etant donné L, l'ensemble des composants demandés de l'ED :
$L=\{x_1, x_5, y_1, y_2, y_7\}$ avec $L_1=\{x_1, x_5\}$ et $L_2=\{y_1, y_2, y_7\}$
Le résultat de la décomposition est l'ensemble $L'=\{x_2, x_3, x_5, y_1, y_4, y_3, y_8, y_9, y_{10}\}$.
Le filtrage de L' donne l'ensemble L'' des composants gardés pour l'entrepôt :
$L''=\{x_2, x_3, x_5, y_1, y_8, y_9, y_{10}\}$.

4.2 Génération du schéma global de l'ED hétérogène

Cette étape permet de spécifier le nombre de vues de l'ED et la structure de ces dernières. Pour tout L_k *un graphe non orienté* G_k est construit à partir de la source BD_k . Les nœuds de G_k sont en correspondance biunivoque avec $\cup_{C\in Lk}(comp_k(c))$. Les Arcs de G_k sont construits de façon qu'il y a un arc entre deux nœuds n(c) et n(c') de G_k si un ensemble de composants de c est relié à un ensemble de composants de c' par le lien ref.

La Détection des sous-graphes connexes permet de rassembler les composants de chaque vue. On détecte les *sous-graphes connexes* pour chaque graphe G_k.Le graphe G_k est donc l'union de ses sous graphes connexes.

Soit Γ_k l'ensemble formé de ces sous graphes, à savoir: $\Gamma_k=\{G_{k_i} / i\in T_k\}$ où T_k désigne une indexation pour Γ_k.

Soit $T=\prod_{k\in D} T_k$ et d = |D|. Pour tout $u=(u_1,\ldots,u_d) \in T$ on considère le graphe R_u *formé de l'union de tous les* G_{u_i} (i=1,…,d). Puisque ces graphes n'ont pas de nœuds ni d'arcs en commun, on entend par union leur juxtaposition.

On dénote par R l'ensemble des graphes ainsi obtenus : $R=\{R_u / u \in T\}$.Un *schéma de vue* V_u est associé à chaque *graphe* R_u. V_u est composée de l'ensemble de composants simples issus de l'intersection de L'' et X_u. X_u est l'ensemble de tous les composants simples associés aux composants relatifs aux nœuds de R_u (les algorithmes sont présentés dans (Hamdoun, 2006). Notons que le nombre de vues ainsi crée est égal à t=|T|.

Exemple 3 (Schéma global de l'ED)
G_1, le graphe associé à BD_1 est composé de deux sous-graphes connexes G_{11} et G_{12}. G_2, le graphe associé à BD_2 est formé d'un seul sous-graphe connexe $G_{21}=G_2$. R_1 (respectivement R_2) est le résultat de l'union de G_{11} et G_{21} (respectivement G_{12} et G_{21}).

Nous obtenons alors deux vues V_1 et V_2 avec,
$V_1=\{x_2, x_3, y_1, y_8, y_9, y_{10}\}$ et $V_2=\{x_5, y_1, y_8, y_9, y_{10}\}$.

4.3 Construction des vues alimentant l'entrepôt

L'étape précédente a permis d'établir le nombre t de vues de l'entrepôt et leurs structures ($ED=\{V_i / i\in[1..t]\}$). L'algorithme **ConstruitEntrepôt_ETL ()** de construction revient à appliquer ConstruitVue( ) pour chacune des vues.
Dans la suite nous notons par S'(k,c,m) et I'(k, c, m), avec $k\in D$, $m\in D$, $k\neq m$ et $c\in C_k$ les ensembles suivants :

- S'(k,c,m)={c'/ $c \in C_m$ et c' S c}, il s'agit de tous les composants de BD_k appartenant à la même classe d'équivalence que c.
- I'(k,c,m)= {c'/ $c \in C_m$ et c' I c}, il s'agit de tous les composants de BD_k qui sont des prédécesseurs de c.

Algorithme ConstruitEntrepôt_ETL ()
Début
Pour toute V de l'entrepôt **faire {ConstruitVue(V)} Finpour**
Fin

Pour toute vue V à construire, l'algorithme **ConstruitVue( )** consiste à intégrer des données de différentes bases afin d'alimenter la vue. Cet algorithme est présenté dans la suite.

Algorithme ConstruitVue (V){
Début
Pour tout $k\in D$ **faire**
Q :=vue_vide /* initialisation vue vide */
V :=vue_vide
V0 := $(L'_k)\cap V$ /* V0 est l'ensemble de composants de V appartenant à BD_k */
V' := V-V0 /* V' est l'ensemble de composants de V n'appartenant pas à BD_k */
y := $|V'|$ /* y est la cardinalité de V' */
Pour tout $v'\in V'$ **faire**
$SI_{v'}$:= $(S'(skl(v'), v',k) \cup I'(skl(v'), v',k))$
/* $SI_{v'}$ contient l'ensemble des composants synonymes à v' dans BD_k et l'ensemble des composants de BD_k inclus dans v' */
Fin pour
Pour tout $(a_1,\ldots,a_y) \in \prod_{v'\in V'} (SI_{v'})$ faire
Q :=V0 $\cup \{a_1,\ldots,a_y\}$ /* Q correspond à une requête d'interrogation de BD_k , il contient l'ensemble des composants à extraire de BD_k */
Q := Q (union) Créer_Vue(Q) /* Q contient le résultat partiel de l'interrogation de la base BD_k */
Fin pour
V := V (union) Q
Fin pour
Fin}

La procédure Créer_Vue(Q) consiste à exécuter une requête de sélection des différents composants figurant dans Q et éventuellement traduire cette requête suivant le modèle de l'entrepôt. Cette procé-

dure dépend donc de la modélisation adoptée pour la source et le SGBD utilisé. Elle dépend également de la modélisation adoptée pour l'entrepôt. Cette dernière est étroitement liée à celles des sources ainsi que le domaine d'application. Par exemple, quand les sources sont de catégories différentes et le domaine d'application nécessite un échange fréquent de données entre différents systèmes, le modèle adopté pour l'entrepôt est le langage XML.

La procédure Créer_Vue(Q) permet de créer une vue. Des types sont associés aux différents composants de la vue. Ces types sont étroitement liés aux SGBD utilisés.

Exemple 4 (Construction des vues de l'ED)

Pour la vue V_1, le résultat partiel de l'interrogation de BD_1 est :

$Q=\{ x_2, x_3, CompNull_1, CompNull_2, CompNull_3, CompNull_4\}$

Pour la source BD_2, le résultat partiel de l'interrogation de BD_2 est :

$Q=\{ CompNull_1, CompNull_2, y_1, y_8, y_9, y_{10}\}$

Pour la vue V_2, le résultat partiel de l'interrogation de BD_1 est :

$Q=\{ x_5, CompNull_1, CompNull_2, CompNull_3, CompNull_4\}$

Pour la source BD_2, le résultat partiel de l'interrogation de BD_2 est :

$Q=\{ CompNull_1, y_1, y_8, y_9, y_{10}\}$

Un ensemble de composants fictifs (de noms $CompNull_i$) est créé afin d'assurer l'Union-compatibilité entre les requêtes construites dans le cas où $SI_{v'}= \varnothing$. Notons également que tous les types correspondant à un composant et ceux qui lui sont liés par la relation S (une classe d'équivalence CE) sont compatibles. Soit Ty l'ensemble de ces types. Il existe donc, pour chaque classe d'équivalence, un type CE, un type t' tel que $\forall\ t_i \in Ty,\ t_i \rightarrow^* t'$. Dans ce cas, le type associé au représentant choisi est égal à t'. De même, tous les types correspondant à un ensemble EI de composants ordonnés (suivant la relation d'ordre I) de L' sont compatibles. Soit Typ l'ensemble de ces types. Il existe donc, pour chaque ensemble d'éléments ordonnés EI de L' un type t' tel que $\forall\ t_i \in Typ,\ t_i\ t_i \rightarrow^* t'$. t' sera le type associé aux composants correspondant aux éléments maximaux de l'ensemble EI.

5 Conclusion

L'originalité de notre travail, contrairement aux différents travaux sur l'intégration, réside dans le fait de couvrir l'intégration de toutes les catégories de données considérées en même temps. En effet, la plupart des travaux dans la littérature ont porté sur l'intégration de données homogènes (Saccol et Heuser, 2002) (Magnani et al., 2005). Dans le cas contraire, ils ont essayé de restreindre les données hétérogènes de façon à les rendre homogènes et/ou structurées (Kim et Park, 2003) (Beneventano et al., 2004) (Zerdazi et Lamolle, 2006).

Il a fallu donc penser à un cadre formel qui pourrait masquer l'hétérogénéité des données. En effet, notre approche théorique constitue les premiers pas vers un formalisme de l'intégration de ces dernières. L'accent a ainsi été mis sur le fait que les correspondances (″liens d'intégration″) peuvent se faire entre des composants de sources de données de différentes catégories. C'est ce qui constitue le point fort de notre démarche.

Signalons que la relation d'équivalence et la relation d'ordre strict proposées comme ″liens d'intégration″ couvrent la plupart des liens cités dans la littérature notamment les liens d'inclusion, de compatibilité, de disjonction et de similarité. L'originalité de ces définitions de liens réside dans le fait qu'ils sont applicables tant sur les structures que sur les données elles-mêmes.

Dans ce papier, nous avons présenté également l'application de notre approche d'intégration de données hétérogènes dans le cadre des données relationnelle-étendues et XML. Nous avons considéré la relation de Synonymie comme étant la relation d'équivalence adoptée sur les sources et l'Inclusion comme étant la relation d'ordre strict. Ces relations permettent de filtrer l'ensemble des composants de l'entrepôt et par la suite de générer son schéma global en fin de construire les vues de l'entrepôt en les alimentant par les données des sources.

Un outil HDI (Heterogeneous Data Integration for Data Warehouses) a été développé à l'aide d'un prototype en utilisant Oracle et PL/SQL.

Une généralisation de notre approche formelle pourrait se faire en considérant les liens d'intégration dans un environnement d'intégration comme étant un ensemble de relations d'équivalence et un ensemble de relations d'ordre sur les composants des sources. Une étude sur la manière de définir ces ensembles de relations et les correspondances éventuelles qui pourraient se faire entre eux devrait être menée.

Références

Badri, M. (2008). Maintenance des entrepôts de données issus de sources hétérogènes. PhD thesis, Université Paris 5.

Benabdeslem, K., Y. Bennani et E. Janvier (2001). Connectionist Approach for Website visitors Behaviors.

Beneventano, D., S. Bergamaschi, S. Castano, V. D. Antonellis, A. Ferrara, F. Guerra, F. Mandreoli, G. C. Ornetti et M. Vincini (2002). Semantic Integration and query optimization of heterogenous data sources. Lecture Notes in Computer Science (LNCS 2426) LNCS 2426, Proceedings of the worshops on advances in Object-Oriented Information Systems (OOIS), pp 154-165.

Beneventano, D. et S. Bergamaschi (2004). The MOMIS Methodology for Integrating Heterogeneous Data Sources. Proceedings of IFIP World Computer Congress, Toulouse France, pp 22-27.

Bennani, Y. (2006). Apprentissage connexionniste. Edition Hermes Science Publications, ISBN : 2746213370.

Cluet, S. et J. Siméon(2000). YATL : a functional and declarative language for XML. Draft manuscript. Mai 2000.

Da Silva, A. S., I.M.R.E. Filha, A. H. F. Laender et D. W. Embley (2002). Representing and querying semistructured Web Data Using Nested Tables With structural Variants. Lecture Notes in Computer Science (LNCS 2503), Proceedings of the 21st International Conference on Conceptual Modeling, Tampere, Finlande,), pp 135-151.

Doan, A., J. Madhavan, P. Domingos et A. Halevy (2002). Learning to Map between Ontologies on the Semantic Web. Proceedings of the World-Wide Web Conference. (WWW-02), pp 662-673.

Hamdoun, S. (2006). Construction d'entrepôts de données par intégration de sources hétérogènes. Thèse présentée à l'université Paris Nord.

Inmon, B. (1995). Multidimensional Data Bases and Data Warehousing. Data Management Review.

Kim, H. H. et S. S. Park (2003). Building a Web-enabled Multimedia Data warehouse. Lecture Notes in Computer Science (LNCS 2713), proceedings of the Web Communication Technologies and Internet-Related Social Issues (HSI2003), pp 594-600.

Magnani, M., N. Rizopoulos, P.J. McBrien et D. Montesi (2005). Schema Integration based on Uncertain Semantic Mappings. Lecture Notes in Computer Science (LNCS 3716), Proceedings of 24th International Conference on Conceptual Modeling, Klagenfurt, Austria (ER05), pp 31-46.

Mining. Proceedings of the International Conference on Computer Systems and Applications, Beirut, Lebanon.

Nassis, V., R. Rajugan , T. S. Dillon et W. Rahayu (2004). Conceptual Design of XML Document Warehouses. Lecture Notes in Computer Science (LNCS 3181), Proceedings of the 6th Interna-

tional Conference on Data Warehousing and Knowledge Discovery (DaWaK), Zaragoza, Spain, pp 1-14.

Pontieri, L., D. Ursino et E. Zumpano (2003). An approach for the extensional integration of data sources with heterogeneous representation formats. Data & Knowledge Engineering 45, pp 291-331, 2003.

Prakash, S., S. S. Bhowmick et S. Madria (2006). Efficient recursive XML query processing using relational database systems. Data & Knowledge Engineering 58 (2006), pp 263-298.

Saccol, D. d. B. et C. A. Heuser (2002). Integration of XML Data. *Lecture Notes in Computer Science (LNCS 2590), proceedings of the Very Large Data Bases (VLDB) workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT2002) and of the international Conference on Advanced Information System Engineering (CAISE) workshop on Databases in Telecommunications and Web (DTWeb2002), Revised papers*, pp 68-80.

Serafini, L., P. Bouquet, B. Magnini et S. Zanobini (2003). An algorithm for matching contextualized schemas via SAT. Technical Report 0301-06, ITC-irst, Trento, Italie.

Wang, L., E. A. Rundensteiner et M. Mani (2006). Updating XML views publihed over relational databases: Towards the existence of a correct mapping. *Data & Knowledge Engineering 58 (2006)*, pp 263-298.

Xu, L. et D. W. Embley (2003). Discovering Direct and Indirect Matches for Schema Elements. Proceedings of the Eighth International Conference on Database Systems for Advanced Applications table of contents, pp 39.

Zerdazi, A. et M. Lamolle (2006). Intégration de sources hétérogènes par matching semi-automatique de schémas XML étendus. Proceedings of INFORSID, pp 991-1006.

Summary

This work describes the construction of a data warehouse by the integration of heterogeneous data. The data sources can be structured, semi-structured or unstructured (relational, object-relational and XML databases). We propose a theoretical approach based on an integration environment definition. This environment is formed by data sources and inter-schema relationships between these sources (equivalence and strict order relations). We consider a data source as a set of components interconnected by compositions and references' relationships.

Our approach is composed of three steps allowing data warehouse component filtering, global schema generation and construction of data warehouse views.