6e Conférence Francophone de MOdélisation et SIMulation - MOSIM'06 - du 3 au 5 avril 2006 - Rabat - Maroc
"Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités"

UN MODELE A BASE ONTOLOGIQUE POUR LA GESTION DE L'EVOLUTION ASYNCHRONE DES ENTREPOTS DE DONNEES

Dung NGUYEN XUAN, Ladjel BELLATRECHE, Guy PIERRA

LISI/ENSMA
2, avenue Clément Ader
86960 Futuroscope - France
(nguyen, bellatreche, pierra)@ensma.fr

RÉSUMÉ : *Une nouvelle génération de systèmes d'intégration utilise des ontologies pour résoudre les conflits sémantiques et structurels entre les différentes sources de données participant au processus d'intégration. Ces systèmes supposent l'existence d'une ontologie partagée de domaine et que chaque source possède une ontologie locale qui référence, et éventuellement étend l'ontologie partagée. Le processus d'intégration est alors basé sur ces références qui constituent une articulation des ontologies locales et l'ontologie partagée. Notons que les sources de données sont indépendantes, chacune peut évoluer indépendamment des autres, ce qui engendre un problème d'évolution asynchrone. Dans le contexte d'une telle intégration à base ontologique, les évolutions concernent donc à la fois les ontologies, les schémas et les données. Dans cet article, nous proposons un modèle pour la gestion de l'évolution des systèmes d'intégration à base ontologique, dont le résultat est matérialisé sous forme d'un entrepôt de données. L'hypothèse fondamentale de notre travail, appelée, le principe de continuité ontologique, stipule qu'une évolution d'une ontologie ne peut infirmer un axiome antérieurement vrai. Ce principe permet d'interpréter toute instance représentée. En conséquence, il simplifie considérablement la gestion de l'évolution des ontologies. Ceci permet d'assurer une intégration automatique. Ce travail a été motivé par l'intégration automatique des catalogues de composants industriels dans les bases de données d'ingénierie. Il a été validé par un prototype sous un environnement ECCO et le langage EXPRESS.*

MOTS-CLÉS : *Système d'intégration, ontologies, évolution asynchrone*

1 INTRODUCTION

Un système d'intégration dans une perspective entrepôt, matérialise des données provenant de diverses sources de données hétérogènes. Dans un tel contexte deux issues sont à considérer : (1) la résolution des conflits dus à l'hétérogénéité schématique et sémantique entre les différentes sources de données, et (2) l'aspect faiblement couplé des sources. L'hétérogénéité sémantique représente un enjeu essentiel pour les systèmes d'intégration (Doan et al. 2004). Afin de réduire cette hétérogénéité, de plus en plus d'approches d'intégration visent à associer aux données une ontologie qui en définit le sens. Une ontologie est définie comme une spécification formelle et explicite d'une conceptualisation partagée (Gruber 1993). L'objectif de ces ontologies, dont chacune est normalement partagée par une communauté, est de représenter formellement la signification des données contenues. Lorsque le domaine comporte une ontologie commune, par exemple normalisée, mais partielle, et lorsque chaque ontologie locale référence, et étend, l'ontologie commune, l'intégration automatique devient alors possible (Bellatreche et al. 2004). En effet, les articulations entre ontologies locales et l'ontologie commune permettent une résolution automatique des différents conflits (conflits de nom, conflits de contexte, conflits de mesure, etc.) entre les sources d'informations participant au processus d'intégration.

Notre domaine d'application cible est le commerce électronique professionnel et l'échange de données techniques dans le domaine des composants industriels. Il s'agit, d'une part, de pouvoir rechercher de façon entièrement automatique quel fournisseur présent sur le Web est susceptible de fournir un roulement à billes ayant des caractéristiques techniques données (par exemple, supporter une charge radiale de 100 Newton et axiale de 6 Newton avec une durée

de vie de 2000 H en tournant à 500 t/s). Et ceci quelle que soit la structure de la base de données du "catalogue" susceptible de le contenir. Il s'agit, d'autre part, de pouvoir intégrer les catalogues de différents fournisseurs, représentée sous forme d'une base de données, au sein d'un entrepôt de données situé dans une entreprise utilisatrice, en offrant des possibilités d'accès ergonomiques. Etant donné que les différents catalogues appartiennent à différents fournisseurs, qui peuvent les faire évoluer indépendamment, cette évolution doit être prise en considération par le système d'intégration.

Dans les domaines techniques, les notions d'interchangeabilité et de normalisation sont très développées. Pour chaque domaine particulier (l'électronique, les composants mécaniques, etc.), un vocabulaire technique consensuel existe donc déjà, sous forme textuelle, pour les termes essentiels de chaque domaine. Afin de construire des ontologies partagées pour les domaines techniques, notre approche a donc consisté, d'abord, à formaliser ce langage consensuel sous forme d'ontologie, puis à exploiter ces ontologies pour les applications d'intégration, d'échange et d'interrogation. Cette formalisation a donné lieu, dans les années 90, un développement d'un modèle d'ontologie, ainsi qu'un modèle d'échange d'instances d'entités décrites en termes de ces ontologies. Ceci constituait le projet PLIB dont les résultats ont en particulier donné lieu à un ensemble de normes ISO dans la série 13584 (Parts Library) dont les différentes parties ont été publiées entre 1996 et 2004. Parallèlement, des ontologies de domaines conformes à ce modèle ont été développées et normalisées au niveau international. La première publiée en 1998, décrit les principales catégories et propriétés de composants électroniques. Aujourd'hui plusieurs dizaines sont actuellement publiées ou en cours de développement. Notons que l'évolution technique impose régulièrement de mettre à jour ces ontologies normalisées, chacune étant désormais associée à une agence de maintenance destinée à assurer cette évolution. Le modèle d'ontologie PLIB est le noyau de notre système d'intégration et d'évolution.

Notre approche d'intégration est basée sur trois principes : (1) chaque source participante dans le processus d'intégration doit contenir sa propre ontologie ; une telle source est appelée base de données à base ontologique (BDBO) (Pierra & et al. 2005), (2) chaque ontologie locale s'articule a priori avec une (ou des) ontologie(s) normalisée(s), et (3) chaque ontologie locale étend ces ontologies partagées pour satisfaire ses besoins (voir Figure 1).

Quelques systèmes d'intégration ont été développés

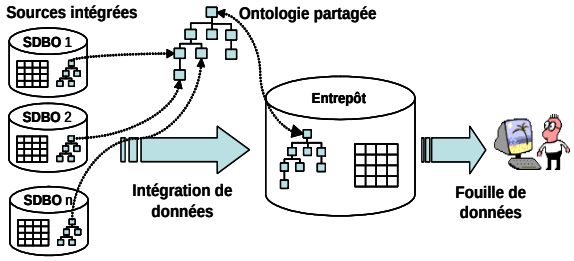

Figure 1: Intégration sémantique par articulation a priori d'ontologies

autour de l'hypothèse d'existence d'une ontologie partagée. C'est le cas du projet Picsel2 (Reynaud & Giraldo 2003) pour intégrer les services Web, à partir de l'ontologie OTA (Open Travel Alliance) et du projet COIN pour échanger les données financières (Goh et al. 1999). Par cotre aucune de ces approches, à notre connaissance, n'a pris en compte le problème d'évolutions des ontologies.

Notons que les fournisseurs des sources de données étant différents, chaque source se comporte indépendamment des autres (à l'exemple des environnements dynamiques comme le WWW) (Heflin & Hendler 2000). En conséquence, la relation entre le système intégré et ses sources est faiblement couplée. Une source peut modifier sa structure et sa population sans informer les autres. Ce qui engendre des anomalies de maintenance (Chen et al. 2004). Dans un tel contexte (asynchrone), la gestion de l'évolution concerne les schémas, les populations des sources, l'ontologie partagée et les ontologies locales.

Dans un système d'intégration à base ontologique, une ontologie locale peut difficilement être identique à l'ontologie partagée, et ce pour deux raisons : une raison de *domaine couvert* (autonomie des domaines), et une *raison de synchronisme* (évolution asynchrone). Le domaine couvert n'est en général pas le même :

- d'une part, chaque source n'utilise en général qu'une partie de l'ontologie partagée,
- inversement, certains concepts partagés ont souvent besoin d'être affinés localement pour représenter les spécificités locales. C'est le cas, tout particulièrement, dans une ontologie de produits techniques, pour représenter les spécificités des produits créés par chaque entreprise. Il faut donc assurer à chaque source une indépendance maximale, tout en permettant une intégration sémantique automatique.

Nous avons déjà proposé une solution au problème de l'autonomie des domaines. C'est l'approche

d'intégration sémantique par articulation a priori d'ontologies (Bellatreche et al. 2004) dans laquelle on exige simplement que chaque classe locale référence sa plus petite classe subsumante dans l'ontologie partagée et en importe, sans les modifier, les propriétés utiles.

Mais, de plus, les ontologies évoluent et il est difficile de les synchroniser complètement. Cela est du aux facteurs suivants : (i) l'indépendance des sources et (ii) le temps nécessaire pour diffuser les évolutions des ontologies partagées dans une communauté.

Dans ce contexte, *deux problèmes* doivent être résolus: (1) la gestion des évolutions d'ontologies afin de maintenir les relations entre les ontologies et les données (Heflin & Hendler 2000) et (2) la gestion de cycle de vie des instances (la connaissance des instances qui existaient dans l'entrepôt à un instant donné).

Pour illustrer le problème de l'évolution dans un contexte asynchrone considérons l'exemple suivant :

Exemple 1 *La Figure 2 représente un catalogue des disques externes. Initialement, la version 1 de l'ontologie locale de la source S_1 ("Disque Externe") référence la version 1 de l'ontologie partagée ("Disque dur"). Supposons que l'ontologie partagée et la source S_1 évoluent d'une manière indépendante. L'ontologie partagée subit les changements suivants : (1) le domaine de données de la propriété "Interface" est étendu, (2) ajout d'une nouvelle propriété "Dimension". Les changements au niveau de la source S_1 sont : (1) l'ajout d'une nouvelle propriété "Garantie" à l'ontologie locale, (2) le renommage de la propriété "Code" par "Série", (3) la suppression et l'ajout dans la table de données des propriétés "Transfert" et "Garantie", respectivement, et (4) l'ajout/suppression d'instances dans la table de données.*
Afin de prendre en compte ces changements, nous devons répondre aux questions suivantes : peut-on accéder aux données de la source S_1 à travers la version 2 de l'ontologie partagée ? comment les problèmes de duplicatas de données et de rafraîchissement automatique de données seront résolus (dans le cas où toutes les données de deux versions de la table sont stockées dans l'entrepôt) ?

Dans les sections suivantes, nous répondons à ces questions. L'objectif de ce papier est de proposer une approche et un modèle en réponse au problème de l'évolution asynchrone des ontologies et des données. Notre approche est fondée sur les caractéristiques particulières d'une ontologie comparée à un modèle conceptuel. Elle consiste à définir un *ensemble de contraintes* qui doit être respecté à la fois par l'ontologie partagée et par toutes les sources de données participantes au processus d'intégration. Notre modèle permet alors de gérer complètement et automatiquement les évolutions asynchrones.

2 INTEGRATION PAR ARTICULATION A PRIORI D'ONTOLOGIES

Avant d'introduire notre méthodologie pour gérer les versions des ontologies et des schémas, quelques concepts et définitions s'imposent.

2.1 Notions de base

Une ontologie O est définie formellement comme un quadruplet (Bellatreche et al. 2004) suivant :
$O :< C, P, Sub, Applic >$, avec :
• C : l'ensemble des classes utilisées pour décrire les concepts d'un domaine donné (comme les service de voyages, les pannes des équipements, les composants électroniques, etc.). Chaque classe est associée à un identifiant universel globalement unique (GUI).
• P : l'ensemble des propriétés utilisées pour décrire les instances de l'ensemble des classes C. Chaque propriété est associée à un identifiant universel (GUI).
• $Sub : C \rightarrow 2^C$ est la relation de subsomption (is-a et is-case-of) (Bellatreche et al. 2004), qui, à chaque classe c_i de l'ontologie, associe ses classes subsumées directes. Sub définit un ordre partiel sur C.
• $Applic : C \rightarrow 2^P$, associe à chaque classe de l'ontologie les propriétés qui sont applicables pour chaque instance de cette classe. Les propriétés qui sont applicables sont héritées de la relation is-a et peuvent être importées de façon sélective à travers la relation *is-case-of*.

Une base de données à base ontologique (BDBO) est définie formellement comme un quadruplet (Pierra & et al. 2005) : $BDBO :< O, I, Pop, Sch >$, avec :
• O représente son ontologie ($O :< C, P, Sub, Applic >$).
• I représente l'ensemble des instances de données de la base de données. La sémantique de ces instances est décrite par O en les associant à des classes et en les décrivant par les valeurs de propriétés définies dans l'ontologie partagée,
• $Pop : C \rightarrow 2^I$, associe à chaque classe les instances qui lui appartiennent (directement où par l'intermédiaire des classes qu'elle subsume). Pop(c_i) constitue donc la population de c_i.
• $Sch : C \rightarrow 2^P$, associe à chaque classe c_i les propriétés qui sont applicables pour cette classe et qui sont effectivement utilisées pour décrire tout ou partie des instances de $Pop(c_i)$. Pour toute classe c_i,

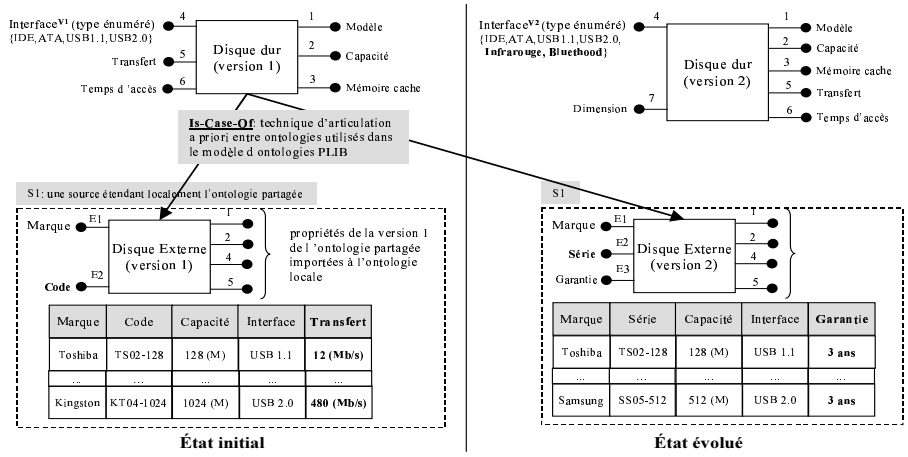

Figure 2: Exemple de l'évolution de données et d'ontologies dans un système d'intégration à base ontologique

Sch(c_i) doit satisfaire: $Sch(c_i) \subseteq Applic(c_i)$.

2.2 Notre approche d'intégration

Soient $O :< C, P, Sub, Applic >$ l'ontologie partagée, et $S = \{S, ..., S_n\}$ un ensemble de BDBOs participantes au processus d'intégration. Dans une intégration a priori, on intègre d'abord les ontologies ensuite les données (Bellatreche et al. 2004). L'intégration des ontologies est effectuée et exprimée à l'aide de relations de subsomption entre classes de différentes ontologies et d'importation de propriétés. Nous supposons donc désormais que chaque S_i a été conçue en deux étapes (Bellatreche et al. 2004) :

1. l'administrateur de cette source défini sa propre ontologie : $O_i : < C_i, P_i, Sub_i, Applic_i >$, en référencant l'ontologie partagée O. Ces références sont stockées dans la source S_i et constituent une *articulation* M_i entre O et O_i définit comme suit : $C \rightarrow 2^{C_i}$, où $M_i(c)$ est l'ensemble des classes de C_i directement subsumées par $c \in C$. A travers ces relations de subsumption, toutes les propriétés pertinentes de P sont importées dans P_i.

2. L'administrateur choisit pour chaque classe c_i son schéma $Sch_i(c_i)$, et ses instances $Pop_i(c_i)$ [1].

Une source S_i articulée avec une ontologie partagée O peut être formulée comme suit: $S_i :< O_i, I_i, Sch_i, Pop_i, M_i >$. Selon les articulations entre les ontologies locales et l'ontologie partagée, plusieurs opérateurs d'intégration de BDBOs peuvent être définis (Bellatreche et al. 2004). Nous considérons seulement un opérateur, appelé *ExtendOnto* dans lequel : (a) chaque classe de chaque S_i référence (directement, ou par héritage) sa (ou ses) plus petites classes subsumante(s) dans l'ontologie partagée, (b) chaque classe de chaque S_i importe de ses classes subsumantes dans l'ontologie partagée les propriétés qu'elle souhaite utiliser, en préservant leurs identifications, et (c) toutes les classes de S_i, ainsi que toutes les propriétés de S_i qui ne sont pas identifiées avec des propriétés importées, sont spécifiques de la source S_i.

Exemple 2 *Pour illustrer cette formalisation de S_i, reprenons la partie initiale de la figure 2. L'ontologie locale de la source S_1 "Disque Externe" référence l'ontologie partagée "Disque dur" en important les propriétés qui lui conviennent : ("Modèle", "Capacité", "Interface", "Transfert"). Localement, elle définie deux nouvelles propriétés "Marque" et "Code". En conséquence, les éléments de la source $S_1 =$ <O_1,I_1, Sch_1, Pop_1, M_1> sont définis comme suit : $C_1 =$ {"Disque Externe"}; Sub_1("Disque Externe") $= \phi$; Sch_1("Disque Externe") = {"Marque", "Code", "Capacité", "Interface", "Transfert"}; M_1("Disque dur") = {"Disque Externe"}.*

Le résultat de l'intégration des BDBOs par ExtendOnto est lui-même une BDBO qui réunit les instances de toutes les sources : $DW :< O_{DW}, I_{DW}, Sch_{DW}, Pop_{DW} >$ qui est déterminé de la manière suivante :

1. O_{DW} est calculée comme l'intégration des ontologies locales au sein de l'ontologie partagée :

   (a) $C_{DW} = C \cup (\cup_{i \in [1..n]} C_i)$,

   (b) $P_{DW} = P \cup (\cup_{i \in [1..n]} P_i)$,

[1] pour les classes non feuilles, ces choix doivent évidemment respecter les contraints résultant du polymorphisme: $\forall c_i \in Sub(c_k) : Pop(c_i) \subset Pop(c_k), Sch(c_i) \cap Applic(c_k) \subset Sch(c_i)$

(c) $Applic_{DW}(c) = \begin{cases} Applic(c), & \textbf{si } c \in C \\ Applic_i(c), \textbf{ si } c \in C_i \end{cases}$

(d) $Sub_{DW}(c) = \begin{cases} Sub(c) \cup (\cup_{i\in[1..n]} M_i(c), & \textbf{si } c \in C \\ Sub_i(c), \textbf{ si } c \in C_i \end{cases}$

2. $I_{DW} = \cup_{i\in[1..n]} I_i$,

   Dans ce cas d'intégration, les instances intégrées sont stockées dans les tables de données comme dans les sources dont elles proviennent. C'est-à-dire que:

   (a) $\forall c_i \in C_{DW} \wedge c_i \in C_i \wedge \forall i \in [1..n]$:

   - $Sch_{DW}(c_i) = Sch_i(c_i)$, et
   - $Pop_{DW}(c_i) = Pop_i(c_i)$,

   (a) $\forall c \in C$

   - $Sch_{DW}(c) = Applic(c) \cap (Sch(c) \cup (\cup_{c_j \in Sub_{DW}(c)} Sch(c_j)))$,
   - $Pop_{DW}(c) = \cup_{c_j \in Sub(c)} Pop(c_j)$

3 CONTRAINTES D'EVOLUTION

3.1 Principe de continuité ontologique

Les contraintes que l'on peut définir pour régler l'évolution des entrepôts de données à base ontologique résultent des différences fondamentales existantes, de point de vue évolution, entre les modèles conceptuels et les ontologies. Un modèle conceptuel est un modèle, c'est-à-dire, selon Minsky, un objet qui, permet de répondre à des questions sur un autre objet à savoir les informations représentées dans une base de données (Minsky 1979). Lorsque les questions changent, c'est-à-dire, lorsque les objectifs organisationnels auxquels répond un système d'information sont modifiés, son modèle conceptuel est modifié, sans que cela signifie le moins du monde que le domaine modélisé est modifié. Au contraire, une ontologie est une conceptualisation visant à représenter l'essence des entités d'un domaine donné sous forme consensuelle pour une communauté. C'est une théorie logique d'une partie du monde, partagée par toute une communauté, et qui permet aux membres de celle-ci de se comprendre. Ce peut être, par exemple, la théorie des ensembles (pour les mathématiciens), la mécanique (pour les mécaniciens) ou la comptabilité analytique (pour les comptables). Pour de telles ontologies, deux types de changements doivent être distingués : *l'évolution normale* d'une théorie est son approfondissement. Des vérités nouvelles, plus détaillées s'ajoutent aux vérités anciennes. Ce qui était vrai hier reste vrai aujourd'hui. Mais il peut également arriver que des axiomes de la théorie aient à être remis en cause. Il ne s'agit plus là d'une évolution mais d'une *révolution*, où deux systèmes logiques différents vont coexister ou s'opposer.

Les ontologies que nous visons correspondent à cette conception. Il s'agit d'ontologies soit normalisées, par exemple au niveau international, soit définies par des consortiums importants et qui formalisent de façon stable les connaissances d'un domaine technique. Les changements auxquels nous nous intéressons dans notre approche ne sont donc pas les révolutions, qui correspondent à un changement d'ontologie, mais les évolutions d'ontologie.

Nous imposerons donc aux ontologies manipulées, qu'elles soient globales ou locales de respecter la contrainte suivante dont nous détaillons les conséquences dans la section 3.2 :

Principe de continuité ontologique : *si l'on considère chaque ontologie intervenant dans le système d'intégration à base ontologique comme un ensemble d'axiomes, tout axiome vrai pour une certaine version de l'ontologie restera vrai pour toutes les versions ultérieures.*

3.2 Contraintes sur les évolutions locales des ontologies

Notons maintenant par un indice supérieur la version des différentes composantes d'une ontologie :
$O^k =< C^k, P^k, Sub^k, Applic^k >$.

Permanence des classes L'existence d'une classe ne pourra être infirmée à une étape ultérieure : $C^k \subset C^{k+1}$. Pour tenir compte de la réalité, il pourra apparaître pertinent de considérer comme obsolète telle ou telle classe. Elle sera alors marquée en tant que telle ("deprecated"), mais continuera à faire partie des versions ultérieures de l'ontologie. Par ailleurs la définition d'une classe pourra être affinée sans que l'appartenance à cette classe d'une instance antérieure ne puisse être remise en cause. Cela signifie que : (i) la définition des classes pourra elle-même évoluer, (ii) chaque définition d'une classe sera associée à un numéro de version, et (iii) la définition (intensionnelle) de chaque classe englobera les définitions (intentionnelles) des ses versions antérieures.

Permanence des propriétés De même $P^k \subset P^{k+1}$. Une propriété pourra, de même, devenir obsolète sans que la valeur existante d'une propriété pour une instance existante puisse être remise en cause, une propriété pourra évoluer dans sa définition ou dans son domaine de valeurs. Le principe de continuité ontologique implique que les domaines de valeurs ne pourront être que croissants, certaines valeurs étant, éventuellement, marquées comme obsolètes.

Permanence de la subsomption La subsomption est également un concept ontologique qui ne pourra être infirmé. Notons $Sub^* : C \rightarrow 2^C$ la fermeture transitive de la relation de subsomption directe Sub. On a alors : $\forall c \in C^k, Sub^{*k}(c) \subset Sub^{*k+1}(c)$. Cette contrainte permet évidemment un enrichissement de la hiérarchie de subsomption des classes, par exemple en intercalant des classes intermédiaires entre deux classes liées par une relation de subsomption.

Description des instances Le fait qu'une propriété $p \in Applic(c)$ signifie que la propriété est rigide pour toute instance de c. Il s'agit encore d'un axiome qui ne pourra être infirmé :

$\forall c \in C^k, Applic^{*k}(c) \subset Applic^{*k+1}(c)$

Soulignons que ceci ne suppose pas que les mêmes propriétés soient toujours utilisées pour décrire les instances d'une même classe. Il ne s'agit en effet plus là d'une caractéristique ontologique, mais seulement de nature schématique.

3.3 Identification des différents éléments

Gérer l'évolution suppose de pouvoir désigner et donc d'identifier, tous les éléments faisant l'objet d'évolution.

3.3.1 Identification des classes et propriétés

Nous avons déjà précisé que toute classe et toute propriété étaient associées à des identifiants universels (GUI). En fait ces identifiants contiennent deux parties : un code (unique) et une version (entière) : *GUI :: = code version.*

Toute référence entre éléments utilisant le GUI, la référence est elle-même versionnée par les versions de ses extrémités. Enfin, toute définition de classe ou de propriété contient, en particulier, la date à partir de laquelle cette version est valide.

Une classe peut connaître trois types d'évolution : (1) une évolution ontologique (par exemple, modification de la définition ou augmentation des propriétés applicables), (2) une évolution schématique (une propriété de plus ou de moins est utilisée pour décrire les instances), et (3)une évolution de sa population (une instance est ajoutée ou enlevée). Pour des raisons de simplicité ces trois types de changements donnent lieu a l'incrémentation d'un même indicateur de version. Une version caractérise donc à la fois une définition, un schéma et une population.

3.3.2 Identification des instances

Dans le domaine qui nous occupe, la durée de vie d'une instance (i.e., un certain composant) peut être largement supérieure au cycle de mise à jour des sources de données (typiquement, chaque année). Afin de pouvoir reconnaître lors d'un rafraîchissement de l'entrepôt, si une instance est déjà existante, toute source doit définir pour chaque classe ayant une population une *clé sémantique*. Cette clé est constituée de la représentation sous la forme d'une unique chaîne de caractères, des valeurs d'une ou plusieurs propriétés applicables de cette classe qui obéissent à une contrainte d'unicité. Les propriétés devront toujours être fournies pour chaque instance de la classe et ne devront jamais être modifiées pour une même instance. Les autres propriétés pourront ou non être fournies selon le choix (à chaque version) du schéma des instances de la classe. Leurs valeurs ne devront pas, par contre, être modifiées d'une version à l'autre.

Le cycle de vie d'une instance (apparition et disparition) est alors défini par les versions des classes auxquelles elle appartient.

4 MODELE DE GESTION

L'objectif de notre modèle de gestion est (i) de permettre un accès global à l'ensemble des instances existantes à un instant donné via l'ontologie partagées de l'entrepôt, (ii) de connaître l'historique des instances, et (iii) éventuellement, de savoir, pour chaque instance, à quelle version de l'ontologie elle correspond. On décrit ci-dessous successivement comment on atteint les trois aspects ci-dessus.

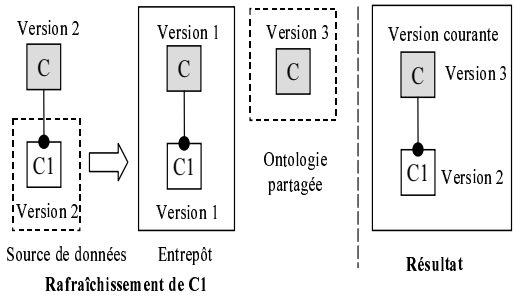

Figure 3: Modèle des versions flottantes

4.1 Réalisation des mises à jour

Nous supposons que notre entrepôt de données est rafraîchi de la façon suivante. A des moments donnés, choisis par l'administrateur de l'entrepôt, la version courante d'une source S_i est intégrée dans l'entrepôt. Cette version courante S_i comporte son ontologie, ses références à l'ontologie partagée, et son contenu *(certaines instances étaient éventuellement déjà intégrées, d'autres sont nouvelles, d'autres sont supprimées)*. Ce scénario correspond, par exemple, dans le domaine de l'ingénierie, à un entrepôt qui consolide les descriptions de composants d'un ensemble de fournisseurs. Un rafraîchissement est effectué chaque fois qu'une nouvelle version d'un catalogue électronique d'un fournisseur est reçue.

4.2 Accès global aux instances courantes : versions flottantes de concepts ontologiques

On appelle instances courantes de l'entrepôt les instances résultant du plus récent rafraîchissement à partir de chacune des sources. La principale difficulté, résultant de l'autonomie de chaque source, est que lors de deux rafraîchissements successifs par deux sources différentes, la même classe de l'ontologie partagée c peut être référencée par une articulation de subsomption (voir section 3.2) dans des versions différentes, par exemple c^k et c^{k+j} par deux classes c_i^n et c_j^p. Il convient cependant de noter que, compte tenu du principe de continuité ontologique : (1) toutes les propriétés applicables à c^k sont également applicable à c^{k+j}, et (2) toutes les classes subsumées par c^k sont également subsumées par c^{k+j},

Donc la relation de subsomption entre c^k et c_i^n pourra être remplacée par une relation de subsomption entre c^{k+j} et c_i^n. La classe c^k n'est donc pas nécessaire pour accéder aux instances de c_i^n. Cette remarque nous amène à proposer un modèle, appelé, *modèle des versions flottantes*, qui nous permet d'accéder aux données via une seule version de l'ontologie de l'entrepôt. Cette version, appelée *"version courante"* de l'ontologie l'entrepôt, est telle que la version courante de chacune de ses classes c^f est supérieure ou égale à la plus grande version de celle même classe référencée par une articulation de subsomption lors d'un quelconque rafraîchissement.

Pratiquement, cette condition est assurée de la façon suivante :
(a) si une articulation M_i référence une classe c^f avec une version inférieure à f, alors M_i est modifié pour référencer c^f,
(b) si une articulation M_i référence une classe c^f avec une version supérieure à f, alors l'entrepôt télécharge la dernière version de l'ontologie partagée et fait migrer toutes les références M_i (i = 1..nombre des sources) vers les nouvelles versions courantes.

Exemple 3 *Lors d'un rafraîchissement, une classe C_1 référence une classe partagée C^2. Mais l'entrepôt contient C^1. Alors l'entrepôt télécharge la version courante de l'ontologie partagée. Celle-ci étant 3, la classe C_1 est modifiée pour référencer C^3 (Figure 3).*

Si le seul besoin des utilisateurs est de connaître les instances courantes, alors, lors de chaque rafraîchissement, la table éventuellement associée à chaque classe provenant d'une ontologie locale dans l'entrepôt est simplement remplacée par la table courante correspondante dans la source locale.

4.3 Représentation des historiques des instances

Il peut être utile, dans certains cas, de savoir quelles instances existaient dans l'entrepôt à tout instant passé. Ceci n'exige pas nécessairement d'archiver également les versions d'ontologies puisque la version courante est compatible avec toutes les instances passées. Ce problème a déjà été étudié sous le nom de "schema versionning" par Wei et al (Wei & Elmasri 1999), où toutes les données versionnées d'une table sont sauvegardées. Pour ce faire deux solutions sont possibles:

- Dans l'approche *stockage explicite* (Bebel et al. 2004), (Wei & Elmasri 1999), toutes les versions de chaque table sont explicitement stockées (voir Figure 4). Cette solution a deux avantages : (i) elle est facile à implémenter et permet une automatisation du processus de mise à jour de données, et (ii) le traitement des requêtes est rapide

Stockage explicite dans des versions d 'une table

Table de disques externes de la version 1

Marque	Code	Capacité	Interface	**Transfert**
Toshiba	TS02-128	128 (M)	USB 1.1	**12 (Mb/s)**
...	...	...	...	...
Kingston	KT04-1024	1024 (M)	USB 2.0	**480 (Mb/s)**

Table de disques externes de la version 2

Marque	Série	Capacité	Interface	**Garantie**
Toshiba	TS02-128	128 (M)	USB 1.1	**3 ans**
...	...	...	...	...
Samsung	SS05-512	512 (M)	USB 2.0	**10 ans**

Stockage implicite dans une table unique (dans notre implémentation)

Marque	**Code**	**Capacité**	**Interface**	Transfert	Garantie	V_min	V_max
Toshiba	TS02-128	128 (M)	USB 1.1	**12 (Mb/s)**	**3 ans**	01	
...	...	...	...	...	...	...	...
Kingston	KT04-1024	1024 (M)	USB 2.0	**480 (Mb/s)**	**null**	01	02
Samsung	SS05-512	512 (M)	USB 2.0	**null**	**10 ans**	02	

Figure 4: Le stockage explicite et implicite

dans le cas où on précise la ou les versions de recherche. Par contre le coût peut être important si la requête nécessite un parcours de toutes les versions de données disponibles dans l'entrepôt. Un autre inconvénient est le coût de stockage à cause de duplicata de données.

- Dans l'approche *stockage implicite* (Wei & Elmasri 1999): une seule version de schéma de chaque table T est stockée. Ce schéma est obtenu en faisant l'*union* de toutes les propriétés figurant dans les différentes versions. On y ajoute a chaque rafraîchissement, toutes les instances existant dans la table courante. Les instances sont complétées par des valeurs nulles (voir Figure 4). Cette solution évite le parcours de plusieurs versions d'une table donnée. Les inconvénients majeurs de cette solution sont : (i) le problème de duplicatas est toujours présent, (ii) l'implémentation plus difficile que l'approche précédente en ce qui concerne le calcul automatique du schéma des tables stockées; (iii) le tracé du cycle de vie de données est difficile à mettre en oeuvre (**"valid time"** (Wei & Elmasri 1999)) et (iv) l'ambiguïté de la sémantique des valeurs nulles.

Notre solution suit la deuxième approche et résout les problèmes de la manière suivante :

1. le problème de **duplicata de données** est évité grâce à l'identification sémantique unique (valeur de la CLE sémantique) de chaque instance de classe,
2. le problème d'automatisation du processus de mise à jour du schéma de table est résolu à travers l'utilisation d'identifiants universels (GUI) pour toutes les propriétés.
3. le problème de la représentation du cycle de vie des instances est résolu par un couple de propriétés : $(Version_{min}, Version_{max})$. Ce dernier nous permet de savoir la validé d'une instance donnée.
4. le problème d'ambiguïté de la sémantique de la valeur nulle : est traité grâce à l'archivage des fonctions Sch de différentes versions de chaque classe de base d'une table. Cet archivage nous permet de déterminer le vrai schéma de version d'une table, et donc la représentation initiale de chaque instance.

4.4 Entrepôt complètement historisé

En fait, l'articulation stockée dans la version courante de l'ontologie d'entrepôt entre une ontologie locale et l'ontologie partagée peut ne pas être sa définition originale (voir la Figure 3) . Dans le cas où il apparaît nécessaire de pouvoir accéder à une instance à travers les définitions ontologiques qui existent lorsque cette instance était elle-même active, il est nécessaire d'archiver également toutes les versions successivement de l'ontologie de l'entrepôt. Ce peut être utile, par exemple, pour connaître ce qu'était, à l'époque de l'instance, le domaine précis d'une de ses propriétés énumérées(dont le domaine peut, ensuite, avoir été étendu).

Nous avons également implémenté cette possibilité en offrant d'archiver, dans un entrepôt, toutes les versions de classes ayant existées dans la vie de l'entrepôt, ainsi que toutes les relations sous la forme originale.

Notons que le principe de la continuité ontologique semble rendre rarement nécessaire cet archivage complexe et coûteux. Pour résumer, nous présentons la structure complète d'un entrepôt de données à base ontologique dans un environnement multiversionné. Cette structure est constituée de trois parties :

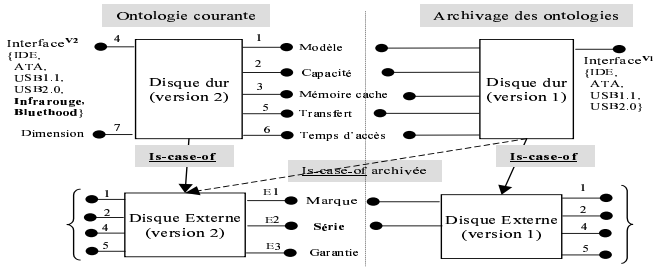

Figure 5: La partie d'ontologie d'un entrepôt historisé

1. L'ontologie courante : elle contient la version flottante de l'ontologie de l'entrepôt. Elle est également l'interface générique d'accès aux données.

2. L'archivage des ontologies : qui contient toutes les versions de chaque classe et propriété de l'ontologie d'entrepôt. Cette partie fournit aux utilisateurs les vraies définitions des versions de chaque concept si cela leur est nécessaire. Les versions du schéma de chaque table T_i sont également historisées en archivant la fonction $Sch^k(c_i)$ de chaque version k de c_i où c_i est une classe de T_i .

3. Les tables multiversionnées contiennent toutes les instances ainsi que leur première et dernière versions d'acitivité.

5 MISE EN OEUVRE

Nous avons implémenté notre proposition. L'architecture est décrite par la figure 6. Les ontologies et les BDBOs auxquelles nous nous intéressons sont celles définies par le modèle PLIB spécifié en langage EXPRESS. Elles sont donc échangeables sous forme de fichier d'instances EXPRESS ("fichier physique"). Pour créer les ontologies et les BDBOs, nous avons utilisé PLIBEditor (www.plib.ensma.fr) qui est un éditeur permettant non seulement de créer des ontologies et des BDBOs, mais également de les modifier (ou les faire évoluer). PLIBEditor est développé en Java et en ECCO (EXPRESS Compiler COmpiler). ECCO est un environnement de développement EXPRESS qui a la particularité de lire et d'écrire des fichiers d'instances EXPRESS et de permettre d'accéder à la description d'un schéma sous forme d'instances d'un méta-schéma.

Notre architecture d'entrepôt a été développée sous JBuilder et ECCO. JBuilder est utilisé pour créer des interfaces d'utilisateurs. ECCO sert à implémenter les APIs d'intégration. Ces APIs nous permettent:

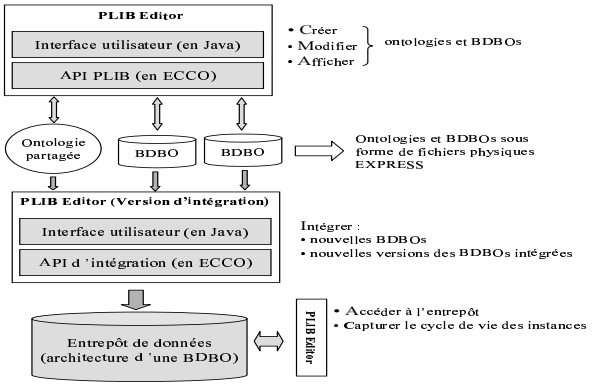

Figure 6: Prototype d'intégration et de gestion d'évolution des BDBOs

- d'une part, d'intégrer les différentes BDBOs selon les scénarii d'intégration définis dans (Bellatreche et al. 2004),

- d'autre part, d'intégrer des différentes versions d'une BDBO au sein de l'entrepôt en respectant le modèle présenté dans la section 4.

A titre d'exception, nous avons réalisé le scénario suivant : nous avons créé une ontologie partagée de disque dur et quatre catalogues référençant cette ontologie. Chaque catalogue possédait ses propres spécificités. L'ontologie partagée et les catalogues ont été crées en PLIBEditor en version 1. Ensuite tout est intégré au sein d'un entrepôt par "PLIBEditor-Version d'Intégration" en choisissant le scénario *ExtendOnto* (voir la section 2.2). Nous avons ensuite fait évoluer l'ontologie partagée et deux catalogues parmi les quatre qui auront la version 2. Un nouveau catalogue est créé en référençant la version 2 de l'ontologie partagée. Finalement, l'entrepôt est rafraîchi en intégrant ce nouveau catalogue et les deux catalogues modifiés. Les étapes d'intégration ci-dessus sont complètement automatiques. Une partie de cet entrepôt est présentée dans la figure 7.

A l'aide de PLIBEditor, nous pouvons également accéder aux instances de données dans l'entrepôt de deux manières : à travers la version courante de l'ontologie partagée, et *spécifique à chaque catalogue* soit en descendant la hiérarchie de l'ontologie courante de l'entrepôt ou soit directement (chaque catalogue intégré apparaissant également explicitement). Le cycle de vie de chaque instance est représenté explicitement dans la table de données (voir la figure 7).

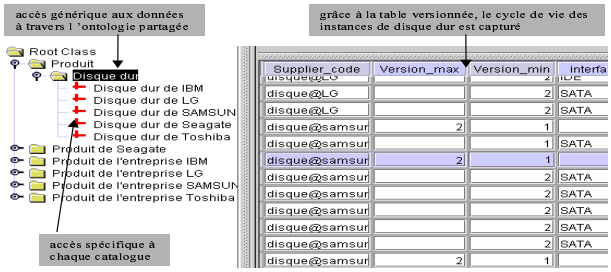

Figure 7: Affichage de l'entrepôt de disque dur dans PLIBEdior

6 CONCLUSION

Dans cet article nous avons présenté le problème d'évolution asynchrone des données et des ontologies dans un système d'intégration de type entrepôt de données. Nous considérons des sources de données à base ontologique, où chacune contient une ontologie locale qui référence une ontologie partagée de domaine. Ces sources sont indépendantes. Notre processus d'intégration intègre d'abord les ontologies ensuite les données. La présence de ces ontologies permet une automatisation du processus d'intégration et une résolution des conflits, mais rend la gestion d'évolution plus difficile. Cela est du à la prise en considération d'une nouvelle dimension d'évolution, à savoir les ontologies. Pour résoudre ce problème, nous avons présenté une approche multi-versionnée. D'abord nous avons défini des contraintes sur les ontologies et les données des sources. L'évolution des ontologies est réalisée selon le principe de continuité ontologique (une évolution d'une ontologie ne peut infirmer un axiome antérieurement vrai). Ensuite, une structure d'un entrepôt à base ontologique (qui référence également l'ontologie partagée) multi versionnée a été présentée. Elle est constituée de trois parties, à savoir, (1) l'ontologie courante qui contient la version courante de l'ontologie de l'entrepôt, (2) l'archivage des ontologies qui contient toutes les versions des définitions ontologiques de chaque classe et propriété, et (3) les tables multiversionnées contenant toutes les instances ainsi que leur première et dernière versions d'activité. Cette structure permet également de tracer le cycle de vie des instances de données. Notre modèle a été validé sous ECCO en considérant plusieurs ontologies de domaines, où pour chaque ontologie, un ensemble de sources a été défini. Des exemples de taille significative ont pu être traités.

Comme perspective, il serait intéressant (1) de considérer le problème de maintenance de vues sur un entrepôt à base ontologique multiversionné et (2) d'étudier la mise oeuvre de la même approche dans une logique médiateur.

References

Bebel, B., Eder, J., Koncilia, C., Morzy, T. & Wrembel, R. (2004). Creation and management of versions in multiversion data warehouse, *Proceedings of the 2004 ACM symposium on Applied computing* pp. 717–723.

Bellatreche, L., Pierra, G., Nguyen Xuan, D., Dehainsala, H. & Ait Ameur, Y. (2004). An a priori approach for automatic integration of heterogeneous and autonomous databases, *International Conference on Database and Expert Systems Applications (DEXA'04)* (475-485).

Chen, S., Liu, B. & Rundensteiner, E. A. (2004). Multiversion-based view maintenance over distributed data sources, *ACM Transactions on Database Systems* **4**(29): 675–709.

Doan, A., Noy, N. F. & Halevy, A. Y. (2004). Introduction to the issue on semantic integration, *SIGMOD Record* **33**(4).

Goh, C., Bressan, S., Madnick, E. & Siegel, M. D. (1999). Context interchange: New features and formalisms for the intelligent integration of information, *ACM Transactions on Information Systems* **17**(3): 270–293.

Gruber, T. (1993). A translation approach to portable ontology specification, *Knowledge Acquisition* **7**.

Heflin, J. & Hendler, J. (2000). Dynamic ontologies on the web, *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI/MIT Press)* pp. 443–449.

Minsky, M. (1979). Computer science and the representation of knowledge, *in The Computer Age: A Twenty-Year View, Michael Dertouzos and Joel Moses, MIT Press* pp. 392–421.

Pierra, G. & et al. (2005). Base de données à base ontologique : principe et mise en oeuvre, *Ingénierie des systèmes d'information, Hermès* **10**(2): 91–116.

Reynaud, C. & Giraldo, G. (2003). An application of the mediator approach to services over the web, *Special track "Data Integration in Engineering, Concurrent Engineering (CE'2003)* pp. 209–216.

Wei, H.-C. & Elmasri, R. (1999). Study and comparison of schema versioning and database conversion techniques for bi-temporal databases, *Proceedings of the Sixth International Workshop on Temporal Representation and Reasoning (IEEE Computer)* .