XeuTL : un outil ETL pour l'intégration de données

Cédric Gueydan

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

présenté par **Cédric Gueydan**

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en INFORMATIQUE

XeuTL : un outil ETL pour l'intégration de données

Soutenu le 30 juin 2010

JURY

Président : M. Eric Gressier-Soudan

Membres : M. Jean-Pierre Giraudin
M. André Plisson
M. Mathias Voisin-Fradin
Mme Ana Simonet
M. Michel Simonet

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

MEMOIRE

présenté par **Cédric Gueydan**

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en INFORMATIQUE

XeuTL : un outil ETL pour l'intégration de données

Soutenu le 30 juin 2010

Les travaux relatifs à ce mémoire ont été effectués au sein de l'équipe OSIRIS du laboratoire TIMC-IMAG de l'Université Joseph Fourier sous la direction de Ana et Michel Simonet.

Remerciements

Je désire tout d'abord remercier les membres du jury :

M. Eric Gressier-Soudan, Professeur au Conservatoire National des Arts et Métiers, qui me fait l'honneur de présider le jury ;

M. Jean-Pierre Giraudin, Professeur à l'Université Pierre Mendès-France et Responsable pédagogique et scientifique du 3e cycle d'ingénieur CNAM en informatique à Grenoble ;

M. André Plisson, Directeur du Conservatoire National des Arts et Métiers de Grenoble ;

M. Mathias Voisin-Fradin, Sous-directeur du CNAM de Grenoble.

Je tiens ensuite à remercier tout particulièrement Ana Simonet, Maître de Conférences à l'UPMF, et M. Michel Simonet, chercheur au CNRS et responsable de l'équipe OSIRIS au laboratoire TIMC, pour m'avoir accueilli au sein de leur équipe et m'avoir soutenu tout au long de la réalisation de ce travail.

J'adresse aussi mes remerciements à tous les membres de l'équipe OSIRIS que j'ai côtoyés durant mon stage.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations et acronymes

BAV : *Both As View*.

CRM : *Customer Relationship Management* ; gestion de la relation client.

CSV : *Comma-Separated Values* ; valeurs séparées par des virgules.

EAI : Enterprise Application Integration ; intégration d'applications d'entreprise (IAE).

EII : Enterprise Information Integration.

ERP : *Enterprise Resource Planning* ; progiciel de gestion intégré (PGI).

ETL : *Extract, Transform, Load* ; extraction, transformation, chargement.

GAV : *Global As View*.

LAV : *Local As View*.

OLAP : *Online Analytical Processing* ; traitement analytique en ligne.

OLTP : *Online Transaction Processing* ; traitement en ligne des transactions.

RDF : *Resource Description Framework*.

SBDF : Système de Bases de Données Fédérées.

SGBD : Système de Gestion de Base de Données.

SQL : *Structured Query Language* ; Langage de requêtes structuré.

XML : *Extensible Markup Language* ; langage de balisage extensible.

XSLT : *eXtensible Stylesheet Language Transformations*.

1 INTRODUCTION

1.1 Contexte

Ce mémoire a été réalisé au sein de l'équipe Osiris du laboratoire TIMC-IMAG[1] de l'Université Joseph Fourier. Cette équipe, dirigée par M. Simonet, s'intéresse à la modélisation et à la représentation des données et des connaissances. Un des axes de recherche concerne la conception de Systèmes d'Information, à travers le projet ISIS (Information System Initial Specification).
ISIS est une méthodologie et un outil d'aide à l'ingénierie et la réingénierie de bases de données. En conception initiale, ISIS part d'une définition ontologique du domaine puis, en s'appuyant d'une part sur des propriétés exprimées par le concepteur et d'autre part sur des règles de transformation de modèles, produit le schéma dans le modèle cible, l'API du noyau fonctionnel, et peut produire une interface prototype. Le logiciel ISIS, développé par l'équipe Osiris, est un outil CASE[2] initialement orienté vers la conception de bases de données. Ses fonctionnalités ont été étendues à la rétro-conception de bases existantes, avec l'objectif de l'enrichir d'outils pour l'intégration. C'est dans cette perspective que s'inscrit mon projet d'ingénieur CNAM.
Ce projet a été proposé en 2005 et j'ai travaillé six mois au sein de l'équipe Osiris à sa réalisation, mais une opportunité professionnelle exceptionnelle m'a fait interrompre ce travail. Pendant ce séjour au sein du laboratoire, j'avais réalisé une étude de la problématique de l'intégration qui est, en partie, reprise dans le chapitre Etat de l'Art de ce mémoire. Cette étude a été complétée par une partie sur le processus ETL[3] qui constitue la phase initiale des processus d'intégration dans l'approche matérialisée et qui est partie intégrante de toute opération de migration de données.
L'étude sur l'intégration était demandée par l'équipe à un moment où elle s'investissait dans ce type de travaux. Deux thèses sur ces thèmes ont été soutenues en 2009 : [Ahm09] et [Ker09]. Cette étude a également été intégrée à un état de l'art sur l'intégration de données dans le cadre du projet européen NOESIS : *Platform for a wide scale integration and visual representation of medical intelligence*[4]**.** Un extrait de cette étude est présenté en annexe 2.

La deuxième partie de mon projet concernait l'étude et la réalisation de modules contribuant à la construction d'un environnement logiciel pour l'intégration de bases de données. J'ai commencé assez naturellement par le processus ETL, car il intervient dans la première phase d'une démarche d'intégration utilisant l'approche matérialisée (les données des bases sources sont effectivement transformées, contrairement à l'approche virtuelle). L'équipe Osiris s'était trouvée confrontée à cette problématique dans plusieurs de ses projets, et a manifesté le besoin d'un outil « libre » et facile d'emploi. Il se trouve que mon activité professionnelle m'a

[1] (Techniques de l'Ingénierie Médicale et de la Complexité - Informatique, Mathématiques et Applications de Grenoble) – UMR 5525 CNRS-UJF
[2] Computer Aided Software Engineering
[3] Extract Transform Load
[4] http://www.noesis-eu.org

également amené à utiliser ce type d'outils et j'ai pu confirmer l'intérêt pour un outil « léger » et gratuit. J'ai donc conçu et réalisé un outil logiciel, XeuTL[5], qui répond à cette demande.

1.2 Motivations

L'utilisation des bases de données s'est répandue dans les entreprises dans le courant des années 60. Les besoins en intégration de données, liés à la nécessité de combiner et de questionner d'une manière homogène des données de plusieurs sources autonomes et hétérogènes, sont alors rapidement apparus. Ces besoins ont évolué au cours du temps mais sont toujours d'actualité : après trois décennies de recherches il n'existe toujours « pas de solution miracle[6] ». Les questions soulevées par l'article de P. Ziegler et KR. Dittrich[7] en 2004 (Three decades of Data Integration : all problems solved ?) demeurent actuelles.

Les ETL sont parmi les outils les plus utilisés lors de l'intégration. Ils permettent de mettre en place des processus d'intégration de données. Ces processus visent à interroger plusieurs sources de données hétérogènes, combiner les résultats obtenus et les charger dans une unique base cible. Les solutions ETL existantes présentent l'avantage de fournir une impressionnante quantité de fonctionnalités, mais aussi certains inconvénients non négligeables : lourdeur de la mise en place et de la maintenance, difficulté d'apprentissage, tarifs élevés, etc. Une solution plus légère, plus simple d'utilisation et moins coûteuse, même si elle possède moins de fonctionnalités, peut alors constituer une alternative intéressante dans certains contextes. Une telle demande était présente dans l'équipe Osiris, dans la perspective d'enrichir l'outil ISIS de fonctionnalités d'intégration et de migration de données.

1.3 Objectifs du stage

L'objectif du stage était double :

- effectuer une étude approfondie de l'état de l'art du domaine de l'intégration de données,
- développer des modules pour l'intégration, ce qui m'a conduit à réaliser un outil ETL[8].

Les entreprises génèrent des quantités de données de plus en plus importantes. Afin d'exploiter au mieux ces données, l'entreprise doit trouver des moyens pour les intégrer. Elle se heurte alors à la problématique de l'hétérogénéité de ces données, qui est au centre des recherches effectuées dans le domaine de l'intégration de données.

Pour répondre à cette problématique il existe deux approches : l'approche virtuelle et l'approche matérialisée.

Dans le cadre de ce stage, nous nous intéressons particulièrement à l'approche matérialisée à travers la réalisation d'un outil de type ETL. Un ETL réalise la tâche complexe d'intégration en extrayant les données de différentes sources, en les transformant (ou combinant) et en

[5] XeuTL (prononcer « gzeu-té-èl ») aurait pu s'appeler XETL (contraction de XML et ETL) mais afin d'éviter l'amalgame avec un langage existant nommé xETL[5], un 'u' a été ajouté

[6] « No silver bullet » dans le texte original. Expression empruntée à [http://www.virtualschool.edu/mon/SoftwareEngineering/BrooksNoSilverBullet.html]

[7] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.6593

[8] Extract, Transform, Load (Extraction, Transformation, Chargement)

chargeant les résultats obtenus dans une base cible. C'est un outil indispensable dans l'approche matérialisée.
La phase de transformation résout l'hétérogénéité des sources de données en se basant sur les correspondances définies entre les schémas sources et le schéma cible. En cela, elle est une étape capitale dans tout processus ETL. Dans le cadre de ce mémoire, nous accorderons une importance particulière à la mise en place de ces correspondances au sein de notre outil.

1.4 Organisation du mémoire

Le chapitre 2 dresse un état de l'art de l'intégration de données. Il présente les grandes directions des travaux effectués dans ce domaine et s'intéresse ensuite au cas spécifique de l'ETL.
Le chapitre 3 est une présentation de l'outil ETL réalisé : XeuTL. Celui-ci permet de définir et d'exécuter des processus ETL en utilisant des fichiers XML comme support d'échange.
Le chapitre 4 décrit les différentes correspondances implémentées dans XeuTL. La phase de transformation repose sur ces correspondances. C'est la phase la plus complexe d'un processus ETL.
Le chapitre 5 détaille la conception et la réalisation de XeuTL.
Le chapitre 6 propose un bilan du travail effectué et des perspectives envisageables pour XeuTL et pour l'intégration de données dans le cadre du projet ISIS.

2 ETAT DE L'ART

2.1 Motivation

Les technologies réseau actuelles (Intranet, Internet, etc.) permettent l'accès à une immense quantité de données stockées à différents emplacements physiques. Selon [CKT01], le principal intérêt de ces technologies pour l'utilisateur est la possibilité d'accéder à une multitude de sources de données entretenant un certain rapport les unes avec les autres, et de combiner ces données afin d'obtenir l'information désirée.

L'intégration de données a pour but, selon [Len02], de combiner les données réparties dans différentes sources et de fournir à l'utilisateur une vue unifiée de ces données.

L'intégration de données peut être réalisée en maintenant les données dans les sources de données (approche virtuelle, ou médiateur) ou en les stockant dans une nouvelle base de données (approche matérialisée).

Les sources de données ont, la plupart du temps, été conçues indépendamment les unes des autres. En conséquence, il existe de nombreuses différences entre ces sources. Celles-ci se situent à plusieurs niveaux :

- *l'hétérogénéité de haut niveau* : chaque source à intégrer modélise le monde réel de manière particulière. La représentation de données de sémantique similaire peut être différente d'une source à une autre. C'est l'hétérogénéité sémantique.
- *l'hétérogénéité de bas niveau* : les systèmes de gestion des sources proposent chacun leurs propres méthodes d'accès et de consultation de données. Ces méthodes ne sont pas nécessairement compatibles entre elles.

Un système d'intégration de données doit prendre en compte ces différentes formes d'hétérogénéité.

Le problème de l'hétérogénéité sémantique est défini dans [SK93] comme l'identification, dans les différentes sources de données, de données sémantiquement liées et la résolution des différences schématiques (différences en termes de représentation ou de structure) entre ces données[9].

Tout au long de ce mémoire, sauf précision explicite, le terme « hétérogénéité » fera référence à l'hétérogénéité sémantique.

Les données sémantiquement liées sont réparties dans plusieurs sources de données qui ont été conçues de différentes manières. Gérer cette hétérogénéité soulève plusieurs questions fondamentales :

[9] "the fundamental question is that of identifying objects in different databases that are semantically related and then resolve the schematic differences among semantically related objects"

- comment fournir une vue intégrée des données sémantiquement liées mais réparties dans des sources multiples ?
- comment conserver cette vue intégrée à jour ?
- comment identifier et spécifier les relations entre des données sémantiquement liées ?

L'intégration de données répond à ces questions.

2.2 L'intégration de données

2.2.1 Les approches

Quelques architectures pour les systèmes d'intégration de données sont présentées dans [Hul97]. Ces architectures permettent l'accès aux sources de données :

- soit en lecture seule,
- soit en lecture/écriture (mise à jour des sources de données).

Permettre l'accès en lecture/écriture dans un système d'intégration ne rentre pas dans le cadre du présent document. Ce mémoire traitera uniquement des systèmes d'intégration de données en lecture seule.

Il existe deux approches permettant l'accès aux données en lecture seule [Hul97]:

- l'approche « virtuelle » : le système accède aux données à la demande en décomposant la requête de l'utilisateur, en envoyant les sous-requêtes aux sources de données et en combinant les résultats obtenus. On parle alors de système médiateur.
- l'approche « matérialisée » : le système stocke dans un « dépôt » une copie des informations provenant des sources de données (ces informations sont, en général, filtrées). L'utilisateur effectue ensuite sa requête sur ce dépôt. L'exemple le plus courant de l'approche « matérialisée », est celui de l'entrepôt de données (data warehouse).

2.2.2 L'approche virtuelle

Il existe à l'heure actuelle deux architectures différentes qui suivent l'approche dite virtuelle : les systèmes de bases de données fédérées et les systèmes à base de médiateur.

2.2.2.1 Système de bases de données fédérées

Un système de bases de données fédérées (SBDF) consiste en des composants semi-autonomes (SGBD) qui participent à une fédération dans le but de partager partiellement des données entre eux [SL90]. Chaque source de données dans la fédération peut aussi opérer indépendamment des autres et de la fédération. Les composants ne peuvent pas être qualifiés de «complètement autonomes » car chacun est modifié par l'ajout d'une interface qui permet la communication avec les autres bases de données de la fédération. Un grand nombre d'interfaces doivent être écrits pour chaque source de données afin de communiquer avec les autres. Les systèmes fédérés ne sont pas très courants de nos jours.

2.2.2.2 Systèmes à base de médiateur

Le principe d'un système à base de médiateur est que l'utilisateur peut interroger un médiateur qui va manipuler et traduire sa requête afin d'interroger les différentes sources de données.

Le médiateur fournit une vue virtuelle des différentes sources de données hétérogènes considérées [CKT01]. Le but est de donner à l'utilisateur le sentiment qu'il interroge une unique source de données. En conséquence, l'utilisateur n'a besoin de connaître ni la localisation des sources de données, ni leurs schémas, ni leurs méthodes d'accès. L'utilisateur pose sa requête dans les termes du schéma global (le schéma du médiateur).

L'approche médiateur est basée sur un modèle de décomposition de requête, d'envoi de sous-requêtes aux sources de données et de combinaison des résultats obtenus.

2.2.2.2.1 Architecture

L'architecture proposée par R. Hull et R. King dans [HK95] définit trois couches fondamentales pour le support à l'intégration de données :

- les services de traduction[10] : les traducteurs, ou wrappers, gèrent l'hétérogénéité au niveau de la plateforme, et apportent aux interfaces non uniformisées (c'est le cas typique des systèmes « patrimoniaux »[11]) une forme plus standardisée (pas exemple une interface de requêtage commune).
- les services de médiation : les médiateurs sont chargés de gérer l'intégration sémantique de données.
- les services de coordination : pour une requête donnée, les « coordinateurs » fournissent un support automatique ou semi-automatique pour identifier des sources d'information pertinentes et pour l'identifier ou générer des médiateurs et traducteurs appropriés, de manière à ce que la requête puisse être traitée.

[10] wrapping services
[11] legacy system

La figure 1 montre une architecture typique de l'approche médiateur.

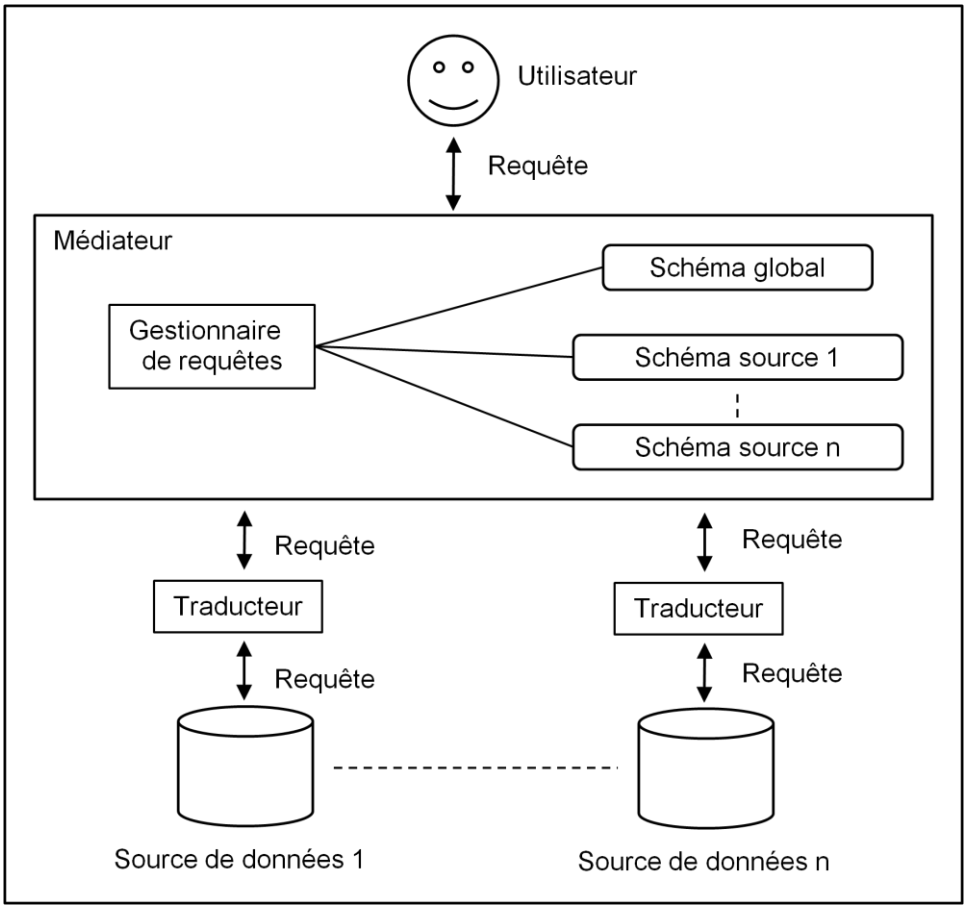


Figure 1: approche médiateur

Les principaux composants d'une architecture de type médiateur sont le médiateur, qui fournit les services de médiation et coordination, et les traducteurs de chaque source de données [CKT01].
La suite d'actions effectuées dans ce type d'architecture est la suivante :

- l'utilisateur formule une requête dans les termes du schéma global (schéma du médiateur),
- le médiateur décompose cette requête en sous-requêtes pour les différentes sources de données, en se basant sur les schémas locaux de ces sources de données,
- le médiateur optimise le plan d'exécution en se basant sur la description des sources de données
- le médiateur envoie les sous-requêtes aux traducteurs,
- chaque traducteur traduit la sous-requête qui lui est destinée en une requête sur le schéma de la source de données à laquelle il est associé (cette requête est exprimée dans le langage spécifique de la source de données),
- le traducteur reçoit la réponse de la source de données,
- le médiateur reçoit des traducteurs les réponses à ses sous-requêtes,
- le médiateur combine ces réponses de manière appropriée,
- l'utilisateur reçoit la réponse finale du médiateur.

2.2.2.2.2 La problématique de l'approche médiateur

Un schéma de médiateur est un ensemble de relations virtuelles qui ont été conçues pour une application d'intégration de données particulière. Le système d'intégration de données doit donc reformuler la requête de l'utilisateur en une requête qui se réfère directement aux schémas des sources de données.

Dans [LW00] A.Y. Levy et D.S. Weld classifient les problèmes adressés dans le domaine de l'intégration d'information comme ci-dessous.

Spécification de schéma médiateur et reformulation

Afin que le système soit capable de reformuler la requête d'un utilisateur, il a besoin de posséder un ensemble de descriptions des sources de données, spécifiant les correspondances sémantiques[12] entre les relations dans les sources de données et les relations dans le schéma médiateur. Les principales approches pour décrire les sources de données sont :

- global as view (GAV) : le schéma médiateur est décrit comme un ensemble de requêtes (ou de vues) sur les schémas des sources de données.
- local as view (LAV) : les sources de données sont décrites comme des requêtes sur les relations du schéma médiateur.
- logique de description : le schéma médiateur et les sources de données sont décrits comme une terminologie[13] grâce à des Logiques de Description (DL).

Différence de capacité dans le traitement des requêtes

Les sources de données web possèdent des capacités de traitement de requêtes extrêmement variées. Dans le cas d'un système d'intégration de données web, cette disparité peut faire apparaître de sérieux problèmes de performance :

- les données concernées peuvent être stockées dans des systèmes patrimoniaux[14] et l'interface est donc naturellement limitée,
- pour des données stockées dans des bases, il est courant que le site web ne dispose que de capacités d'accès limitées pour des raisons de sécurité ou de performance.

Optimisation de requête

Une fois que l'ensemble minimal de sources de données est sélectionné pour une certaine requête, un problème majeur est de trouver le plan d'exécution de requête optimal. Le plan d'exécution de requête spécifie l'ordre dans lequel on accède aux sources de données et les algorithmes particuliers utilisés pour combiner les données obtenues des sources de données. Ce problème semble identique au problème de l'optimisation des requêtes rencontré dans les systèmes de gestion de bases de données. Il est toutefois plus complexe dans le cadre de cette approche :

- il existe peu de statistiques concernant les sources de données sous-jacentes,
- il peut y avoir des retards significatifs dans la transmission des données, dus par exemple au trafic du réseau.

[12] semantic mappings

[13] l'ensemble des termes, rigoureusement définis, qui sont spécifiques d'une science, d'une technique, d'un domaine particulier de l'activité humaine. [wikipedia]

[14] legacy system

2.2.2.2.3 Lien entre schéma global et schémas locaux

Il existe principalement deux manières différentes d'établir un lien entre le schéma global et les schémas des sources de données : l'approche global-as-view (GAV), aussi appelée l'approche global-schema-centric, et l'approche local-as-view (LAV), aussi appelée source-centric [Len03].

L'approche GAV consiste à exprimer le schéma global dans les termes des schémas des sources de données alors que l'approche LAV consiste à exprimer les schémas des sources de données dans les termes du schéma global [CCG02].

Voici un exemple extrait tiré de [Len03] illustrant ces deux approches :

Le système est basé sur un médiateur et deux sources de données : la première source de données (r1) contient des films de réalisateurs européens de 1960 à nos jours, la seconde source de données (r2) contient des critiques de films de 1990 à nos jours. La figure 2 représente les schémas de ce système.

```
Schéma global:
movie(Title;Year; Director)
european(Director)
review(Title; Critique)

Source de données 1:
r1 (Title;Year; Director )

Source de données 2:
r2 (Title;Critique)
```

Figure 2: exemple – schémas généraux

Un exemple SQL d'une requête utilisateur posée à ce système est présenté figure 3.

```
Titres et critiques de films en 1998:

SELECT Title, Critique
FROM movie JOIN review
WHERE Year=1998
```

Figure 3: exemple – requête utilisateur

La figure 4 montre une règle de correspondance en termes de vues entre le schéma global et les schémas des sources de données selon le type d'approche utilisé (GAV ou LAV).

Règles de correspondances GAV :

```
CREATE VIEW movie (Title; Year; Director) AS
SELECT Title, Year, Director FROM r1

CREATE VIEW european (Director) AS
SELECT Director FROM r1

CREATE VIEW review (Title; Critique) AS
SELECT Title, Critique FROM r2
```

Règles de correspondances LAV :

```
CREATE VIEW r1 (Title; Year; Director) AS
SELECT Title, Year, Director
FROM movie JOIN european
WHERE Year >= 1960

CREATE VIEW r2 (Director) AS
SELECT Title, Critique
FROM movie JOIN review
WHERE YEAR >= 1990
```

Figure 4: exemples de règles de correspondance

2.2.2.2.4 Reformulation de requête

La requête utilisateur exprimée dans les termes du schéma global doit être reformulée en une requête exprimée dans les termes des schémas des sources de données [Kno03].

La reformulation de la requête doit répondre à deux critères :

- exactitude sémantique de la reformulation : les réponses des sources de données sont des réponses correctes à la requête originale (globale),
- minimisation de l'accès aux sources de données : seules les sources pouvant effectivement contribuer à la réponse finale doivent être consultées.

La reformulation de requête dépend du type de correspondance utilisé entre le schéma global et les schémas des sources de données :

- avec la correspondance GAV, la reformulation de requête est directe et accomplie par « dépliage » de la requête,
- avec la correspondance LAV, la reformulation de requête est plus complexe et nécessite l'emploi d'algorithmes utilisant l'inférence.

La figure 5 montre un exemple de reformulation de requête selon la correspondance GAV [Len03]. La requête utilisateur est traitée au moyen du « dépliage » de la requête ; chaque atome est « déplié » selon sa définition associée dans les règles de correspondance définies entre le schéma global et les schémas des sources de données.

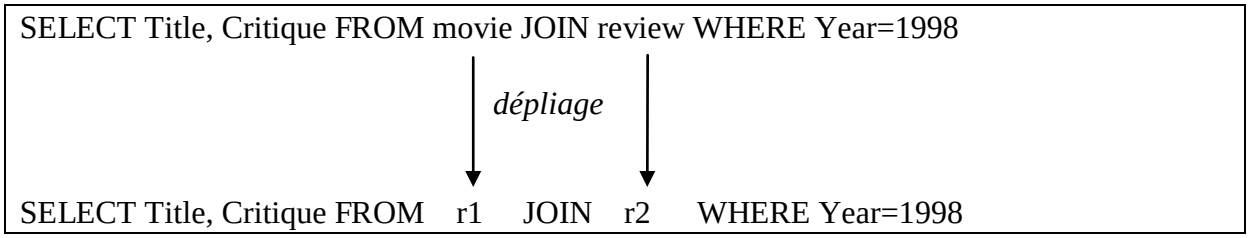


Figure 5: reformulation de requête (GAV)

La figure 6 montre un exemple de reformulation de requête selon la correspondance LAV. La requête utilisateur est traitée au moyen d'un mécanisme d'inférence qui vise à exprimer les atomes du schéma global dans les termes d'atomes de sources.

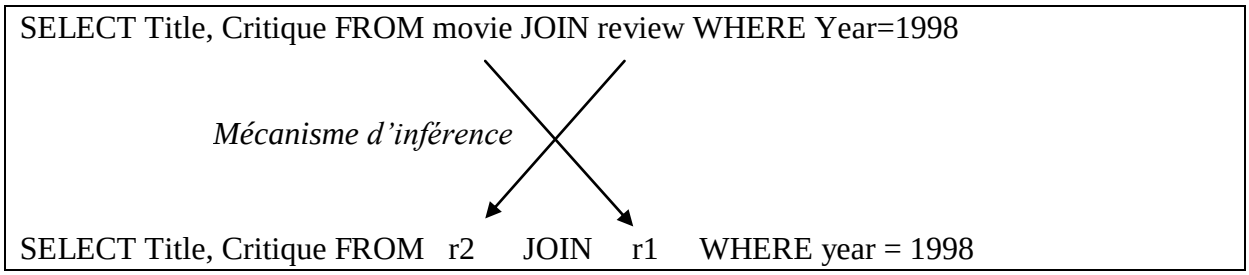


Figure 6: reformulation de requête (LAV)

Il existe des algorithmes permettant de résoudre le problème de la reformulation de requête en LAV : bucket, inverse-rules, minicon, shared-variable-bucket, corecover. Ils sont détaillés dans [CKT01].

2.2.2.2.5 Approche GAV et approche LAV

Les principales caractéristiques de l'approche GAV sont les suivantes :

- l'approche GAV n'est pas modulaire : l'ajout ou la modification des sources de données nécessite des modifications du schéma global,
- la reformulation de requête est simple, elle se réduit à un « dépliage » des vues.

Les principales caractéristiques de l'approche LAV sont les suivantes :

- l'approche LAV est modulaire : l'ajout ou la modification des sources de données est simple (les sources sont définies indépendamment les unes des autres)
- la reformulation de requête est complexe.

L'approche GAV est en général plus appropriée aux systèmes utilisant des sources de données stables tandis que l'approche LAV convient à des systèmes utilisant beaucoup de sources de données relativement « inconnues » du système et nécessitant l'addition, la modification ou la suppression de sources.

2.2.3 L'approche matérialisée

2.2.3.1 Généralités

Le principe de l'approche matérialisée est que l'utilisateur interroge une seule base (la base cible) contenant une copie des données provenant de différentes sources de données [CKT01]. La figure 7 montre un exemple d'architecture de ce type.

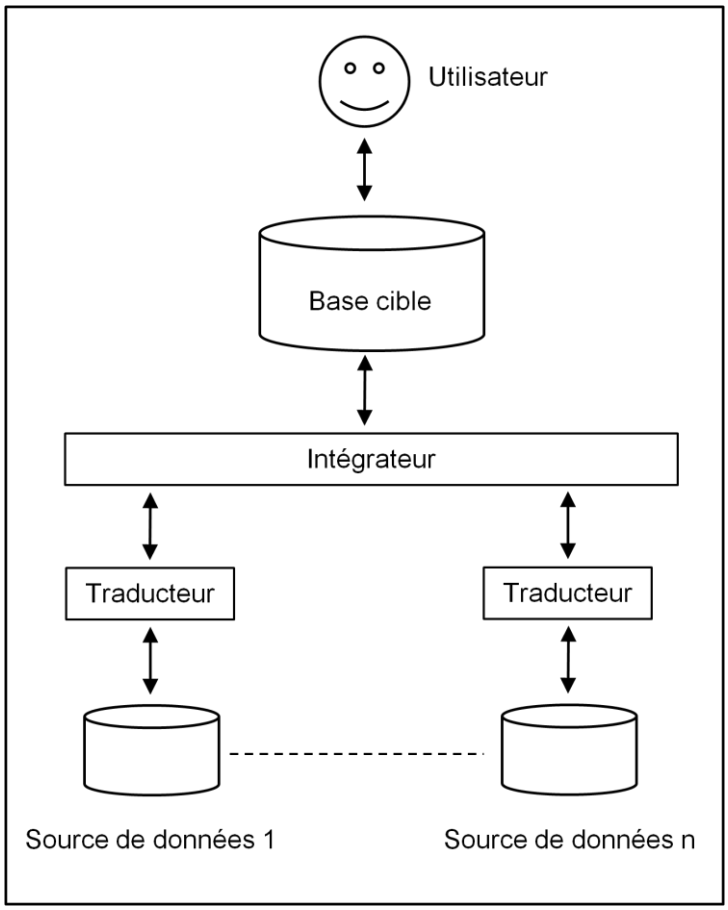


Figure 7: approche matérialisée

Comme dans le cas de l'approche virtuelle, chaque source de données est liée à un traducteur, ou wrapper. Au niveau supérieur, un intégrateur est chargé d'alimenter la base cible avec une copie des données issues des sources selon le schéma global défini pour cette base. L'utilisateur interroge la base cible, et donc la copie des données intégrées des sources de données, sans avoir à connaître le schéma ou les méthodes d'accès propres à chaque source de données.

Une des principales problématiques liée à la matérialisation est la conservation des données de la base cible à jour. Selon les besoins, ces données sont généralement mises à jour sur des bases périodiques : mensuelles, hebdomadaires, quotidiennes, horaires, ….
Les mises à jour peuvent être effectuées au moyen de différentes techniques, dont les plus courantes sont :

- le rafraîchissement total : la base cible est entièrement vidée puis repeuplée, ou réalimentée, avec des données à jour,

- la réalimentation incrémentale : la base cible est peuplée une fois à sa création ; chaque mise à jour contient ensuite uniquement les données qui diffèrent entre le moment de la mise à jour et celui de la précédente alimentation,
- l'historisation puis la réalimentation : plutôt que de vider la base cible, les données sont historisées (datées et/ou marquées), puis, à la manière du rafraîchissement total, la réalimentation est complète.

La base cible peut être une base de données standard (OLTP) ou un entrepôt de données (OLAP), qui est un type particulier de base de données, et ainsi posséder des propriétés particulières qui sont détaillées dans la section suivante.

2.2.3.2 L'entrepôt de données

L'entrepôt de données est la forme la plus courante d'intégration suivant l'approche matérialisée. Les entrepôts de données sont aujourd'hui présents dans de très nombreuses entreprises. Ils permettent de collecter et stocker les données des différents système d'information de l'entreprise. L'information peut ensuite être extraite, analysée et présentée en vue d'offrir une aide à la décision et de permettre aux responsables de la stratégie d'entreprise d'avoir une vue d'ensemble de l'activité traitée. Les entrepôts de données sont généralement utilisés pour transformer des données de production en données stratégiques.

Ce type de système est principalement employé pour l'aide à la décision. Les banques utilisent des entrepôts de données pour décider des investissements futurs, les commerces peuvent reposer sur des entrepôts de données hébergeant les données des ventes de produits pour mieux cibler les besoins de leurs clients etc. En conséquence un entrepôt de données contient souvent des données historisées et/ou résumées [CKT01]. L'entrepôt de données doit supporter des charges de travail intensives dues aux requêtes complexes qu'il traite de manière rapide et efficace. Les utilisateurs accèdent en général à l'information contenue dans un entrepôt de données en lecture seule.

La réalisation et la maintenance d'un entrepôt de données soulèvent plusieurs questions :

- modélisation et conception :
  - quelles sont les informations de chacune des sources de données qui vont être stockées dans l'entrepôt ?
  - quelles sont les requêtes que l'entrepôt devra traiter ?
  - quel sera le schéma global de l'entrepôt de données ?
- maintenance :
  - comment alimenter initialement l'entrepôt de données?
  - comment le rafraîchir lorsque les sources de données sont mises à jour ?
- opérationnel :
  - comment gérer le traitement des requêtes, le stockage et l'indexation ?

Les réponses à ces questions sont très étroitement liées aux besoins de l'entrepôt. Les nombreuses solutions et techniques ne seront pas abordées dans ce mémoire.
Les publications du Kimball Group sont une base de référence en la matière : [NET01].

2.2.3.3 Différences entre bases de données standard et entrepôts de données

Les bases de données standard sont principalement utilisées pour effectuer des traitements en ligne des transactions (OLTP : On-Line Transaction Processing). Les entrepôts de données sont basés sur le concept OLAP (On-Line Analytical Processing) dont le but est l'efficacité dans les opérations de collecte, de stockage, de traitement, et de restitution de données multidimensionnelles à des fins d'analyse.

Il est important de noter que le terme « entrepôt de données » désignait initialement une base de données spécifiquement conçue autour du concept OLAP. L'entrepôt de données était alors destiné à des fins d'analyse et possédait alors cette structure de données multidimensionnelle qui lui est propre. Au fil des années, le terme « entrepôt de données » s'est étendu à toute base cible d'un processus d'intégration dans une approche matérialisée, indépendamment de sa finalité et de sa structure. Pour lever toute ambiguïté, nous utilisons, dans le cadre de ce mémoire, le terme « entrepôt de données » dans sa signification initiale : une base de données basée sur une structure multidimensionnelle et destinée à l'aide à la décision. Le terme « base cible », désigne la base cible d'un processus d'intégration, celle-ci pouvant être un entrepôt de données (cas OLAP) ou une base de données standard (cas OLTP).

Contrairement à une base de données standard, les données d'un entrepôt sont représentées sous une forme multidimensionnelle afin d'obtenir une visualisation rapide et intuitive. Une des dimensions est souvent la dimension temporelle. Cette représentation multidimensionnelle est appelée représentation en cube. La figure 8 présente un exemple de cube permettant d'effectuer des analyses des ventes de certaines catégories de produits par région et par année.

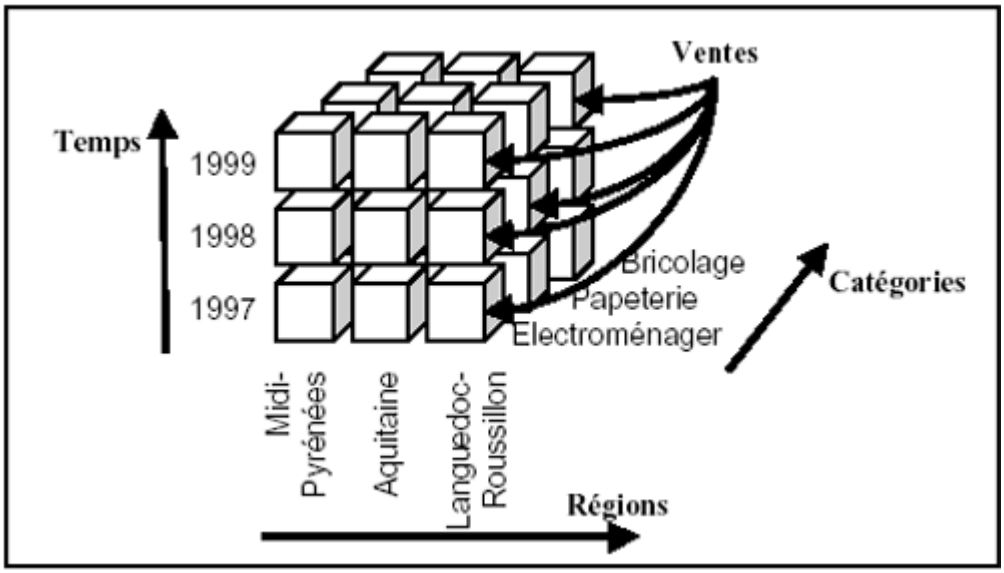


Figure 8 : exemple de cube OLAP [NET02]

Il existe différentes modélisations conceptuelles pour un entrepôt de données. Elles ont en commun les notions de tables de dimensions et tables de faits:

- les tables de dimensions contiennent les critères à évaluer lors de l'analyse (exemple : le temps, les produits, …),
- les tables de faits contiennent les données à analyser, couramment appelées « mesures ». Elles sont souvent stockées dans des champs numériques (exemple : des volumes de ventes).

Dans la modélisation en étoile, les tables de dimensions sont reliées à une table de faits. La modélisation en flocon est une variante de la modélisation en étoile ou l'on créé des hiérarchies de dimensions pour des raisons de performance.

Le tableau 1, inspiré de [NET03], présente les principales différences entre les concepts OLTP (bases de données standard) et OLAP (entrepôt de données).

	OLTP	OLAP
Types d'utilisateurs	- employés de bureau - informaticiens - …	- analystes - décisionnaires - statisticiens - …
Fonction	support d'opérations quotidiennes	aide à la décision
Conception des bases	orientée métier ou application	orientée sujet
Propriétés des données hébergées	- mise à jour continue - détaillées - isolées - description selon une approche relationnelle	- mise à jour périodique - historisées - résumées - intégrées - consolidées - description selon une approche multidimensionnelle
Accès	lecture/écriture	lecture seule (scan)
Requêtes	courtes, simples	complexes
Nombre d'enregistrements accédés	plusieurs dizaines, centaines ou milliers	plusieurs millions
Nombre d'utilisateurs	Quelques centaines à milliers	quelques dizaines à centaines

Tableau 1 : principales différences entre OLTP et OLAP

2.2.4 Comparaison des approches

Hull distingue deux directions majeures en intégration de données en lecture seule : l'une basée sur la Closed World Assumption (l'union des sources de données équivaut à l'ensemble des données significatives), l'autre basée sur l'Open World Assumption (l'union des sources de données est incluse dans l'ensemble des données significatives) [Hul97]. Dans le cas de la CWA, le schéma global est généralement construit de manière ascendante (bottom-up), en partant des schémas des sources de données (méthode GAV). Dans le cas OWA, l'approche descendante (top-down) est utilisée : un schéma global contenant toute les informations significatives est créé, et les données contenues dans les sources sont exprimées par des vues sur ce schéma global (méthode LAV).

La figure 9 illustre les différentes directions de recherche existantes.

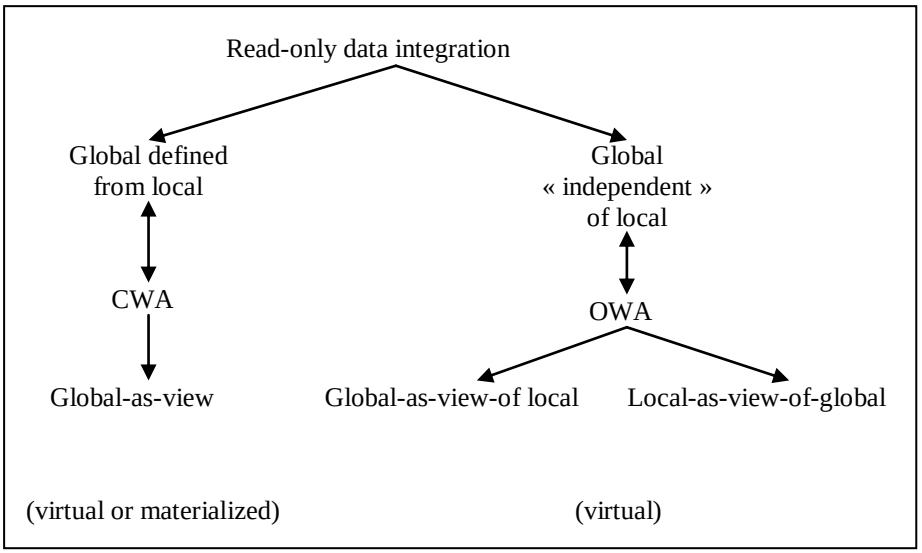


Figure 9: les directions de recherche existantes [Hul97]

Le tableau 2 présente les avantages et les inconvénients de l'approche médiateur.

Approche médiateur	Avantages
	- l'alimentation d'une base cible n'est pas nécessaire (il n'existe pas de base cible) - les données sont toujours à jour car les réponses aux requêtes de l'utilisateur sont directement tirées des bases sources
	Inconvénients
	- les temps de réponse peuvent être longs **:** - les requêtes effectuées sur la cible sont traduites dans le langage de la source de données puis exécutées sur le serveur des bases sources - le serveur des bases sources peut être amené à traiter en parallèle des requêtes du médiateur et des requêtes opérationnelles - la gestion de l'hétérogénéité s'effectue en temps réel - il existe des possibilités d'incohérences dues à d'éventuelles données redondantes sur différentes sources de données ou encore à des traitements concurrents mal contrôlés - la mise en place d'un médiateur est une tâche complexe - l'approche est inadaptée aux requêtes nécessitant des agrégations

Tableau 2 : avantages et inconvénients de l'approche médiateur

Le Tableau 3 présente les avantages et les inconvénients de l'approche matérialisée.

Approche matérialisée	Avantages
	- les temps de réponses sont courts car l'utilisateur interroge une seule base de données (la base cible) - les réponses sont sûres car il n'existe pas d'incohérence liée au données redondantes sur différentes sources ou aux traitements concurrents - les techniques sont éprouvées car cette approche, notamment l'entrepôt de données, est utilisée par les entreprises depuis plusieurs années (le terme « entrepôt de données » date du début des années 90) - l'approche est adaptée aux requêtes nécessitant des agrégations
	Inconvénients
	- le processus d'alimentation est long et coûteux en termes de ressources réseaux et système - les données datent de la dernière alimentation (ou réalimentation) effectuée

Tableau 3 : avantages et inconvénients de l'approche matérialisée

En règle générale l'approche virtuelle est préférable à l'approche matérialisée dans les cas suivants [CKT01] :

- le nombre de sources de données dans le système d'intégration est très grand, évolutif, et/ou les sources sont susceptibles d'être mises à jour fréquemment (comme dans le cas des sources web),
- il n'existe aucun moyen de prédire les types de requêtes que l'utilisateur va formuler.

L'approche matérialisée est préférable lorsque :

- les sources de données sont permanentes,
- les sources de données ne sont pas mises à jour trop souvent,
- les concepteurs du système d'intégration connaissent, en général, les types des requêtes qui vont être effectuées par les utilisateurs,
- de faibles temps de réponse sont nécessaires.

2.2.5 Autres axes de recherche

2.2.5.1 Les ontologies

2.2.5.1.1 Introduction

« Ontologie » est un terme utilisé pour se référer à la compréhension partagée d'un certain domaine d'intérêt [UG96]. Une ontologie exprime un point de vue sur le monde réel et concerne en général un domaine donné. Cette vue est souvent composée d'un ensemble de concepts, de leurs définitions et de leurs interrelations. Une ontologie est une spécification explicite d'une conceptualisation [Gru93].

Une ontologie peut prendre différentes formes, mais inclura nécessairement un vocabulaire de termes et les spécifications de leurs significations. L'utilisation des ontologies pour la description de connaissances implicites et cachées est une approche possible pour surmonter le problème de l'hétérogénéité sémantique [WVV01].

[MAB03] présente les principaux bénéfices d'une approche basée sur les ontologies dans le cadre de l'intégration de données :

- la capacité à décrire toutes les structures de données rencontrées fait certainement des ontologies le modèle de représentation de connaissance le plus avancé de nos jours ;
- la combinaison de systèmes de déduction et de bases de données relationnelles, amène les capacités de correspondance et de logique applicative à un plus haut degré d'abstraction, le modèle étant séparé du stockage des données ;
- l'extension du système est aisée et ce dernier bénéficie d'une bonne réutilisabilité.

2.2.5.1.2 Le rôle des ontologies

Les ontologies sont utilisées en particulier pour décrire explicitement la sémantique de sources d'information. Différentes approches existent [WVV01] :

- l'approche *ontologie simple* : toutes les sources d'information sont liées à une seule ontologie globale. Cette technique est en général utilisée quand les sources d'information à intégrer fournissent quasiment la même vue d'un domaine ;
- l'approche *ontologies multiples* : chaque source d'information est décrite au moyen d'une ontologie qui lui est propre. Ceci simplifie l'intégration et permet aisément la modification dans les sources ;
- l'approche *hybride* : la sémantique de chaque source est décrite au moyen d'une ontologie qui lui est propre, mais afin de pouvoir comparer les différentes ontologies entre elles, celles-ci sont construites à partir d'un vocabulaire partagé global qui peut lui-même être une ontologie (les ontologies des sources sont comparables grâce à ce vocabulaire partagé).

La figure 10 illustre ces trois architectures.

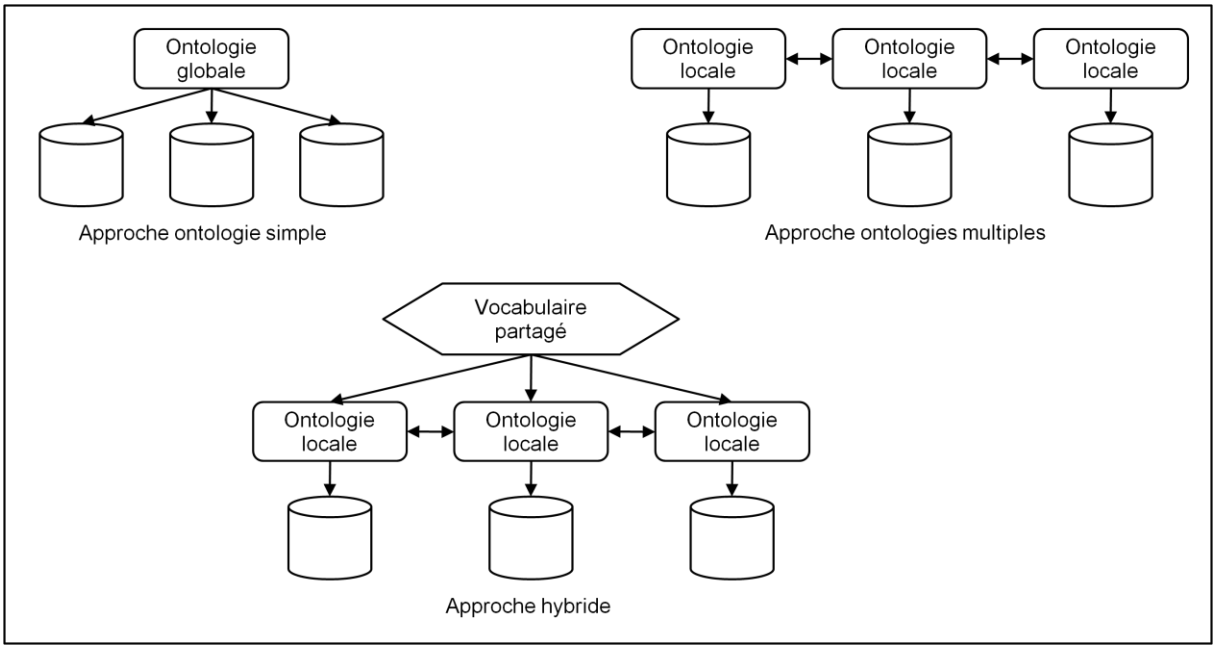


Figure 10 : les trois architectures d'ontologies pour l'intégration [WVV01]

Les ontologies peuvent aussi être utilisées :

- comme modèle de requête global : l'utilisateur formule alors ses requêtes dans les termes de l'ontologie,
- pour la vérification des descriptions de l'intégration : l'exactitude des correspondances spécifiées entre le schéma global et les schémas locaux peut être vérifiée automatiquement en s'appuyant sur des ontologies.

2.2.5.1.3 Les techniques liées aux ontologies

Il existe différentes manières de représenter une ontologie (en utilisant des logiques de descriptions, les systèmes à base de frames, …), d'établir des correspondances entre des ontologies et des sources d'information (en utilisant la ressemblance de structures, la définition des termes, l'enrichissement de structure, les méta-annotations), et d'établir des correspondances entre les ontologies (définition de correspondances, relations lexicales, correspondances sémantiques). On pourra se référer à [WVV01] pour avoir plus de détails sur ces techniques.

2.2.5.1.4 Conclusion

Les auteurs de [SWV01] précisent que certaines questions restent ouvertes dans ce domaine :

- les correspondances entre ontologies ne sont pas encore bien fondées et s'effectuent plutôt de manière « ad hoc ou arbitraire »,
- il existe un besoin d'investigation sur des bases théoriques et empiriques,
- il y a un manque de méthodologies concernant le développement et l'utilisation des ontologies,
- la méthodologie devrait être indépendante de la langue.

Les ontologies sont utilisées par de nombreux systèmes d'intégration (par exemple MOMIS, présenté en annexe 2) et aident à résoudre l'hétérogénéité sémantique. Ces systèmes d'intégration sont, la plupart du temps, basés sur une approche virtuelle.

2.2.5.2 Le web sémantique

Le W3C présente le web sémantique[15] comme une extension du web actuel, extension dans laquelle une signification précisément définie est donnée à l'information afin de permettre aux ordinateurs et aux utilisateurs de travailler en coopération. Le web sémantique est basé sur l'idée d'avoir des données web définies et liées entre elles de manière à permettre des découvertes, des automations, des intégrations, des réutilisations plus efficaces au travers de diverses applications. Afin que le web atteigne son potentiel maximum, il doit évoluer vers le web sémantique, et ainsi fournir une plateforme universellement accessible qui permette aux données d'être partagées et traitées autant par les utilisateurs que par des outils automatisés.
Les récentes avancées dans les domaines du web sémantique tels XML[16], RDF[17], les agents intelligents ou les ontologies[18] devraient hautement faciliter la tâche d'intégration de données.

2.2.5.3 Les architectures P2P

De nos jours les architectures peer-to-peer (aussi appelées « p2p », « poste à poste » ou « pair à pair ») sont couramment utilisées pour des applications de type partage de fichiers. Dans ces systèmes, chaque poste (peer) agit à la fois comme client et comme serveur, en fournissant à l'environnement distribué une partie de l'ensemble de ses informations disponibles, sans s'appuyer sur une vue globale unique. Ce type d'architecture soulève de nouvelles questions dans le domaine de l'intégration de données.

Les systèmes « peer-to-peer » actuels sont encore relativement limités (granularité élevée, absence de sémantique à tous les niveaux) et ne profitent pas au mieux des différentes techniques de distribution de données. Ces systèmes de partage sont limités à des applications dans lesquelles les objets sont décrits par leur nom et présentent de fortes limitations dans l'établissement de liens complexes entre les points.

Contrairement à la plupart des systèmes d'intégration de données, l'intégration dans les systèmes peer-to-peer n'est pas basée sur un schéma global. En effet, chaque point représente un système d'information autonome, et l'intégration d'information est effectuée en établissant des mappings peer-to-peer (mapping entre divers postes). Les requêtes sont posées à un poste, et le rôle du traitement de requête est d'exploiter à la fois les données internes au poste et les mappings avec les autres postes du système.

Les articles suivants présentent différentes techniques en intégration de données « peer-to-peer » (telles que les techniques GLAV ou BAV combinant les approches LAV et GAV) : [MBP03], [Ive03a], [PMT03], [HIS03], [FKL04], [Len03], [Ahm09], [Ker09].

[15] http://www.w3.org/2002/07/swint
[16] http://www.w3.org/XML/
[17] http://www.w3.org/RDF/
[18] http://www.w3.org/2001/sw/WebOnt/

2.3 Les ETL

2.3.1 Introduction

Au fil du temps, les systèmes d'information des entreprises ont tendance à se complexifier :
- les technologies évoluent et se diversifient,
- les nouveaux systèmes d'information doivent collaborer avec les anciens,
- les sources de données se multiplient.

De très nombreuses entreprises sont amenées à faire face à la problématique de l'intégration des données réparties dans leur système d'information. L'approche matérialisée est souvent utilisée dans des cas courants d'intégration :
- migration de données d'un ancien système vers un système plus récent. Un processus de migration de données n'est exécuté qu'une fois. Suite à la migration, la base source n'est plus utilisée.
- alimentation d'une base cible avec des données provenant d'une ou plusieurs bases sources. La base cible est une base standard ou un entrepôt de données. Contrairement à la migration, les bases sources sont toujours utilisées par des applications.

Les outils les plus communément utilisés dans ce domaine sont appelés des ETL (Extract, Transform, Load ou Extraction, Transformation, Chargement). Ce type d'outil est destiné à extraire les données de diverses sources (bases de données, fichiers, etc.), à les transformer et à les charger dans une base cible, permettant ainsi d'effectuer des synchronisations massives d'information d'une base de données vers une autre.

2.3.2 Historique

Les besoins des entreprises en intégration de données ont toujours été en constante augmentation et les ETL ont su évoluer en fonction de cette évolution. On distingue quatre périodes dans l'évolution des ETL [NET04] :
- Avant les années 90 chaque processus ETL est codé manuellement. Le développeur doit faire appel à plusieurs langages et différentes technologies pour mettre en place ces processus. Ceux-ci sont flexibles car le développeur code le processus de A à Z. Les limitations rencontrées sont celles liées aux technologies utilisées. Les inconvénients sont multiples : réalisation coûteuse et peu réutilisable, maintenance complexe, compatibilité limitée, etc.
- Au milieu des années 90, la première génération d'outils ETL voit le jour. Beaucoup de ces outils génèrent du code (souvent COBOL) pour extraire des données stockées dans des mainframe. La génération automatique du code permet d'éviter la lourde tâche de codage manuel des processus ETL. Les ETL sont performants avec les systèmes patrimoniaux mais nécessitent des connaissances approfondies en C ou en COBOL et ne sont pas adaptés pour les bases de données relationnelles.
- A la fin des années 90 apparaît la seconde génération d'outils ETL. Ceux-ci ne génèrent plus de code mais s'appuient sur des moteurs internes capables de gérer les processus ETL définis dans l'application. Les performances sur de gros volumes de données se trouvent diminuées du fait que le flot complet de données doit être traité par le moteur de l'outil ligne par ligne (le moteur devient le goulot d'étranglement du système). Les interfaces graphiques et les transformations font leur apparition, démocratisant l'utilisation de ces outils.

- Depuis 2003 les ETL proposent de nombreuses fonctionnalités avancées : gestion de multiples sources de données en entrée et en sortie, et de différents types de sources (bases de données relationnelles, fichiers plats, …), transformations complexes (requêtes d'agrégation, …), parallélisme offrant de meilleures performances... Les interfaces graphiques sont plus accessibles. Les ETL basés sur de la génération de code produisent non plus du C ou du COBOL mais du SQL qui peut être exécuté sur de multiples plateformes.

2.3.3 Caractéristiques et fonctionnalités

2.3.3.1 Les phases d'un processus ETL

2.3.3.1.1 Extraction

L'extraction permet d'extraire les données depuis les sources de données de production de l'entreprise (CRM, ERP, SBGD métier, …). L'extraction des données peut s'effectuer selon deux méthodes :

- à l'aide de triggers (méthode push) : ceux-ci sont déclenchés lors de modifications sur les bases sources et « poussent » les données modifiées vers l'outil ETL. Cette méthode nécessite d'intervenir au niveau des bases sources pour mettre en place ces triggers ce qui n'en fait pas la méthode la plus répandue ;
- à l'aide de requêtes (méthode pull) : le système ETL interroge les bases sources pour extraire les données. Cette méthode est la plus répandue car elle ne nécessite pas la modification des sources de données pour mettre en place les processus ETL.

L'extraction peut considérablement charger les systèmes de gestion des bases de données opérationnelles et perturber les applications OLTP.

2.3.3.1.2 Transformation

La phase de transformation des données permet d'effectuer différentes opérations sur celles-ci afin de les concilier et d'obtenir un format qui respecte celui de la base cible. L'utilisateur définit des correspondances entre les schémas des bases sources et de la base cible. L'ETL s'appuie sur ces correspondances pour appliquer les transformations nécessaires sur les données et résoudre ainsi l'hétérogénéité sémantique.

La tendance actuelle est de proposer une interface graphique permettant de définir visuellement les correspondances.

Les transformations sont le cœur d'un ETL.

2.3.3.1.3 Chargement

Le chargement permet de charger les données obtenues à l'issue de le phase de transformation vers la base cible (entrepôt de données ou autre). La réalimentation d'une base cible (ou rafraîchissement) est un procédé un peu différent de l'alimentation (ou chargement) initiale. Les données anciennes (déjà présentes dans la base cible) ne sont jamais mises à jour avec les nouvelles données, mais archivées ou supprimées. L'insertion des nouvelles données ne doit pas modifier les données existantes. Une technique pour garantir la cohérence des données consiste à générer de manière automatique pour les nouvelles données les valeurs des clés des tables de la base cible au moment du chargement.

2.3.3.1.4 Nettoyage

Selon les besoins, une phase de nettoyage de données est réalisée. Celle-ci est effectuée conjointement à la phase de transformation. Elle vise à améliorer la qualité des données à transférer vers la base cible.
Les données « à nettoyer » sont :

- les doublons,
- les données erronées,
- les valeurs manquantes,
- …

Les techniques employées sont :

- le rejet des données,
- le dédoublonnage,
- l'introduction de valeurs fixes, moyennes, …
- …

2.3.3.2 Volumétrie des données

Les processus ETL travaillent sur de gros volumes de données. Les outils doivent être capables de gérer cette importante volumétrie en s'appuyant sur des techniques particulières qui permettent d'améliorer leurs performances :

- la technique du streaming consiste à transférer les données en flux continu,
- la technique du parallélisme consiste à utiliser plusieurs processeurs ou threads pour une tâche.

2.3.3.3 Mode batch

Les processus ETL sont gourmands en termes de ressources. Ils fonctionnent la plupart du temps en mode batch : le processus ETL complet est programmé pour être exécuté à des moments où l'impact sera réduit pour les utilisateurs de ces mêmes ressources. Ces processus ETL se déroulent souvent la nuit ou le week-end pour ne pas gêner les utilisateurs qui partagent ces mêmes ressources (réseau ou sources de données). Les données sont potentiellement moins soumises à modification durant ces périodes.

2.3.3.4 Cas de l'entrepôt de données

Les ETL utilisés pour l'alimentation d'entrepôts de données fournissent des fonctionnalités particulières adaptées aux caractéristiques spécifiques des entrepôts.

2.3.3.4.1 Marquage et datation des données

Les données chargées doivent être marquées et datées. A tout moment le système peut identifier la provenance de données et connaître leur date de chargement.

2.3.3.4.2 Réalimentation

La réalimentation (ou rafraîchissement) d'un entrepôt de données peut être effectuée de manière :

- complète : toutes les données sources sont chargées,
- incrémentale : seules sont chargées les nouvelles données sources par rapport au précédent chargement.

Dans le cas d'une réalimentation incrémentale, l'ETL doit être capable d'identifier les « nouvelles » données. Il existe, pour cela, plusieurs possibilités, qui sont présentées dans [BT98] :

- si les données sources sont datées, le système peut se reposer sur ces informations,

- le système peut effectuer des comparaisons de données entre les sources et la cible,
- des triggers peuvent être mis en place au niveau des sources de données. Ceux-ci se déclenchent à la mise à jour des données et stockent ainsi les changement effectués dans un espace réservé,
- les logs de transactions peuvent être analysés afin de tracer les changements,
- …

2.3.3.4.3 Gestion des performances

La gestion des performances est cruciale dans le domaine des entrepôts de données. Les techniques utilisées par les entrepôts de données, comme les index ou les vues persistantes, sont prises en compte par les ETL qui effectuent la mise à jour de ces index et vues persistantes lors des phases d'alimentation.

2.3.3.4.4 Gestion des dysfonctionnements

L'exécution d'un processus ETL peut être source de dysfonctionnements . L'ETL fournit des outils permettant de les gérer :

- reprise de processus lorsque celui-ci a été stoppé avant d'être terminé,
- identification et traitement les données rejetées lors de la phase de nettoyage ou d'alimentation de la cible.

2.3.4 Marché actuel

De nombreux ETL sont actuellement disponibles sur le marché. Le « Magic Quadrant for Data Integration Tools 2009 » du cabinet d'étude Gartner, présenté sur la figure 11, donne une idée des principaux acteurs du domaine.

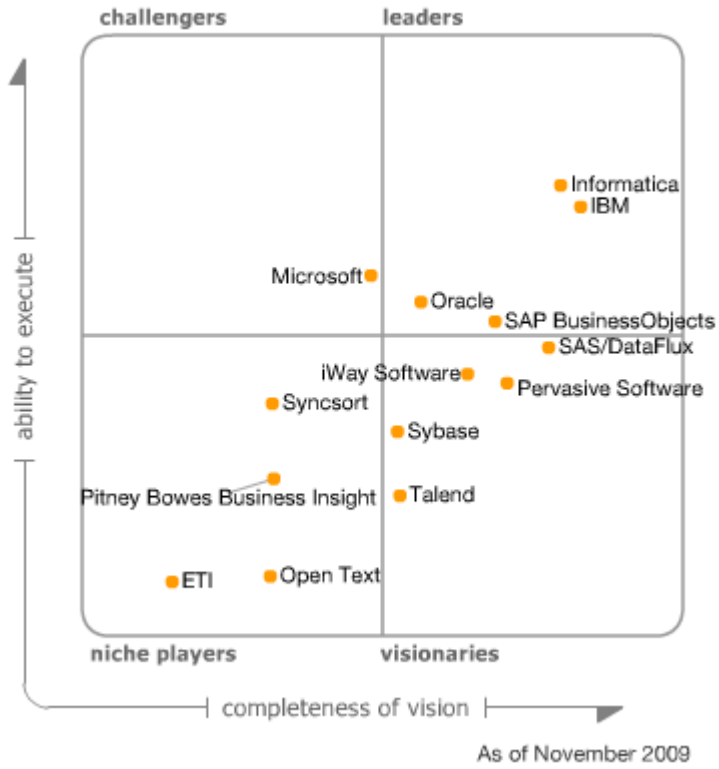


Figure 11 : Magic Quadrant for Data Integration Tools 2009

Comme évoqué en début de section, la complexité des systèmes d'information des entreprises ne cesse de croitre et les sources de données de se diversifier. Un ETL doit donc être capable de s'adapter à des sources de données hétérogènes et variées. La compatibilité est un critère important dans le choix d'un ETL.

Les besoins en termes de décision ont eux aussi tendance à être de plus en plus variés. L'étape de transformation suit cette évolution et doit pouvoir proposer un éventail de transformations de plus en plus riche.

Les solutions comparées sur la figure 11 couvrent des besoins beaucoup plus larges que ceux d'un « simple » ETL en proposant :

- des composants décisionnels avancés remplissant des fonctions de :
    - reporting : il permet aux gestionnaires de réaliser rapidement des rapports selon des formats prédéterminés répondant à leurs besoins,
    - présentation des données : cette fonction régit les conditions d'accès de l'utilisateur aux informations. Elle assure le contrôle d'accès, la prise en charge des requêtes des utilisateurs, la visualisation des résultats, etc.
    - diffusion des données : la diffusion met les données à la disposition des utilisateurs, selon des schémas correspondant au profil ou au métier de chacun.
- un environnement d'administration complet pouvant proposer divers outils :
    - des planificateurs de tâches pour planifier l'exécution des processus ETL,
    - des gestionnaires de reprise qui permettent de relancer les processus ETL en cas de problème,
    - des moniteurs d'exécution permettant de suivre le bon fonctionnement de l'exécution des processus ETL en temps réel,
    - ….

La mise en place de telles solutions nécessite des compétences particulières, une maintenance lourde et des investissements conséquents.

3 L'OUTIL XEUTL

XeuTL est un outil pour l'intégration de données basé sur l'approche matérialisée. C'est un outil de type ETL qui permet de concevoir et d'exécuter des processus ETL afin d'alimenter une base de données cible avec des données issues de différentes bases de données sources. Cette base de données cible peut être une base de données standard ou prendre la forme d'un entrepôt de données. XeuTL utilise des fichiers XML comme support d'échange.

Ce chapitre présente le fonctionnement général de XeuTL.

3.1 Objectifs

Un processus ETL est une suite d'opérations nécessaires à l'alimentation d'une base cible avec des données réparties dans différentes bases sources hétérogènes : Extraction, Transformation, Chargement (voir § 2.3). La mise en place de processus ETL dans l'entreprise est une tâche lourde et complexe. XeuTL a pour objectif de simplifier cette tâche. XeuTL ne se veut pas un concurrent des solutions lourdes du marché (cf. § 2.3.4) mais se présente comme une alternative **légère** et plus **accessible** en terme d'apprentissage, de mise en place et d'utilisation.
XeuTL permet de réaliser des migrations de données et des alimentations ou réalimentations. XeuTL permet notamment de « combiner » les données de différentes sources hétérogènes vers une base cible et ainsi réaliser des intégrations verticales (exemple : intégration de la base patient d'un centre hospitalier avec celle d'un autre centre) ou des fusions de données (exemple : intégration d'une base « patient » avec une base « analyses biologiques » vers une base « suivi patient »).
XeuTL est centré sur les processus ETL et ne fournit pas de composants décisionnels.
On ne peut parler de processus ETL sans parler d'hétérogénéité (cf. § 2.12.2). XeuTL traite cette problématique de l'hétérogénéité de différentes manières, comme nous le verrons dans les chapitres suivants.

3.2 Fonctionnement général

XeuTL permet aux utilisateurs :

- de définir des processus ETL de manière simple,
- d'exécuter les processus ETL.

3.2.1 Définition d'un processus ETL

La figure 12 présente les différentes étapes de la définition d'un processus ETL dans XeuTL.

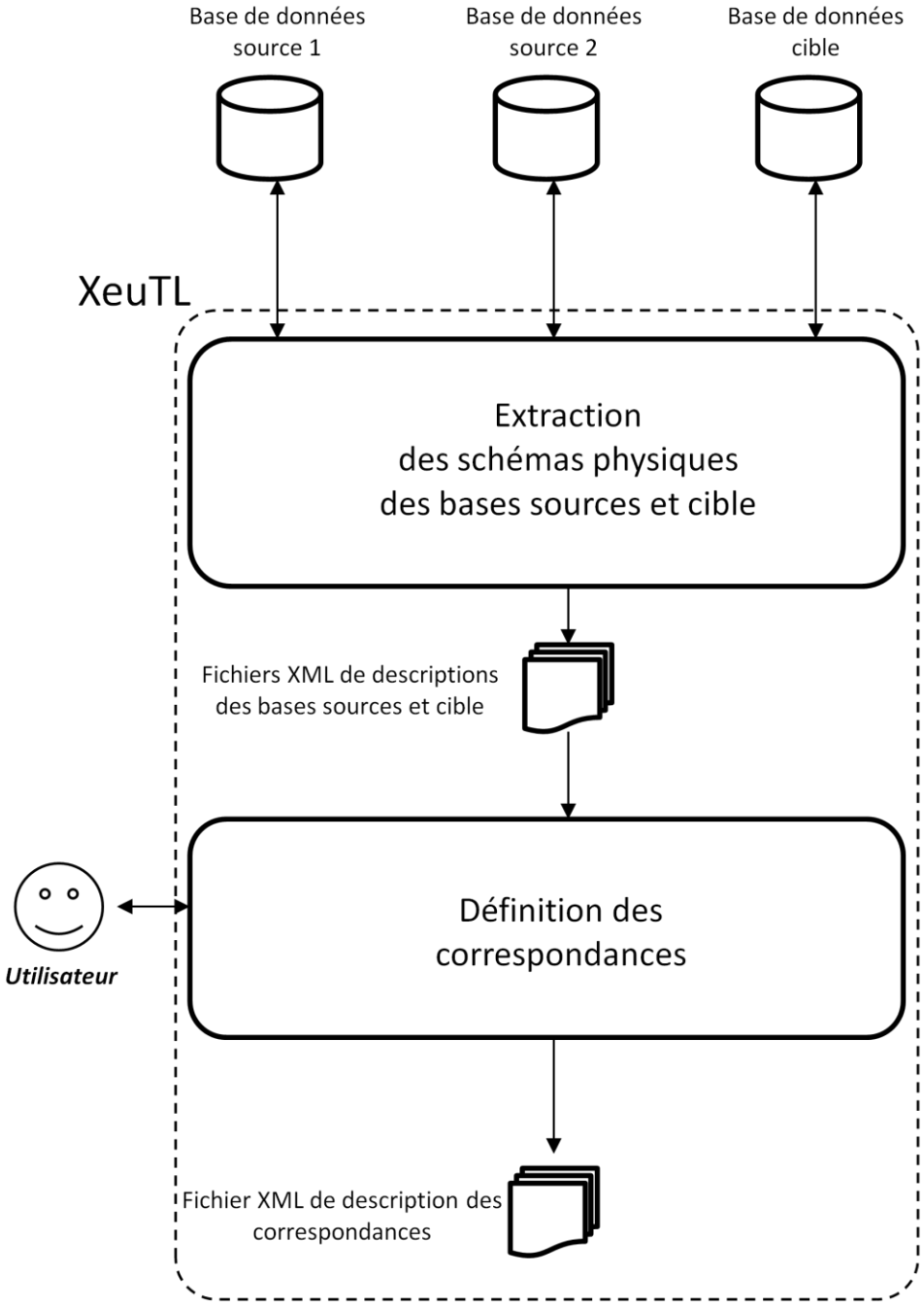


Figure 12 : définition d'un processus ETL

Dans une première étape, XeuTL extrait le schéma physique des bases sources et cibles sous la forme de fichiers XML. L'utilisateur définit ensuite les correspondances entre les bases sources et cible. Les correspondances sont stockées dans un fichier XML.

3.2.2 Exécution d'un processus ETL

Le schéma de la figure 13 résume l'exécution d'un processus ETL XeuTL.

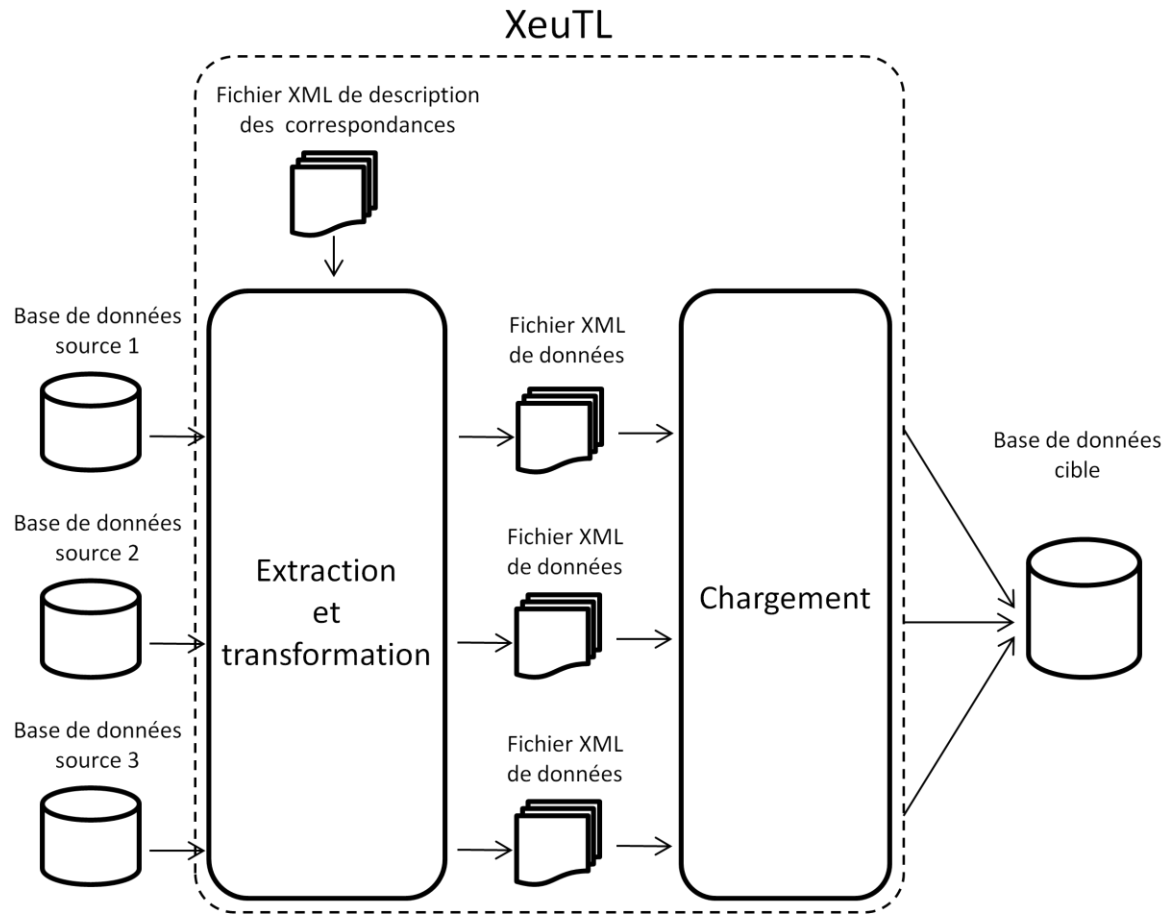


Figure 13 : exécution d'un processus ETL par XeuTL

L'exécution d'un processus ETL de XeuTL se décompose en deux phases:

- phase d'extraction et de transformation : cette phase permet d'extraire les données d'une ou de plusieurs bases de données relationnelles, d'y appliquer un ensemble de transformations selon les correspondances définies dans l'étape de définition du processus ETL et de stocker le résultat obtenu dans des fichiers au format XML.
- phase de chargement : cette phase permet de charger dans une base de données relationnelle cible les données contenues dans les fichiers XML issus des phases d'extraction et de transformation précédemment réalisées.

Ces deux phases se déroulent en séquence.

3.3 Principales fonctionnalités

Cette section présente les principales fonctionnalités de XeuTL.

3.3.1 Connexion et extraction du schéma physique des bases de données

Cette fonctionnalité permet de se connecter à une base et d'extraire ses métadonnées (tables du schéma physique, clés, contraintes, …). Ces métadonnées sont stockées dans un fichier XML (un fichier par base). XeuTL est compatible avec différents systèmes de gestion de bases de données (ceux accessibles au travers de l'API[19] JDBC[20] : cf. § 5.2.1).

3.3.2 Etablissement des correspondances entre les schémas sources et cible

XeuTL fournit des outils permettant de mettre en place des correspondances variées entre les schémas sources et cible d'un processus ETL. Ces correspondances sont utilisées par XeuTL pour effectuer les transformations qui sont au cœur du processus ETL.

Le chapitre 4 détaille les correspondances et transformations possibles.

3.3.3 Extraction des données des bases sources au format XML

XeuTL possède un mécanisme d'extraction de données adapté à différents SGBD. Lors de l'extraction, les données sont transformées en utilisant les correspondances précédemment définies. Le résultat de l'extraction de données des bases sources est un fichier XML contenant les données à transférer vers la base cible.

Beaucoup de SGBD (notamment tous les SGBD relationnels) utilisent le langage SQL comme langage de manipulation de données. Cependant, les implémentations du langage SQL par différents constructeurs ne sont pas toutes identiques. Les différences sont encore plus marquées lorsque les gestionnaires des sources de données sont de différents types (fichiers, SGBD réseaux, hiérarchiques, relationnels, …).

Dans XeuTL, le langage XML est utilisé comme format pivot. Ceci permet de transférer des données entre des SGBD hétérogènes.

3.3.4 Chargement des données dans la base cible

Cette fonctionnalité permet de charger dans la base cible les données stockées dans le fichier XML résultant de l'extraction et de la transformation des données des bases sources.

[19] Application Programming Interface
[20] Java DataBase Connectivity

3.4 Contraintes

La réalisation de XeuTL a été guidée par un certain nombre de contraintes, qui sont détaillées dans cette section.

3.4.1 Indépendance vis-à-vis des systèmes de gestion de bases de données utilisés

XeuTL peut se connecter de manière uniforme à différents systèmes de gestion de bases de données. La couche XML permet d'effectuer des transferts de données entre systèmes de gestion de bases de données hétérogènes en servant de langage commun à ces différents systèmes (format pivot).

3.4.2 Volume des données à traiter

Le rôle d'un outil ETL est de transférer des données depuis une ou plusieurs sources de données (bases de données relationnelles, fichiers plats, etc.), vers une base de données cible. Ce type d'outil doit nécessairement être capable de gérer de grandes quantités de données.

La gestion de gros volumes de données implique des temps de traitements qui peuvent être très importants. Dans le monde des ETL la question de la performance n'est pas, dans une certaine limite, la priorité. En règle générale, les processus ETL sont exécutés durant des périodes où le système n'est pas surchargé et où les ressources peuvent être entièrement dédiées à ces opérations.

La priorité a donc toujours été donnée aux capacités de traitements de gros volumes de données plutôt qu'aux performances en termes de temps de traitement.

3.4.3 Automatisation des processus ETL

Les processus ETL sont souvent programmés pour être exécutés en mode batch (cf. § 2.3.3.3). XeuTL offre la possibilité de stocker l'ensemble des informations nécessaires à l'exécution d'un processus ETL, sous forme de fichier XML. Ceci permet de programmer le lancement de ces processus de manière à ce qu'ils soient exécutés automatiquement sans qu'il soit nécessaire de se connecter à l'application.

3.4.4 Transport de données

L'acheminement des données provenant des bases sources vers la base cible n'est pas directement pris en charge par XeuTL. XeuTL stocke les données à transférer dans des fichiers XML. Ceci laisse une grande liberté aux utilisateurs du système dans le choix des moyens de transport à mettre en œuvre pour leurs processus ETL. Les données peuvent ainsi transiter par un réseau (par sérialisation, par courrier électronique, …), par un support de stockage (disque, clé usb, …), être encryptées ou non.

3.4.5 Facilité de prise en main

XeuTL permet une prise en main relativement aisée. XeuTL fournit à l'utilisateur une interface permettant de réaliser des processus ETL complets sans avoir à maîtriser les langages SQL ou XML (bien que ceci soit utile pour certains types de processus complexes).

4 LES CORRESPONDANCES

Les correspondances sont le cœur de XeuTL. Les correspondances définies par l'utilisateur indiquent au système la provenance des données qui vont alimenter la base cible lors de l'exécution du processus ETL, ainsi que le format qu'elles doivent respecter. A l'exécution du processus ETL, XeuTL déduit de ces correspondances les transformations que le système doit appliquer sur les données sources afin de les intégrer à la base cible.
Chaque correspondance peut être vue comme un lien entre une ou plusieurs colonnes des bases sources et une colonne de la base cible d'un processus ETL.
Cette section présente l'ensemble des correspondances possibles dans XeuTL. La description de ces correspondances est présentée dans le paragraphe 5.4.3.4.1.
Nous avons distingué deux types de correspondances :

- les correspondances simples, définies entre une unique colonne d'une base source et une unique colonne de la base cible,
- les correspondances complexes, qui peuvent être définies entre plusieurs colonnes des bases sources et une unique colonne de la base cible.

4.1 Les correspondances « simples »

Par souci de simplicité nous avons décidé dans cette section de ne parler que de requêtes simples : requêtes de type calcul, requêtes de type transtypage, etc. Il est intéressant de noter qu'une même requête peut présenter plusieurs caractéristiques ; par exemple, XeuTL est capable de générer des requêtes de type calcul et d'effectuer un transtypage sur le résultat de ce calcul.

4.1.1 Correspondances atomiques

Une correspondance atomique est la correspondance la plus simple qui puisse être définie dans le système. Elle associe une colonne d'une table de la base source à une colonne d'une table de la base cible. Lors de l'exécution du processus ETL les données de la colonne source seront copiées dans la colonne cible.

La figure 14 présente des exemples de correspondances atomiques entre une base source et une base cible.

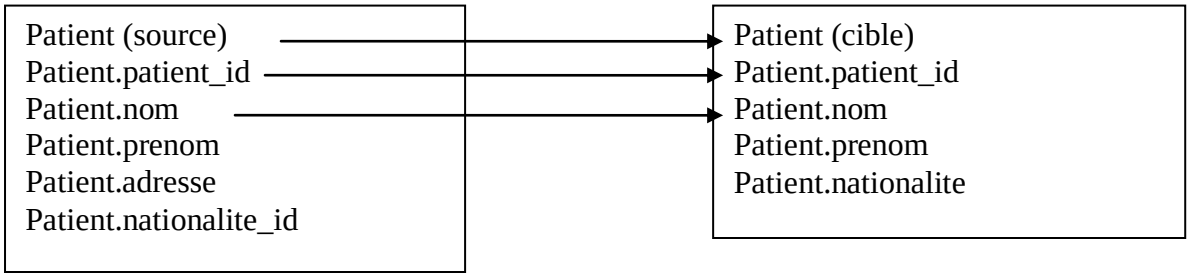


Figure 14 : exemples de correspondances atomiques

Les flèches indiquent les correspondances atomiques.

Considérons une table *Patient* de la base source avec la structure suivante :

Patient				
patient_id	nom	prenom	adresse	nationalite_id
1	Dupond	Alexandre	Rue Montorge	1
2	Durand	Martine	7 Avenue du bois, 39098	2
3	Martin	Celia	Lyon	2

Figure 15 : table *Patient* de la base source

A l'exécution du processus ETL, le traitement des correspondances atomiques présentées dans la figure 14 donne les résultats de la figure 16.

Patient			
patient_id	nom	prenom	nationalite_id
1	Dupond	Alexandre	
2	Durand	Martine	
3	Martin	Celia	

Figure 16 : table *Patient* de la base cible une fois le processus ETL effectué

4.1.2 Correspondances de type calcul

Une correspondance de type calcul est une correspondance qui va appliquer un calcul sur une valeur d'une colonne source avant de copier le résultat obtenu dans la colonne cible lors de l'exécution du processus ETL.
Les calculs disponibles sont des calculs simples relativement standard (addition, soustraction, multiplication, division, …).

La figure 17 présente un exemple d'une correspondance de type calcul.

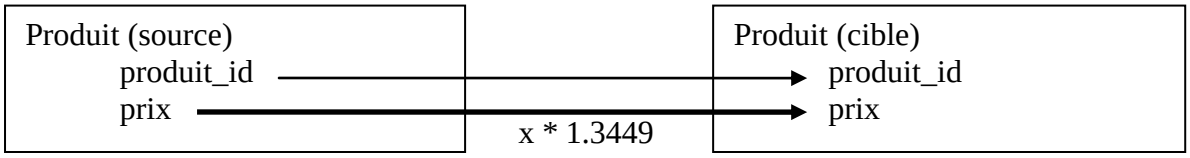


Figure 17: exemple de correspondance de type calcul

La flèche en gras indique une correspondance de type calcul. Le calcul défini ici est x * 1.3449 ou x est la valeur source. Le résultat de l'opération est la valeur cible.

Considérons la table *Produit* de la base source présentant les prix de trois produits définis en euros (Figure 18) :

Produit	
produit_id	prix
1	10
2	12.5
3	2.30

Figure 18 : table *Produit* de la base source

Supposons une base cible *Produit* présentant les prix en dollars US. Les transformations déduites des correspondances présentées dans figure 17 ont pour résultat ceux présentés sur la figure 19.

Produit	
produit _id	prix
1	13.449
2	16.81125
3	28.75

Figure 19: table *Produit* de la base cible après exécution du processus ETL

4.1.3 Correspondances de type valeur fixe

Une correspondance de type « valeur fixe » est une correspondance qui va affecter pour chaque enregistrement une même valeur fixe à la colonne cible. La figure 20 donne un exemple de correspondance de type valeur fixe.

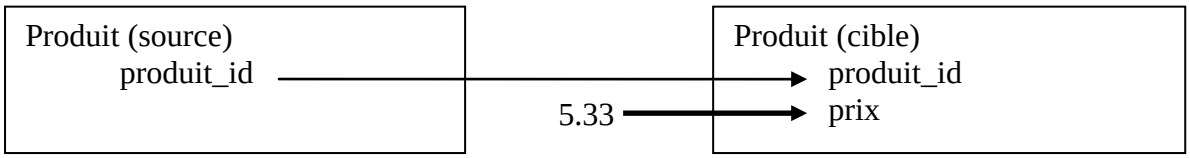


Figure 20 : exemple de correspondance de type valeur fixe

Considérons la table source présenté sur la figure 21.

Produit
produit_id
1
2
3

Figure 21 : table *Produit* de la base source

Le résultat de la transformation déduite de la correspondance définie ici est présenté sur la figure 22.

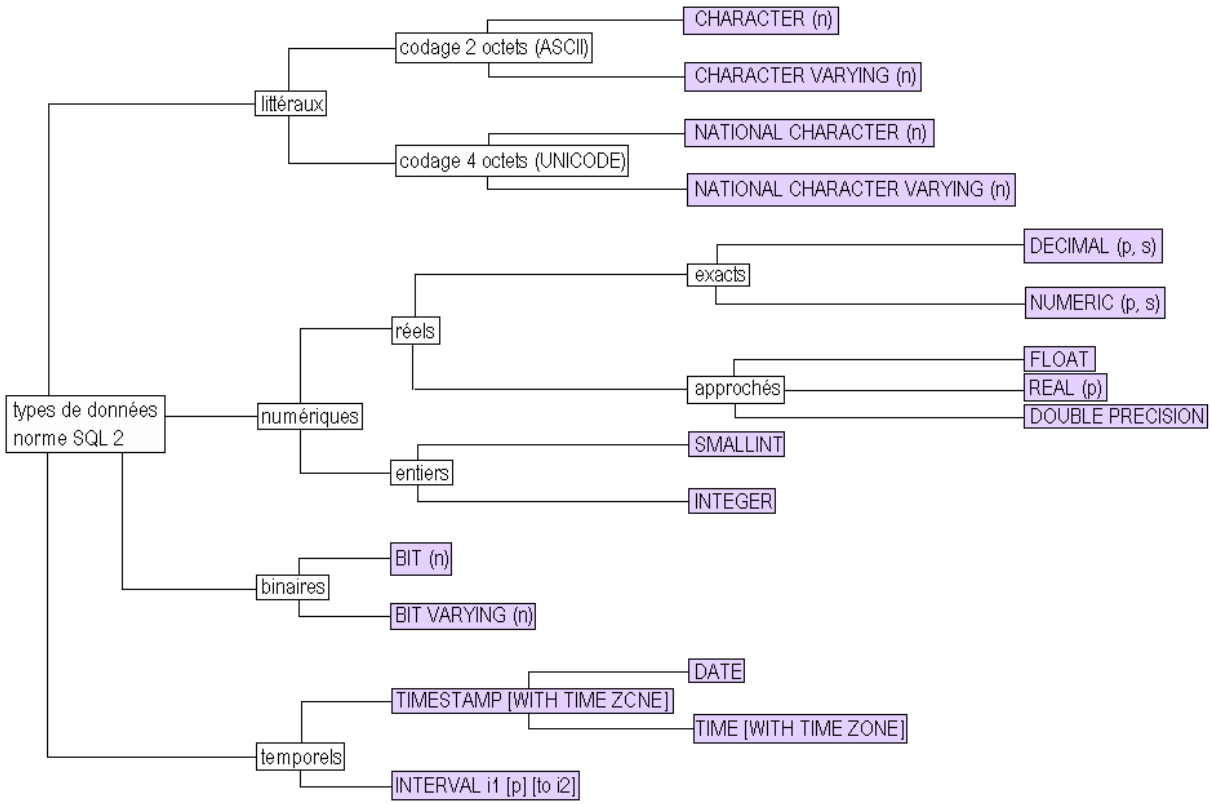
Produit	
produit_id	prix
1	5.33
2	5.33
3	5.33

Figure 22 : table *Produit* de la base cible après exécution du processus ETL

4.1.4 Correspondances de type transtypage

Une correspondance de type transtypage est une correspondance qui va effectuer une modification du type des valeurs copiées de la colonne source vers la colonne cible lors de l'exécution du processus ETL. Ce type de correspondance est particulièrement utile lorsque le système de gestion de base de données de la base source est différent de celui de la base cible. Par exemple, le type *VARCHAR* en Mysql a une capacité de 255 caractères alors qu'en SQL Server, le type *VARCHAR* a une capacité de 4000 caractères. Le type de champ Mysql correspondant à un *VARCHAR* SQL Server serait plutôt un champ de type *TEXT*. XeuTL permet à l'utilisateur d'utiliser le transtypage et ainsi de résoudre ce type d'hétérogénéité.

Afin de limiter les problèmes de compatibilité de types de données entre bases lors de l'exécution des processus ETL, il convient d'utiliser autant que possible des types de données issus de la norme SQL92. La figure 23 présente les différents types de données de la norme SQL92 : les types sont indiqués en gris, les familles et sous-familles en blanc.




```
n : longueur associée au type,
p : précision (nombre de chiffres significatifs),
s : décimales
i1, i2 : précisions de durée parmi YEAR, MONTH,
            DAY, HOUR, MINUTE, SECOND.
```

Figure 23 : les types SQL92

4.1.5 Correspondances de type clé de substitution

Une correspondance de type clé de substitution est une correspondance qui permet de générer une clé automatique pour une colonne de la base cible.

La figure 24 donne un exemple de correspondance de type clé de substitution.

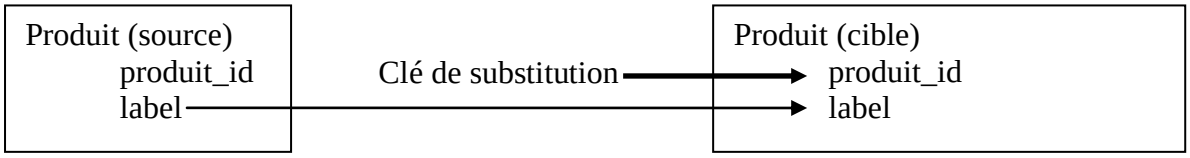


Figure 24 : exemple de correspondance de type clé de substitution

Considérons la table source présenté sur la figure 25.

Produit	
produit_id	label
1	Stylo
2	Papier
3	Pile

Figure 25 : table *Produit* de la base source

Supposons la table *Produit* de la base cible définie ci-dessous avant l'exécution du processus ETL :

Produit	
produit_id	label
20	Livre
21	Chargeur
22	DVD
23	CD
24	Prise

Figure 26 : table *Produit* de la base cible avant exécution du processus ETL

Le résultat de la transformation déduite de la correspondance définie ici est présenté sur la figure 27.

Produit	
produit_id	label
20	Livre
21	Chargeur
22	DVD
23	CD
24	Prise
25	Stylo
26	Papier
27	Pile

Figure 27 : table *Produit* de la base cible après exécution du processus ETL

Le résultat de l'application d'une transformation de type clé de substitution est identique à celui d'une requête d'insertion effectuée sur une table dont la clé primaire est auto-incrémentée.
Dans notre exemple, la correspondance définie aurait le même résultat que l'exécution des requêtes SQL suivantes si la colonne *produit_id* de la table *produit* de la cible était auto-incrémentée :

```
Insert into Produit(Label) values ('Stylo');
Insert into Produit(Label) values ('Papier');
Insert into Produit(Label) values ('Pile');
```

La correspondance de type clé de substitution est utile dans plusieurs situations :

- la table source ne possède pas de clé et la table cible en possède une qui n'est pas incrémentée,
- le projet d'intégration nécessite des réalimentations et l'utilisateur souhaite historiser les données intégrées plutôt que de les écraser lors de chaque réalimentation. En utilisant pour la table cible les mêmes valeurs de clés que celles de la table source, l'unicité des clés de la table ne serait pas conservée. La correspondance de type clé de substitution permet de générer une nouvelle clé en garantissant que celle-ci est unique dans la table.

4.1.6 Correspondances de type troncature

Une correspondance de type troncature est une correspondance qui va tronquer la valeur d'une colonne source avant de copier le résultat obtenu dans la colonne cible lors de l'exécution du processus ETL. La colonne source doit être de type *literal* (cf. § 4.1.4).
La troncature dans XeuTL peut s'effectuer à gauche ou à droite.

La figure ci-dessous présente un exemple d'une correspondance de type troncature à gauche.

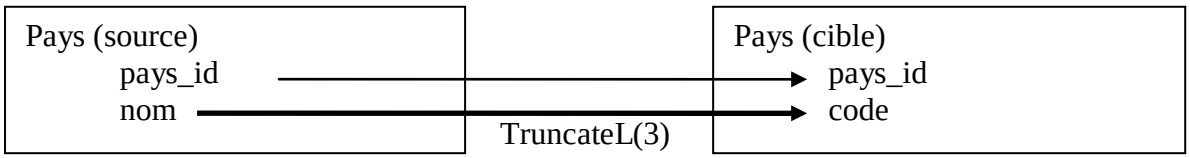


Figure 28: exemple de correspondance de type troncature

La flèche en gras indique une correspondance de type troncature. La troncature définie ici copie dans le champ cible une chaîne composée des trois premiers caractères de la valeur source.

Considérons la table *Pays* de la base source définie comme suit :

Pays	
pays_id	nom
1	FRANCE
2	ANGLETERRE
3	IRLANDE

Figure 29 : table *Pays* de la base source

Supposons une base *Pays* cible présentant les pays sous forme de code de trois caractères. Les transformations déduites des correspondances présentées dans la figure 28 auraient pour résultats ceux présentés sur la figure 30.

Pays	
pays_id	code
1	FRA
2	ANG
3	IRL

Figure 30: table *Pays* de la base cible après exécution du processus ETL

Ce type de correspondance peut être utilisé à la manière d'une correspondance de type transtypage. La valeur d'un champ source d'une capacité supérieure à celle du champ cible devra être tronquée afin de pouvoir être chargée dans le champ cible.

4.2 Les correspondances « complexes »

4.2.1 Correspondances de type référence

4.2.1.1 Depuis la base source

Soient deux tables sources T1 et T2 et C1 une des colonnes de T1 définie comme clé étrangère référençant C2, une des colonnes de T2. Une correspondance de type référence associe une colonne de T2 (autre que la colonne référencée par la clé étrangère de T1) à une colonne de la table cible.
A l'exécution du processus ETL, la valeur copiée dans la base cible sera celle de l'enregistrement référencé par la clé étrangère. En d'autres termes, une table source fait référence à une autre table source. La valeur récupérée pour alimenter la cible sera alors la valeur de la colonne de la seconde table de l'enregistrement référencé par la clé étrangère de la première table.

La figure ci-dessous présente un tel exemple de correspondance.

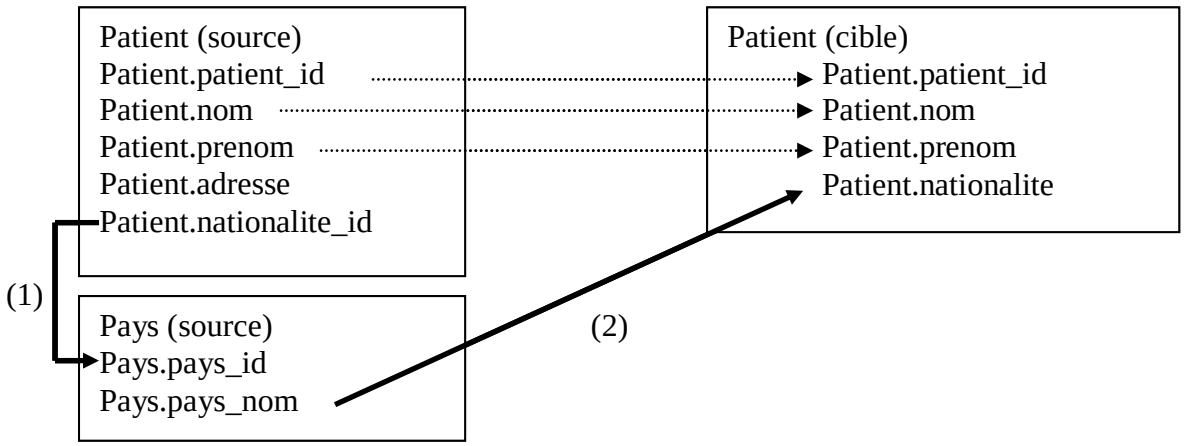


Figure 31 : correspondance de type référence (en gras)

La flèche (1) indique la clé étrangère définie entre ces deux tables. La flèches (2) indique la correspondance de type référence ; les autres flèches indiquent des correspondances de type atomiques.

Considérons la table *Patient* de la base source (Figure 32) :

Patient				
patient_id	nom	prenom	adresse	nationalite_id
1	Dupond	Alexandre	Rue Montorge	1
2	Durand	Martine	7 Avenue du bois, 39098	2
3	Martin	Celia	Lyon	2

Figure 32: table *Patient* de la base source

La table *Pays* de la base source est présenté ci-dessous.

Pays	
pays_id	pays_nom
1	France
2	Suisse
3	Royaume-Uni

Figure 33 : table *Pays* de la base source

A l'exécution du processus ETL, les transformations déduites des correspondances atomiques ainsi que la transformation déduite de la correspondance de type référence présentées dans la figure 31 ont pour résultat ceux de la figure ci-dessous.

Patient			
patient_id	nom	prenom	nationalite
1	Dupond	Alexandre	France
2	Durand	Martine	Suisse
3	Martin	Celia	Suisse

Figure 34: table *Patient* de la base cible après exécution du processus ETL

4.2.1.2 Depuis la base cible

Une correspondance de type référence depuis la cible est une correspondance définie entre une colonne d'une table source et un colonne d'une table cible, cette dernière possédant une clé étrangère référençant une autre table cible. A l'exécution du processus ETL, la colonne cible de la correspondance prendra comme valeur celle de la colonne source. Le système génèrera de manière automatique une clé qui alimentera les colonnes référencées et référençant la clé étrangère définie entre les deux tables cibles.

La figure ci-dessous présente un exemple de correspondance de type référence depuis la cible.

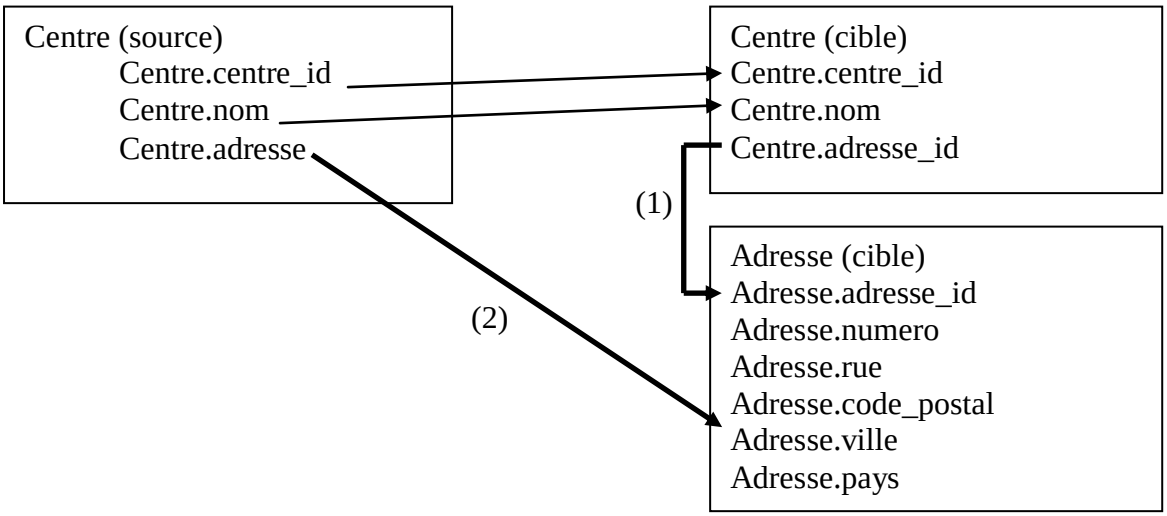


Figure 35 : correspondance de type référence (en gras)

La flèche (1) indique une clé étrangère. La flèche (2) indique la correspondance de type référence, les autres indiquent les correspondances atomiques.

Considérons la table *Centre* de la base source (Figure 36) :

Centre		
centre_id	nom	adresse
1	CHU Grenoble	Grenoble
2	HUG Genève	Genève
3	Hôpital sud	Echirolles

Figure 36 : table *Centre* de la base source

A l'exécution du processus ETL, le traitement des correspondances présentées dans la figure 35 aurait pour résultat ceux présentés dans les deux figures ci-dessous.

Centre		
centre_id	nom	adresse_id
1	CHU Grenoble	1
2	HUG Genève	3
3	Hôpital sud	2

Figure 37 : table *Centre* de la base cible après l'opération de migration

Adresse					
adresse_id	numero	rue	code_postal	ville	pays
1				Grenoble	
2				Echirolles	
3				Genève	

Figure 38 : table *Adresse* de la base cible après l'opération de migration

4.2.2 Correspondances de type concaténation

Une correspondance de type concaténation permet de concaténer les valeurs de plusieurs champs d'un enregistrement de la base source et de copier la chaîne ainsi obtenue dans un champ de la base cible.

La figure ci-dessous présente un exemple de correspondance de type concaténation.

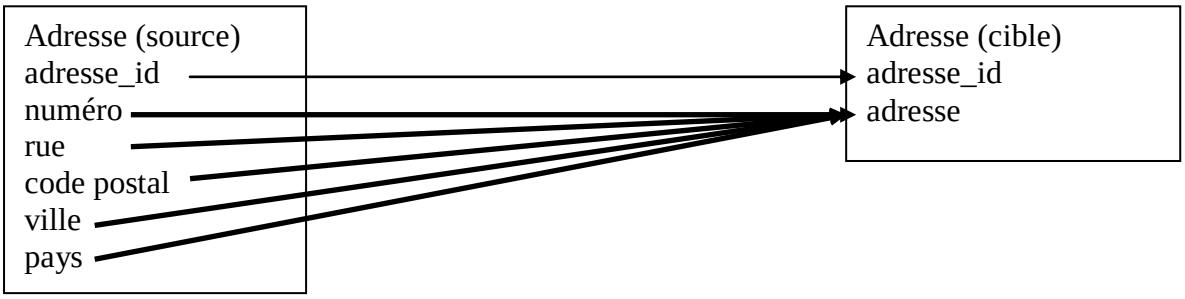


Figure 39 : correspondance de type concaténation

Les flèches en gras indiquent les correspondances de type concaténation.

Considérons la table *Adresse* de la base source ci-dessous :

Adresse				
adresse_id	numéro	rue	code postal	ville
1	12	Turenne	38000	Grenoble
2	102	Alembert	38000	Grenoble
3	7	République	69000	Lyon

Figure 40 : table *Adresse* de la base source

La base cible *Adresse* résultant de l'exécution du processus ETL défini ci-dessus est représentée figure ci-dessous.

Adresse	
adresse_id	adresse
1	12 Turenne 38000 Grenoble
2	102 Alembert 38000 Grenoble
3	7 République 69000 Lyon

Figure 41 : table *Adresse* de la base cible

4.2.3 Correspondances de type requête imbriquée

Les correspondances de type requête imbriquée permettent de déduire la valeur d'un champ cible d'une requête effectuée sur la base source.

La figure 42 présente un exemple de cette correspondance.

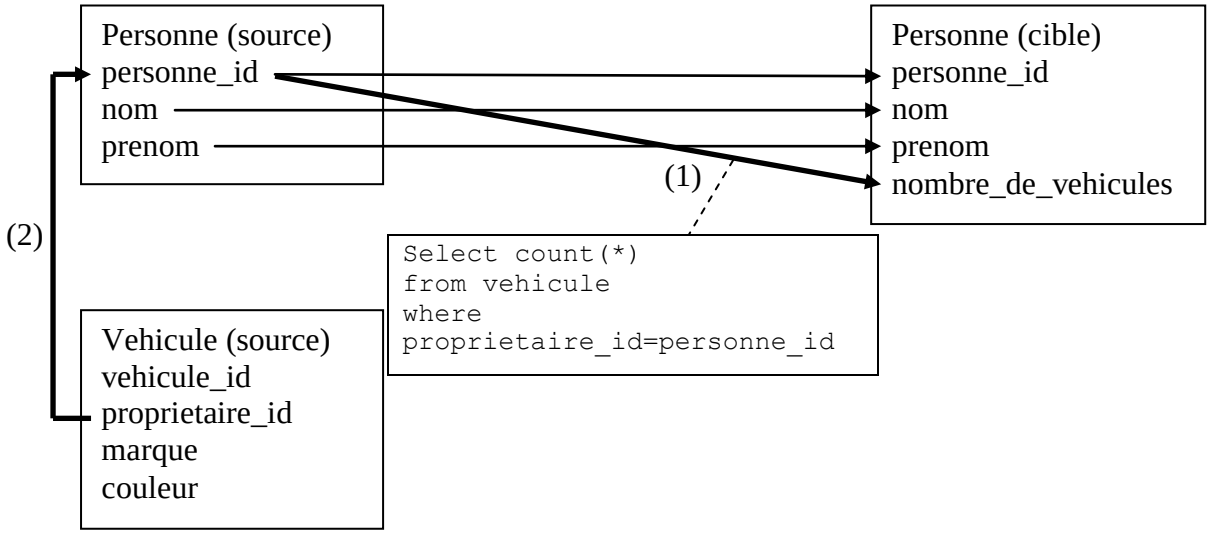


Figure 42 : correspondance de type requête imbriquée

La flèche en gras entre la base source et la base cible indique une correspondance de type requête imbriquée ; celle entre les deux bases sources indique une clé étrangère, et les autres indiquent les correspondances atomiques. `proprietaire_id` est une clé étrangère référençant `personne_id` et implémentant la relation *propriétaire*.

Dans ce contexte, la requête d'agrégation

```
(Select count(*) from vehicule where proprietaire_id = personne_id)
```

permet de compter le nombre de véhicules dont une personne est propriétaire et de stocker cette information dans la base cible.

Considérons les tables *Personne* (Figure 43) et *Vehicule* (Figure 44) de la base source :

Personne		
personne_id	nom	prenom
1	Durand	Pierre
2	Dupond	Alexandre
3	Rougemont	Yves
4	Martin	Simon

Figure 43 : table *Personne* de la base source

Considérons la table *Vehicule* de la base source définie comme suit :

Vehicule			
vehicule_id	proprietaire_id	marque	couleur
1	2	Peugeot	rouge
2	2	Citroën	grise
3	2	Audi	noire
4	4	Chrysler	blanche

Figure 44 : table *Vehicule* de la base source

La base cible *Personne* résultant de l'exécution du processus ETL défini ci-dessus est représentée ci-dessous.

Personne			
personne_id	nom	prenom	nombre_de_vehicule
1	Durand	Pierre	0
2	Dupond	Alexandre	3
3	Rougemont	Yves	0
4	Martin	Simon	1

Figure 45 : table *Personne* de la base cible

5 CONCEPTION ET REALISATION

Ce chapitre expose la conception et la réalisation technique de l'outil XeuTL.

5.1 Conception globale

5.1.1 Les cas d'utilisations

Du point de vue de l'utilisateur du système, un processus ETL de XeuTL peut se décomposer en cinq opérations successives :

- extraction d'un schéma physique d'une base de données (source ou cible),
- chargement d'un schéma physique,
- établissement des correspondances entre les schémas sources et cible,
- extraction et transformation des données des bases sources,
- chargement des données dans la base cible.

La figure 46 représente le diagramme de cas d'utilisation de XeuTL.

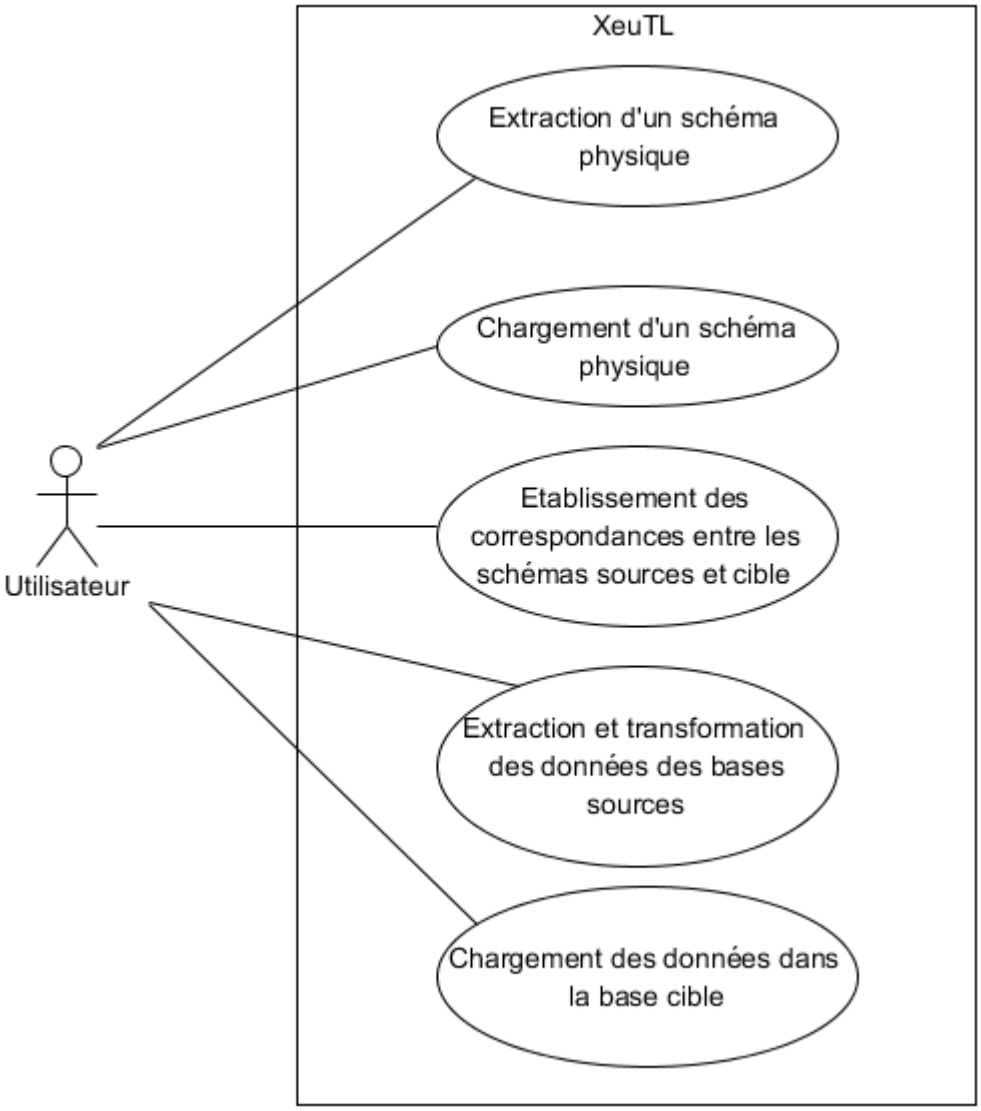


Figure 46 : diagramme de cas d'utilisation

5.1.2 Diagramme de classes

Une première analyse des besoins d'un logiciel ETL nous a conduit à définir les principales classes suivantes :

- XeuTL : noyau du système,
- XDatabase : représentation d'une base de données,
- XTable : représentation d'une table d'une base de données,
- XColonne : représentation d'une colonne d'une table d'une base de données,
- Transformation : représentation d'une correspondance,
- Batch : représentation d'un lot qui est un groupement de correspondances.

Le diagramme de classes de niveau analyse est présenté figure 47.

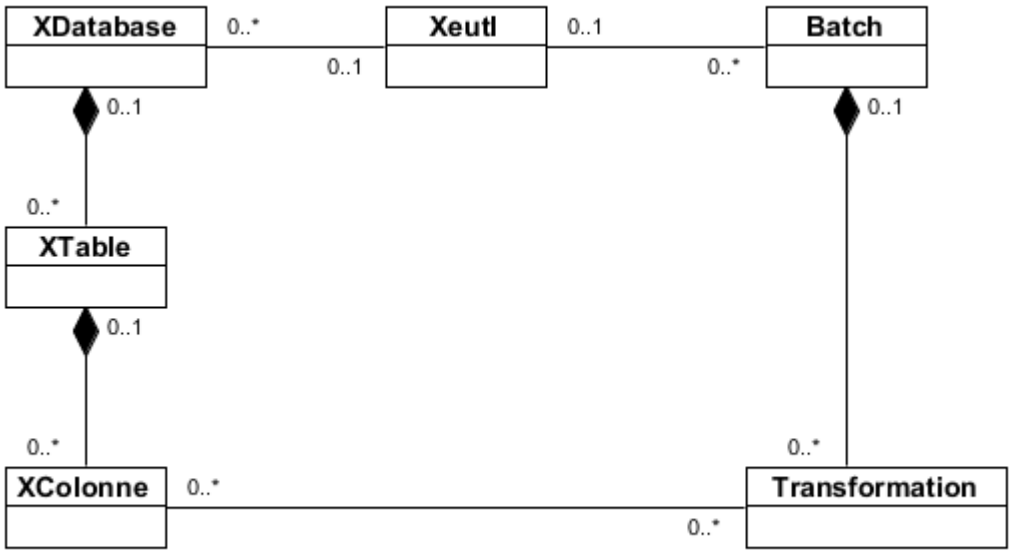


Figure 47 : diagramme de classes de niveau analyse

5.1.3 Les cas d'utilisation en détail

Le format de présentation des cas d'utilisation est inspiré des recommandations d'Alistair Cockburn [NET05].

5.1.3.1 Extraction d'un schéma physique

L'extraction de la structure des sources de données va permettre de représenter les schémas des bases sources et cibles à l'utilisateur afin que celui-ci puisse définir les correspondances qu'il souhaite mettre en place entre les bases sources et la base cible.

Cas d'utilisation : extraction d'un schéma physique.
Acteur : utilisateur.
Parties prenantes et intérêts :

utilisateur : il souhaite consulter le schéma physique des bases concernées par le processus ETL qu'il est en train de définir.

Pré-conditions : les bases sont accessibles (par JDBC).
Scénario nominal :

- l'utilisateur indique la base de données à laquelle il souhaite se connecter,
- le système créé un fichier XML contenant les informations de connexion à la base,
- l'utilisateur indique qu'il souhaite extraire le schéma physique de la base,
- le système établit la connexion à la base en utilisant les information du fichier XML de connexion,
- le système extrait le schéma physique de la base et le stocke dans un fichier XML.

Annexe : la figure 48 présente le diagramme de séquence « extraction d'un schéma physique ».

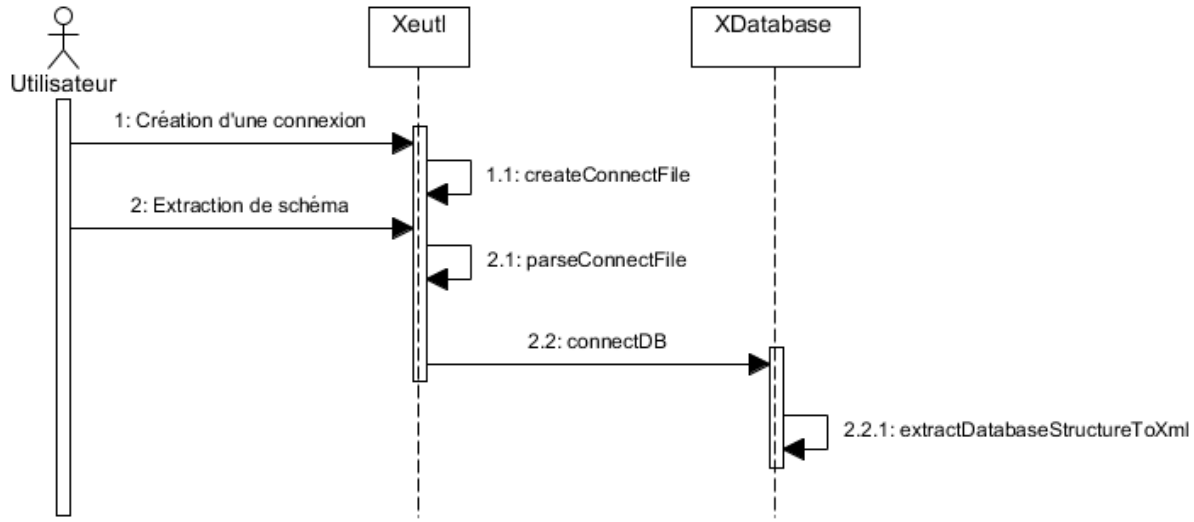


Figure 48 : diagramme de séquence « extraction d'un schéma physique »

5.1.3.2 Chargement d'un schéma physique

Le chargement d'un schéma physique permet à l'utilisateur de charger et consulter un schéma physique précédemment extrait.

Cas d'utilisation : chargement d'un schéma physique.
Acteur : utilisateur.
Parties prenantes et intérêts :
utilisateur : l'utilisateur souhaite consulter un schéma physique précédemment extrait.
Pré-conditions : le fichier XML contenant la description du schéma physique existe.
Scénario nominal :

- l'utilisateur demande à charger un schéma physique défini dans un fichier XML,
- le système charge le fichier,
- le système parcourt le fichier,
- à chaque balise <*database*> rencontrée un objet *XDatabase* est créé,
- à chaque balise <*table*> rencontrée un objet *XTable* est créé et rattaché à l'objet *XDatabase* courant,
- à chaque balise <*colonne*> rencontrée un objet *XColonne* est créé et rattaché à l'objet *XTable* courant.

Annexe : la figure 49 présente le diagramme de séquence « chargement d'un schéma physique ».

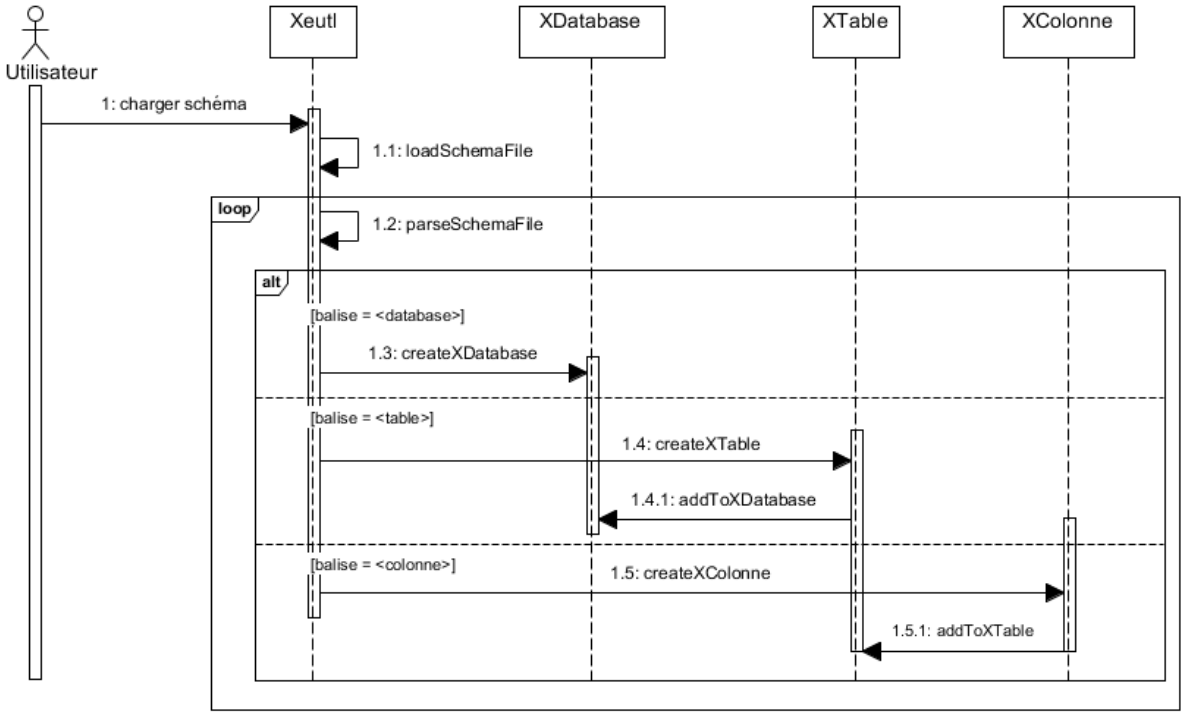


Figure 49 : diagramme de séquence « chargement d'un schéma physique »

5.1.3.3 Etablissement des correspondances entre les schémas sources et cible

En se basant sur les schémas générés précédemment, l'utilisateur définit des correspondances entre les bases sources et la base cible.
La définition de ces correspondances est stockée dans un fichier XML sur lequel s'appuiera le processus ETL.

Cas d'utilisation : définition des correspondances entre les schémas sources et cible.
Acteur : utilisateur.
Parties prenantes et intérêts :

utilisateur : l'utilisateur souhaite définir les correspondances qu'il veut mettre en place entre les bases sources et la base cible dans son processus ETL.

Pré-conditions : tous les fichiers XML de définition de schémas physiques sont disponibles.
Scénario nominal :

- l'utilisateur demande de créer un nouveau lot de transformations,
- le système crée un nouvel objet *batch*,
- l'utilisateur crée une nouvelle correspondance,
- le système crée un nouvel objet *Transformation* et le lie à l'objet *batch* courant,
- l'utilisateur indique une colonne source,
- le système recherche l'objet *XColonne* correspondant,
- le système lie l'objet *XColonne* à l'objet *Transformation* courant,
- l'utilisateur indique une colonne cible,
- le système recherche l'objet *XColonne* correspondant,
- le système lie l'objet *XColonne* à l'objet *Transformation* courant.

Annexe : la figure 50 présente le diagramme de séquence de l'opération « établissement des correspondances entre les schémas sources et cible ».

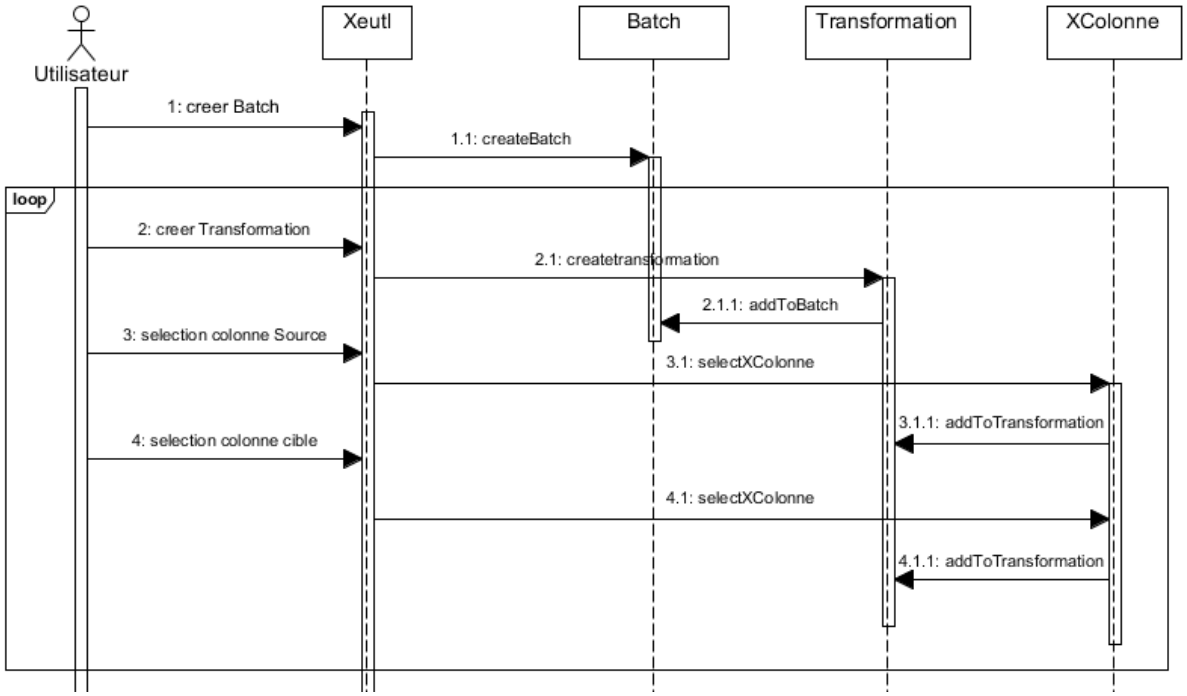


Figure 50 : diagramme de séquence « établissement des correspondances entre les schémas sources et cible »

5.1.3.4 Extraction et transformation des données des bases sources

Cas d'utilisation : extraction et transformation des données des bases sources.
Acteur : utilisateur.
Parties prenantes et intérêts :

utilisateur : l'utilisateur souhaite exécuter la première phase de son processus ETL et extraire les données des bases sources en y appliquant les transformations déduites des correspondances qu'il a définies.

Niveau : objectif utilisateur.
Portée : système.
Pré-conditions : les bases sources sont accessibles et les fichiers XML de définition de correspondances existent.
Scénario nominal :

- l'utilisateur demande à procéder à l'extraction et à la transformation des données,
- le système récupère la liste des objets *batch* concernés par le processus,
- le système récupère la liste des objets *Transformation* liés à l'objet *batch* courant,
- le système récupère la liste des objet *XColonne* sources de la transformation,
- le système récupère la liste des objets *XColonne* cibles de la transformation,
- le système génère l'ordre SQL nécessaire,
- le système exécute l'ordre SQL généré,
- le système stocke le résultat de l'ordre SQL dans le fichier XML de données extraites.

Annexe : la Figure 51 présente le diagramme de séquence de l'opération « extraction et transformation des données des bases sources ».

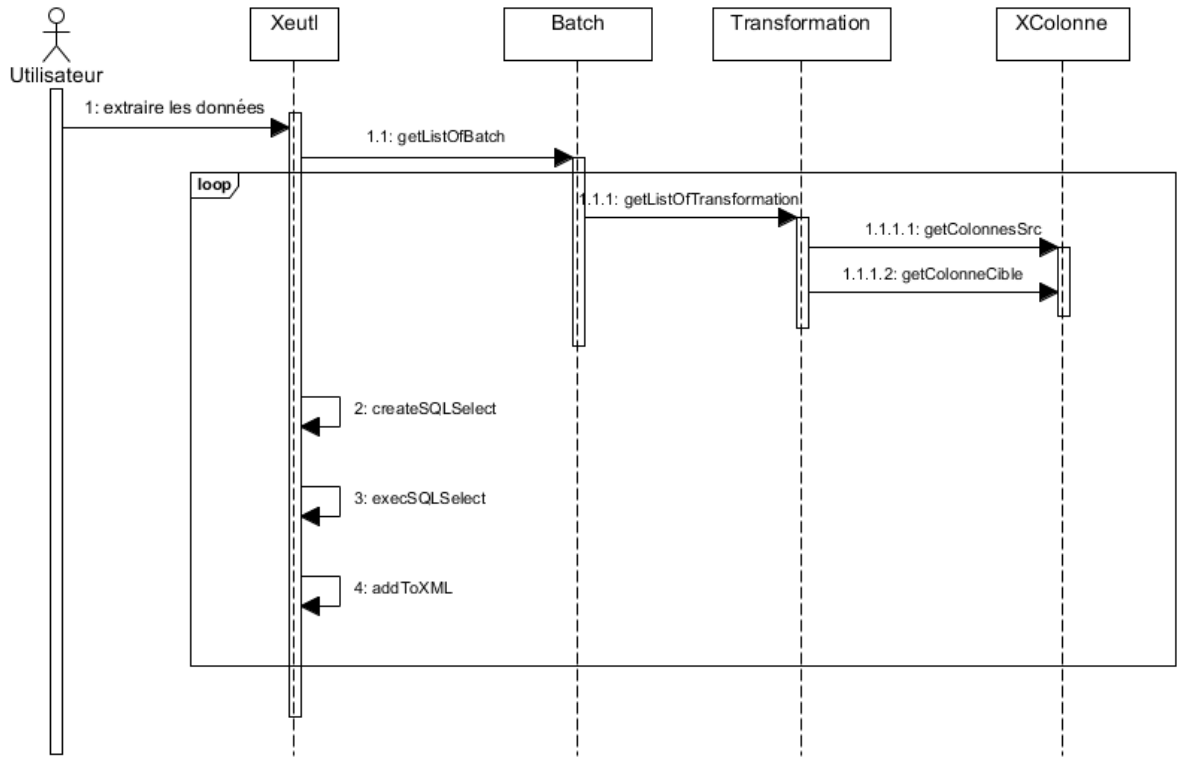


Figure 51 : diagramme de séquence « extraction et transformation des données des bases sources »

5.1.3.5 Chargement des données dans la base cible

A l'issue de la phase d'extraction/transformation les données sont stockées sous la forme d'un fichier XML dont la structure correspond au schéma de la base cible. L'opération de chargement des données permet de charger ces données dans la base cible.

Cas d'utilisation : chargement des données dans la base cible.
Acteur : utilisateur.
Parties prenantes et intérêts :

utilisateur : l'utilisateur souhaite exécuter la seconde phase du processus ETL et charger les données extraites des bases sources dans la base cible.

Pré-conditions : le fichier XML de données existe et la base cible est accessible.
Scénario nominal :

- l'utilisateur demande à charger les données dans la base cible,
- le système se connecte à la base cible,
- le système lit dans le fichier XML de données extraites l'équivalent d'un ordre SQL d'insertion,
- le système créé l'ordre SQL d'insertion correspondant,
- le système exécute l'ordre SQL d'insertion sur la base cible,
- le système se déconnecte de la base cible.

Annexe : la Figure 52 présente le diagramme de séquence de l'opération « chargement des données dans la base cible ».

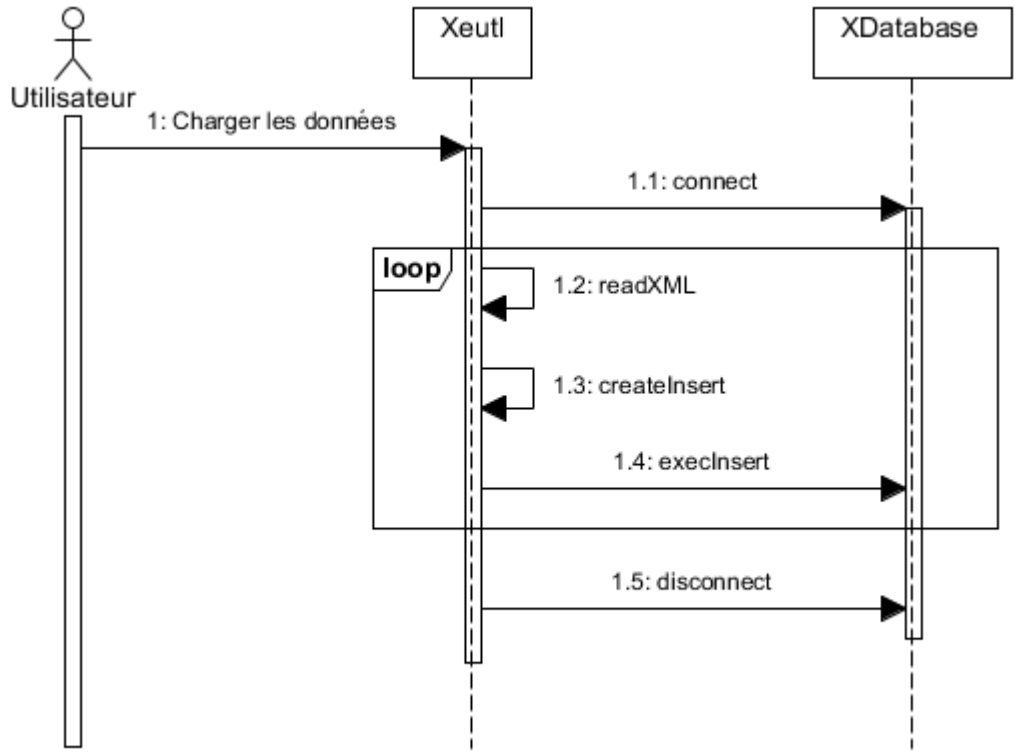


Figure 52 : diagramme de séquence « chargement des données dans la base cible »

5.2 Choix techniques

5.2.1 Java

Tout comme les bases de données sur lesquelles travaille XeuTL, les systèmes sur lesquels l'utilisateur peut être amené à utiliser XeuTL sont eux aussi hétérogènes. C'est pourquoi XeuTL a été conçu comme une application portable.
Pour répondre à ce besoin de portabilité, XeuTL a été développé en Java. Les avantages de Java dans le contexte de XeuTL sont multiples : robustesse, portabilité, performance, …

Autre apport essentiel de Java dans le cas de XeuTL : l'API JDBC. Cette API permet de se connecter et d'interroger de multiples sources de données via une interface commune. La plupart des bases de données relationnelles possèdent des pilotes JDBC qui permettent d'utiliser cette API.

JDBC propose notamment les interfaces *Statement* et *ResultSet* ; la première représente une requête SQL, la seconde permet de récupérer les enregistrements retournés par la requête.

5.2.2 XML

XeuTL créé et analyse des documents XML :

- fichiers de métadonnées,
- fichiers de données extraites et transformées,
- fichiers de définitions des correspondances.

Java fournit des outils efficaces pour traiter les documents XML.
Il existe globalement deux types d'API[21] permettant de traiter un documents XML :

- les API basées sur les arbres, qui fournissent une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et qui en permettent la manipulation (parcours, recherche et mise à jour),
- les API basées sur les évènements, qui déclenchent des évènements à l'analyse des documents XML, permettant ainsi à l'application d'effectuer les opérations voulues en fonction du type d'évènement reçu.

L'API DOM (Document Object Model) et la principale API basée sur les arbres. L'API SAX (Simple API for XML) est la plus connue des API basées sur les évènements.

DOM présente un document XML sous la forme d'une arborescence. La manipulation d'une telle arborescence est aisée :

- chaque élément, ou portion du document, composant l'arborescence est directement et rapidement accessible,
- la modification de l'arborescence est possible.

L'inconvénient majeur de la méthode DOM est qu'elle ne permet pas le traitement de documents de grande taille. Cette méthode nécessite en effet le chargement en mémoire de l'arbre complet des données. L'occupation mémoire d'un arbre XML chargé en mémoire selon la méthode DOM est estimée à 10 fois la taille du fichier XML représenté. Au delà de quelques centaines de Mo, les ressources du système utilisé peuvent rapidement arriver à saturation et rendre l'application inutilisable.

[21] Application Programming Interface

SAX traite les documents XML élément par élément, de manière séquentielle. Concrètement, lors du parcours d'un document XML, chaque élément est considéré comme un évènement auquel peut correspondre une méthode qui sera appelée par l'analyseur. Contrairement à DOM, il n'est pas possible d'accéder directement à un élément dans le document XML ce qui en fait l'inconvénient majeur de SAX. Les principaux atouts de SAX sont qu'il est peu gourmand en mémoire (l'arborescence du document XML n'est pas entièrement chargée en mémoire) et que le temps de traitement est très faible.

Les opérations d'extraction et de chargement de données peuvent porter sur de grandes quantités de données. Les API basées sur les évènements semblent donc être les plus appropriées pour ces opérations.

STAX (Streaming API for XML) s'inspire à la fois de DOM (possibilité d'écrire dans un document XML) et de SAX (accès séquentiel aux données, peu de ressources mémoire nécessaires), tout en se distinguant de ces deux API :

- la méthode d'écriture dans un document XML est différente entre DOM et STAX : l'arborescence XML doit être complètement chargée en mémoire dans le cas de DOM avant écriture, ce qui n'est pas le cas avec STAX.
- SAX fonctionne en « push », STAX en « pull » : en mode « push » l'analyseur XML (parseur) envoie les données XML à l'application au fur et à mesure qu'il les rencontre, sans avoir connaissance de l'état de l'application (est-elle prête à recevoir et traiter ces données ?). En mode « pull » l'application obtient les données XML lorsqu'elle en fait la demande. L'application peut filtrer des balises, stopper ou relancer l'analyse, …

Ce dernier point améliorant le contrôle est particulièrement intéressant dans le cadre de notre application.

STAX fournit deux API : une API de type évènementiel et une API de type curseur. Les deux API permettent le parcours des événements émis lors du parcours de documents XML. L'API de type événementiel émet des objets de type XMLEvent. Cette API est plus simple à mettre en place car les objets XMLEvent contiennent toutes les données nécessaires au traitement. L'API de type curseur émet des évènements de type entier. Celle-ci est plus performante car elle n'instancie pas un objet XMLEvent à chaque évènement.

Le choix final des technologies XML s'est porté sur deux interfaces clés de STAX particulièrement efficaces pour le traitements de documents XML :

- XMLStreamReader pour l'analyse de documents XML
- XMLStreamWriter pour la génération de documents XML

Ces deux interfaces proviennent de l'API de type curseur.

5.3 Architecture du système

La figure 53 présente l'architecture générale de XeuTL.

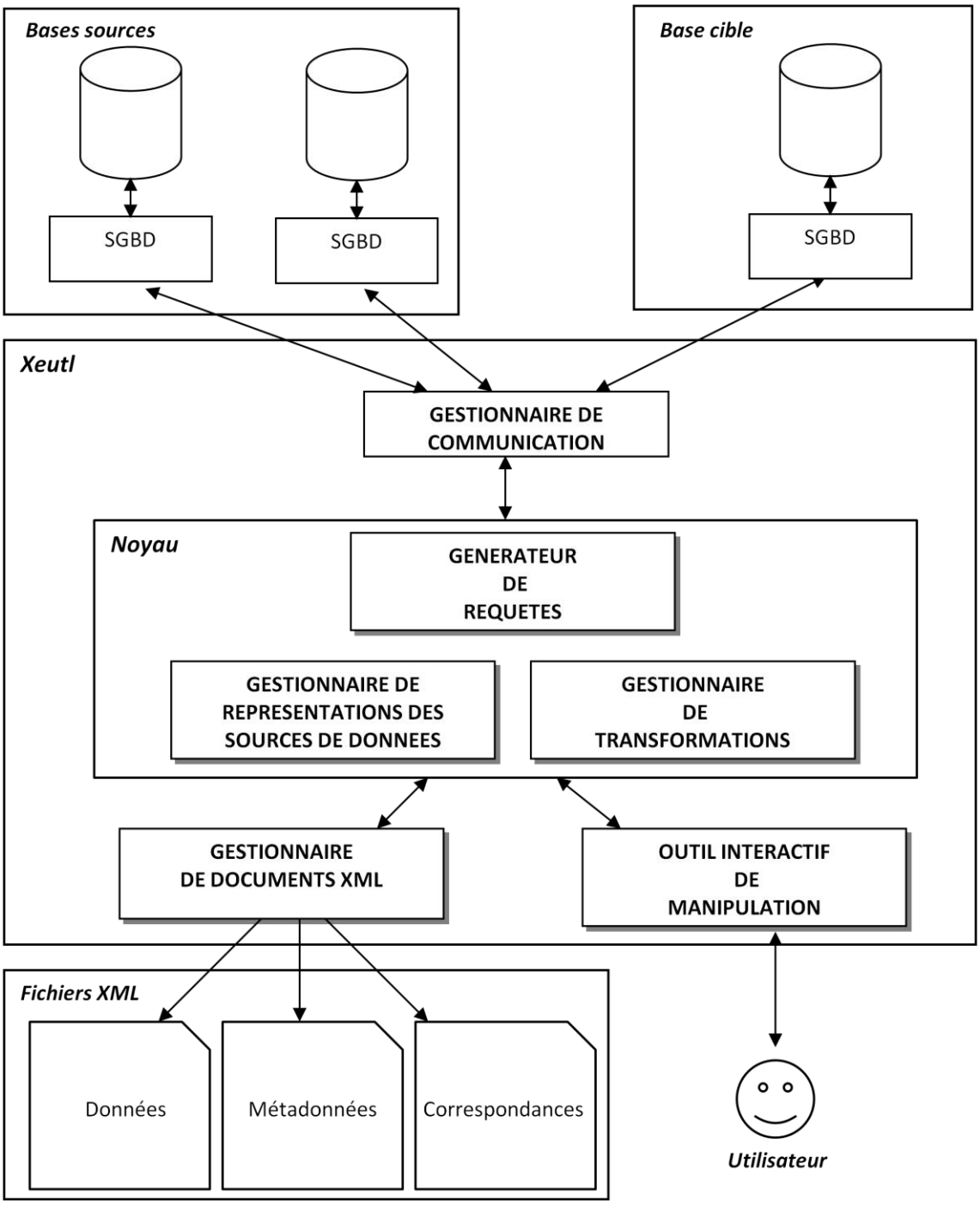


Figure 53 : architecture de XeuTL

XeuTL se compose d'un noyau chargé de réaliser l'intégration de données et de différents modules qui permettent au système d'interagir avec l'utilisateur, les sources de données et les fichiers XML utilisés par l'application.
Les trois modules implémentés autour du noyau sont :

- le gestionnaire de communication : il est chargé de dialoguer avec les différentes bases de données concernées. Il effectue les connexions aux différentes bases de données, exécute les requêtes SQL fournit les résultats au noyau ;
- le gestionnaire de documents XML : il permet de générer et d'analyser les documents XML utilisés par l'application. Il crée les fichiers XML lorsque cela est nécessaire, lit le contenu des fichiers XML et y écrit des données ;
- l'outil interactif de manipulation : il permet à l'utilisateur d'interagir avec le système. Deux modes de manipulation sont possibles : le premier via une API qui fournit l'ensemble des méthodes nécessaires à l'utilisateur pour manipuler XeuTL, le second via une interface graphique. Celle-ci étant en cours de développement, elle n'est pas présentée dans ce mémoire.

Le noyau de l'application est composé de trois modules :

- le générateur de requêtes : il est chargé de créer les ordres SQL qui permettent d'extraire, de transformer ou d'importer des données depuis ou vers une base de données ;
- le gestionnaire de représentations des sources de données : il permet de représenter en mémoire le schéma physique des bases de données à traiter ;
- le gestionnaire de transformations : il travaille en étroite collaboration avec le générateur de requêtes et le gestionnaire de représentations, afin de traiter les demandes de transformations paramétrées par l'utilisateur sous forme de correspondances.

5.4 Conception détaillée

5.4.1 Le gestionnaire de communication

Le gestionnaire de communication est essentiellement basé sur JDBC. Il met à la disposition du noyau différentes méthodes permettant :

- de se connecter à une base de données : il retourne alors un objet de type *Connection* à la base ;
- d'exécuter des ordres SQL : il retourne alors les données obtenues sous la forme d'un objet de type *Resultset* et/ou *ResultSetMetaData*. L'objet *ResultSet* contient les résultats d'une requête SQL, l'objet *ResultSetMetaData* contient les métadonnées du *ResultSet* associé (nom des colonnes du *ResultSet*, type des colonnes, …).

5.4.2 Le gestionnaire de documents XML

Pour chaque processus ETL, XeuTL produit trois types de fichiers XML :

- le fichier de données qui contient les données transformées et extraites des bases sources,
- les fichiers de métadonnées qui contiennent la description des schémas physiques des bases sources et cibles,
- le fichier de correspondances qui contient les correspondances définies par l'utilisateur.

Le gestionnaire de documents XML est essentiellement basé sur l'API STAX (cf. § 5.2.2). Ce module fournit au noyau des méthodes permettant de créer un fichier XML, de l'analyser, ou d'écrire dans celui-ci. L'analyse et l'écriture d'un fichier XML avec STAX sont relativement simples à mettre en place :

- cas de l'analyse : un parseur de type curseur est créé. Celui-ci parcours l'ensemble du fichier XML et renvoie le type d'élément rencontré. Ce type d'élément va définir quelles sont les méthodes du parseur qui sont accessibles pour cet élément. Les différentes méthodes permettent de récupérer le nom d'une balise XML, le texte qu'elle contient, les attributs qui lui sont affectés, etc. Le noyau effectue alors les opérations nécessaires au traitement des éléments rencontrés.
- cas de l'écriture : une fabrique[22] est nécessaire pour récupérer une instance d'un générateur XML STAX. Plusieurs méthodes permettent ensuite de créer des balises, d'y ajouter du texte, des attributs, etc.

[22] Une fabrique de création (*Factory* en langage Java) a pour rôle la construction d'objets qui implémentent une interface commune.

5.4.3 Le noyau

5.4.3.1 Préambule

Afin de mieux appréhender les exemples XML présentés dans ce chapitre, les différentes balises XML utilisées par le fichier de correspondances sont détaillées dans le tableau 4. Par souci de lisibilité, seules sont présentées les balises ouvrantes. Toutes les balises utilisées dans un fichier de correspondances XeuTL contiennent du texte et/ou d'autres balises. Aucune n'est affectée d'un attribut. La colonne « Parent » indique le numéro de la balise parent le cas échéant.

Numéro	Balise	Parent	Description
0	<*XeuTLMapping*>	-	Elément racine du fichier de correspondances
1	<*batch*>	0	Définition d'un lot de correspondances
2	<*id*>	1	Identifiant d'un lot de correspondances
3	<*transformation*>	0	Définition d'une correspondance
4	<*id*>	3	Identifiant d'une correspondance
5	<*batchId*>	3	Identifiant du lot dont fait partie la correspondance
6	<*dataSrcName*>	3	Nom de la base de données source
7	<*tableSrcName*>	3	Nom de la table source
8	<*colonneSrcName*>	3	Nom de la colonne source
9	<*tableDestName*>	3	Nom de la table cible
10	<*colonneDestName*>	3	Nom de la colonne cible
11	<*calc*>	3	Définition d'un calcul
12	<*type*>	11	Type de calcul
13	<*value*>	11	Valeur de l'opérande du calcul
14	<*cast*>	3	Définition d'un transtypage
15	<*fixedValue*>	3	Définition de l'application d'une valeur fixe
16	<*as*>	3	Nom temporaire (nécessaire pour certaines transformations)
17	<*reference*>	0	Définition d'une référence
18	<*refOrigSchema*>	18	Nom de la base de données source de la référence
19	<*refOrigtable*>	18	Nom de la table source de la référence
20	<*refOrigColonne*>	18	Nom de la colonne source de la référence
21	<*refCibleSchema*>	18	Nom de la base de données cible de la référence
22	<*refCibletable*>	18	Nom de la table cible de la référence
23	<*refCibleColonne*>	18	Nom de la colonne cible de la référence

Tableau 4 : balises XML utilisées dans le fichier de correspondances

5.4.3.2 Le gestionnaire de représentations

Le gestionnaire de représentations est chargé de représenter en mémoire le schéma physique des tables sources ou cibles concernées par le processus ETL.
Ce module est appelé à deux niveaux :

- lors de l'extraction des schémas physiques des bases,
- lors de la phase de définition des correspondances.

5.4.3.2.1 Extraction des schémas physiques

A ce niveau, les informations qui nous intéressent sont celles qui définissent le schéma physique des bases : liste des tables, caractéristiques de ces tables (nom, encodage, …) , liste des colonnes des tables, paramètres de ces colonnes (nom, type, taille, …). Dans le monde des systèmes de gestion de bases de données, ces informations sont communément appelées métadonnées.
Chaque système de gestion de bases de données utilise sa propre méthode pour stocker et organiser ces métadonnées. Heureusement, des outils existent pour uniformiser leur consultation. Nous avons utilisé JDBC (cf. § 5.2.1) pour la gestion de ces métadonnées.
Le gestionnaire de représentations interagit dans un premier temps avec le gestionnaire de communication.
Une fois que l'utilisateur a défini une connexion à une base de données, XeuTL en extrait les métadonnées et en déduit les bases, tables, et colonnes qui la composent. Lors de cette phase, une instance de la classe XDatabase est instanciée pour chaque base de données. XeuTL crée pour chaque table de la base de données un objet XTable lié à l'objet XDatabase concerné et pour chaque colonne un objet XColonne lié à l'objet XColonne concerné.
Dans un second temps, le schéma extrait est stocké dans des fichiers XML.
Le gestionnaire de représentations s'appuie alors sur le gestionnaire de documents XML.

Un extrait de fichier XML représentant le schéma physique d'une base de données généré par XeuTL est présenté sur la figure ci-dessous :

```
<table name="centre">
<colonne name="centre_id" type="VARCHAR" size="10" nullable="0"
isAutoIncrement="NO"/>
<colonne name="nom" type="VARCHAR" size="20" nullable="1"
isAutoIncrement="NO"/>
<colonne name="adresse" type="VARCHAR" size="255" nullable="1"
isAutoIncrement="NO"/>
</table>
<table name="medecin">
<colonne name="medecin_id" type="VARCHAR" size="10" nullable="0"
isAutoIncrement="NO"/>
<colonne name="centre_id" type="VARCHAR" size="20" nullable="1"
isAutoIncrement="NO"/>
<colonne name="nom" type="VARCHAR" size="20" nullable="1"
isAutoIncrement="NO"/>
<colonne name="prenom" type="VARCHAR" size="20" nullable="1"
isAutoIncrement="NO"/>
</table>
```

Figure 54 : extrait d'un schéma d'une base de données généré par XeuTL

5.4.3.2.2 Définition des correspondances

Lors de la phase de définition des correspondances, le gestionnaire de représentations interagit avec le gestionnaire de documents XML pour représenter en mémoire le schéma physique des bases du processus ETL. Le fichier XML est parcouru de manière séquentielle et les objets (XDatabase, Xtable, XColonne) créés au fur et à mesure des éléments rencontrés.

5.4.3.3 Gestionnaire de transformations

Le gestionnaire de transformations est en charge de gérer les correspondances définies par l'utilisateur. En fonction de ces correspondances, le gestionnaire de transformations va créer et appliquer des transformations sur les données. Ce module intervient à deux niveaux :

- lors de la définition des correspondances,
- lors de l'exécution d'un processus ETL.

5.4.3.3.1 Définition des correspondances

Un objet de type transformation est créé pour chaque correspondance définie par l'utilisateur entre les schémas source et cible. Dans le cas d'une correspondance complexe, plusieurs objets de type transformation peuvent être instanciés. Le gestionnaire de transformations crée ces objets en réponse aux demandes provenant de l'outil interactif de manipulation.
Une fois créé l'ensemble des transformations, le gestionnaire de transformations s'appuie sur le gestionnaire de documents XML pour stocker la description de l'ensemble des transformations définies sous la forme d'une fichier XML.

5.4.3.3.2 Exécution d'un processus ETL

Le système fait appel au gestionnaire de documents XML qui va fournir au gestionnaire de transformations la description des différentes transformations précédemment stockées dans le document XML correspondant. Le gestionnaire de transformations crée alors les objets *Transformation* correspondant aux descriptions stockées dans le fichier. Il travaille ensuite en étroite collaboration avec le générateur de requêtes. Ce dernier créé les ordres SQL de type *SELECT* qui vont permettre d'extraire les données des bases sources selon les transformations définies par l'utilisateur.

5.4.3.4 Le générateur de requêtes

Le générateur de requêtes est le module le plus complexe de XeuTL.
Cette section est découpée en deux parties :

- dans la première partie nous présentons le travail du générateur de requêtes lors de la phase d'extraction/transformation des données. La complexité du module se situe à ce niveau : le générateur, en fonction des correspondances définies par l'utilisateur, doit être capable d'établir des ordres SQL nécessaires. Ces ordres SQL sont des requêtes de type requête de *sélection* qui vont permettre d'extraire et de transformer les données des bases sources.
- dans la seconde partie nous abordons le travail du générateur lors de la phase de chargement. La complexité est moindre qu'à la phase précédente. Le générateur doit établir des requêtes de types *insertion* qui vont permettre de charger les données dans la base cible.

5.4.3.4.1 Phase d'extraction/transformation

5.4.3.4.1.1 Requêtes simples

Considérons l'ordre SQL suivant :

```
SELECT liste_de_colonnes FROM liste_de_tables
```

liste_de_colonnes liste les colonnes du résultat de l'exécution de l'ordre et *liste_de_tables* liste les tables sur lesquelles porte cet ordre.
L'exécution de cette simple requête permet d'extraire d'une base de données les données des colonnes *liste_de_colonnes* des tables *liste_de_tables*.
Il est possible de définir explicitement un nom pour la, ou les, colonne(s) de ces tables de résultat en utilisant l'opérateur *AS* comme ci-dessous :

```
 SELECT nomColonne AS nomColonneResultat FROM table
```

nomColonne désigne la colonne à interroger dans la table *table* de notre base de données. *nomColonneResultat* désigne le nom de la colonne que nous donnons au résultat obtenu.

Dans la plupart des SGBD *nomColonne* peut être écrit sous sa forme absolue, ou étendue, soit : *nomSchema.nomTable.nomColonne*. La colonne *nomColonne* se trouve dans la table *nomTable* qui est elle-même définie dans le schéma *nomSchéma*.

Toutes les requêtes générées par XeuTL utilisent la forme absolue, ou étendue, pour définir la liste des colonnes à interroger. Ces requêtes utilisent l'opérateur *AS* pour définir pour chaque

colonne résultat un nom unique et ainsi éviter d'obtenir deux colonnes de même nom (qui pourraient provenir de deux tables différentes) dans un même résultat.

Ce type de requête, dite requête simple, est généré par XeuTL lorsque la transformation à effectuer est de type atomique.

Une correspondance simple est présentée ci-dessous.

```
      <transformation>
            <id>2</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>centre</tableSrcName>
            <colonneSrcName>nom</colonneSrcName>
            <tableDestName>centre</tableDestName>
            <colonneDestName>nom</colonneDestName>
      </transformation>
```

Cette correspondance génère la requête suivante :

```
SELECT XeuTL_src.centre.nom
AS XeuTL_src_centre_nom
FROM XeuTL_src.centre
```

5.4.3.4.1.2 Requête de type calcul

Il est possible en SQL d'effectuer certains calculs mathématiques de base sur les données des colonnes, comme ci-dessous :

```
SELECT libellé, prix*1.3508 FROM produit
```

Supposons que `1,3508` soit le taux de conversion actuel de l'euro vers le dollars.
En exécutant cette requête sur une table `produit` contenant le `libellé` du produit associé à un `prix` en euros, nous obtiendrions l'équivalent en dollars US.
Cette correspondance est présentée ci-dessous.

```
      <transformation>
            <id>1</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>produit</tableSrcName>
            <colonneSrcName>prix</colonneSrcName>
            <tableDestName>produit</tableDestName>
            <colonneDestName>prix</colonneDestName>
            <calc>
                  <type>mutliply</type>
                  <value>1.3508</value>
            </calc>
      </transformation>
```

Cette correspondance génère la requête suivante :

```
SELECT XeuTL_src.produit.prix*1.3508
      AS XeuTL_src_produit_prix
FROM XeuTL_src.produit
```

5.4.3.4.1.3 Requête de type valeur fixe

Il est possible en SQL de fixer la valeur d'une colonne retournée comme suit :

```
SELECT nom,4 FROM patient
```

La définition d'une telle requête dans XeuTL est présentée ci-dessous.

```
      <transformation>
            <id>1</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>personne</tableSrcName>
            <colonneSrcName>nom </colonneSrcName>
            <tableDestName>personne</tableDestName>
            <colonneDestName>nom</colonneDestName>
      </transformation>
      <transformation>
            <id>2</id>
            <batchId>0</batchId>
            <fixedValue>4</fixedValue>
            <as>tmpName</as>
            <tableDestName>personne</tableDestName>
            <colonneDestName>numero</colonneDestName>
      </transformation>
```

Cette correspondance génère la requête suivante :

```
SELECT XeuTL_src.personne.nom AS XeuTL_src_personne_nom,
4 AS tmpName
FROM XeuTL_src.patient
```

5.4.3.4.1.4 Requête de type transtypage

SQL permet d'effectuer des comparaisons de types de données hétérogènes en modifiant le type de certaines colonnes. Cette opération est appelée transtypage.

Considérons le schéma de table suivant :

```
utilisateur (nom varchar(255), matricule varchar(20))
```

Si tous les matricules sont des entiers positifs, nous pouvons alors effectuer la requête suivante (syntaxe à adapter selon le SGBD concerné) :

```
SELECT nom, CAST(matricule AS SIGNED) FROM utilisateur
```

Le matricule de résultat sera alors de type `SIGNED`.
A noter que cette opération est impossible si certains utilisateurs possèdent, dans notre base de données, un matricule contenant des caractères alphabétiques (exemple : « A534 »).
Plus généralement le transtypage n'est pas possible dans tous les cas. Il doit y avoir compatibilité entre le typage des données sources et des données cibles.

Voici un exemple de correspondance de type transtypage dans XeuTL :

```
      <transformation>
            <id>1</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>centre</tableSrcName>
            <colonneSrcName>centre_id</colonneSrcName>
            <tableDestName>centre</tableDestName>
            <colonneDestName>centre_id</colonneDestName>
            <cast>SIGNED</cast>
      </transformation>
```

Cette correspondance génère la requête suivante :

```
SELECT CAST (XeuTL_src.centre.centre_id AS SIGNED)
      AS XeuTL_src_centre_centre_id
FROM XeuTL_src.centre
```

5.4.3.4.1.5 Requête avec jointure

Une jointure est un sous-ensemble d'un produit cartésien de deux ou plusieurs tables.
SQL permet d'effectuer des jointures entre tables, voire, pour certains SGBD, entre schémas.

Considérons les deux schémas de tables suivants :

```
Patient (nom_patient varchar(255), nationalite_id int(10))
Pays (pays_id int(10), nom_pays varchar(10))
```

Supposons que `nationalite_id` est une clé étrangère référençant `pays_id`.
Nous pouvons alors écrire la requête suivante :

```
SELECT nom_patient, nom_pays
FROM Patient LEFT JOIN pays ON nationalite_id=pays_id
```

Cette requête permet d'obtenir une liste des noms des patients de la table patient avec, pour chacun, le nom du pays de leur nationalité.

Voici un exemple de déclaration d'une correspondance de type jointure dans XeuTL :

```
      <transformation>
            <id>1</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>patient</tableSrcName>
            <colonneSrcName>nom_patient</colonneSrcName>
            <tableDestName>patient</tableDestName>
            <colonneDestName>nom</colonneDestName>
      </transformation>
      <transformation>
            <id>2</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>pays</tableSrcName>
            <colonneSrcName>nom_pays</colonneSrcName>
            <tableDestName>patient</tableDestName>
            <colonneDestName>nationalite</colonneDestName>
      </transformation>
      <reference>
            <refOrigSchema>XeuTL_src</refOrigSchema>
            <refOrigtable>patient</refOrigtable>
            <refOrigColonne>nationalite_id</refOrigColonne>
            <refCibleSchema>XeuTL_src</refCibleSchema>
            <refCibletable>pays</refCibletable>
            <refCibleColonne>nom_pays</refCibleColonne>
      </reference>
```

Cette correspondance génère la requête suivante :

```
SELECT XeuTL_src.patient.nom
            AS XeuTL_src_patient_nom,
      XeuTL_src.pays.nom
            AS XeuTL_src_pays_nom
FROM XeuTL_src.patient
      LEFT JOIN XeuTL_src.pays
            on XeuTL_src.patient.nationalite_id=
                  XeuTL_src.pays.pays_id;
```

5.4.3.4.1.6 Requête imbriquée

Considérons le schéma de base de données suivant :

```
Personne(personne_id int(10), nom varchar(255))
Vehicule(proprietaire_id int(10), immatriculation varchar(255))
```

`proprietaire_id` est une clé étrangère référençant `personne_id`.

Imaginons que nous désirons compter le nombre de véhicules que possède chaque personne de la table `personne`. La requête suivante répond à notre demande:

```
SELECT nom,
(SELECT COUNT(*) FROM vehicule WHERE proprietaire_id=personne_id)
FROM personne
```

Ce type de requête est extrêmement utile dans le contexte des processus ETL. Il permet à l'utilisateur d'effectuer des transformations complexes et d'obtenir dans la base cible une information qui n'est pas directement accessible dans les bases sources (exemple : le nombre de véhicules d'une personne n'est pas directement accessible dans la base source de notre exemple, mais le sera dans la base cible une fois le processus ETL exécuté).

XeuTL permet de définir ce type de requête de la manière suivante :

Une correspondance de type requête imbriquée est présentée ci-dessous.

```
      <transformation>
            <id>1</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>personne</tableSrcName>
            <colonneSrcName>nom </colonneSrcName>
            <tableDestName>personne</tableDestName>
            <colonneDestName>nom</colonneDestName>
      </transformation>
      <transformation>
            <id>2</id>
            <batchId>0</batchId>
            <sql>
select count(*)
from vehicule
where proprietaire_id=personne_id
<sql>
<as>tmpName</as>
            <tableDestName>personne</tableDestName>
            <colonneDestName>nb_vehicule</colonneDestName>
      </transformation>
```

Cette correspondance génère la requête suivante :

```
SELECT XeuTL_src.personne.nom
            AS XeuTL_src_personne_nom,
      (SELECT COUNT(*)
       FROM XeuTL_src.vehicule
       WHERE XeuTL_src.vehicule.proprietaire.id =
            XeuTL_src.personne.personne_id)
      AS tmpName
FROM XeuTL_src.personne
```

5.4.3.4.1.7 Regroupement

Comme nous avons pu le voir précédemment, le fichier XML obtenu à l'issue de la phase d'extraction/transformation/stockage des données respecte le schéma défini par la base cible. Pour obtenir ce résultat, l'exécution séquentielle des transformations définies dans le fichier de correspondances n'est pas suffisant. Il est en effet nécessaire d'effectuer des regroupements par tables sources et/ou cibles afin d'obtenir un résultat cohérent.

L'exemple ci-dessous présente cette problématique.

Considérons les schémas de tables sources et cibles suivants :

Table source :

```
Personne(nom varchar(255), prenom varchar(255))
```

Table cible :

```
Personne(nom varchar(255), prenom varchar(255))
```

Paramétrons les deux correspondances qui vont permettre d'effectuer la migration complète des données de la table source vers la table cible comme suit :

```
      <transformation>
            <id>1</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>personne</tableSrcName>
            <colonneSrcName>nom</colonneSrcName>
            <tableDestName>personne</tableDestName>
            <colonneDestName>nom</colonneDestName>
      </transformation>
      <transformation>
            <id>2</id>
            <batchId>0</batchId>
            <dataSrcName>XeuTL_src</dataSrcName>
            <tableSrcName>personne</tableSrcName>
            <colonneSrcName>prenom</colonneSrcName>
            <tableDestName>personne</tableDestName>
            <colonneDestName>prenom</colonneDestName>
      </transformation>
```

Sans regroupement, voici les requêtes SQL de sélection que l'on pourrait déduire de ces correspondances :

```
SELECT XeuTL_src.personne.nom AS
XeuTL_src_personne_nom FROM XeuTL_src.personne
SELECT XeuTL_src.personne.prenom AS
XeuTL_src_personne_prenom FROM XeuTL_src.personne
```

Le résultat obtenu avec ces deux requêtes serait composé de deux tables distinctes : une première contenant tous les noms des personnes, une seconde tous les prénoms et le lien entre le nom et le prénom des personnes serait alors perdu.

Le regroupement effectué par le générateur de requêtes permet d'obtenir la requête suivante :

```
SELECT XeuTL_src.personne.nom AS XeuTL_src_personne_nom,
 XeuTL_src.personne.prenom AS XeuTL_src_personne_prenom
FROM XeuTL_src.personne
```

Le résultat d'une telle requête serait alors une table contenant le nom et le prénom des personnes. La relation est donc conservée, ce qui correspond à notre besoin.

5.4.3.4.1.8 Portabilité des ordres SQL

Chaque SGBD utilise une implémentation du langage SQL qui lui est propre (cf. § 3.3.3). Un ordre SQL valide sur un SGBD ne le sera pas forcément sur un autre. Afin de minimiser ces problèmes de compatibilité, il convient de respecter au mieux la norme SQL-92 dans les ordres SQL générés (cf. § 4.1.4). Toutefois les méthodes du langage peuvent varier d'un SGBD à l'autre (exemple : une concaténation en Mysql ou Oracle s'effectue avec un appel à la méthode *concat ;* en SQL Server celle-ci est effectuée en utilisant l'opérateur '+').
XeuTL se base sur le type du SGBD de chaque source de données pour générer des ordres SQL compatibles avec celui-ci. Il s'appuie pour cela sur un traducteur fournissant des équivalences de commandes SQL entre différents SGBD.

5.4.3.4.1.9 Algorithme

L'algorithme permettant de générer et traiter ces requêtes se décompose en deux parties :

- la génération des requêtes SQL de type SELECT,
- l'exécution de ces requêtes et le stockage en parallèle des résultats obtenus dans un fichier XML.

Chaque ordre SQL généré est directement exécuté sur les bases sources. L'algorithme est présenté figure 55.

```
pour chaque batch {
      // creation
      pour chaque correspondance {
            si (correspondanceType = 'copy1to1'){
                  // contruction de la partie SELECT :
                        - construction d'un CAST si nécessaire
                        - construction d'un ou plusieurs CALC si nécessaire
                        - construction du AS // ex: xxx AS yyy
                  // construction de la partie FROM:
                        - construction du FROM
                  si le champ est une référence est définie pour le champ:
                        - construction d'un LEFT JOIN
                  si le FROM n'existe pas déjà on l'ajoute
                  on ajoute le SELECT
            }
            si (correspondanceType = 'aggregToOne' ou 'valueToOne'){
                  construction du SELECT
            }
      }
      - creation de la requete complete
      - execution de la requete
      // parcours des resultat:
      pour chaque ligne
            pour chaque table cible
                  pour chaque correspondance par table cible
                        construction du INSERT INTO
                        si (type = 'copy1to1') construction du VALUES
                        si (type = 'valueToOne') {
                              si (type = 'reference) gestion de la reference
                        }
                        si (type = 'aggregToOne'){
                              output = tmpName
                        }
                  }
            }
      }
```

Figure 55 : algorithme de génération et de traitement de requêtes

5.4.3.4.2 Phase de chargement

Tout le travail de transformation des données a été effectué lors de la phase d'extraction/transformation. Ici le générateur de requêtes traite les données fournies par le gestionnaire de documents XML pour en déduire les ordres SQL de type « INSERT » qui vont permettre le chargement des données dans la base cible. La structure des données qu'il reçoit respectant déjà le schéma de la base cible, le traitement est linéaire : à chaque ligne lue dans le fichier XML de données, le générateur établit un ordre SQL d'insertion qu'il exécute immédiatement.

Il existe une exception à ce fonctionnement : lors du traitement d'un enregistrement nécessitant de générer une clé de substitution (cf. § 4.1.5) le générateur de requêtes exécute un ordre SQL sur la base cible afin de calculer la valeur de cette clé et vérifier qu'elle n'est pas déjà utilisée dans la table concernée.

6 BILAN ET PERSPECTIVES

6.1 Bilan

6.1.1 Rappel des objectifs

Le travail réalisé au cours de ce mémoire a consisté en une étude de l'intégration de données et la réalisation d'un outil d'intégration de données : l'ETL XeuTL. L'étude a permis de mettre en évidence la problématique de l'intégration de données. L'hétérogénéité sémantique a notamment été abordée et différentes solutions pour la résoudre ont été présentées.

Les outils de type ETL permettent d'intégrer des données en s'appuyant sur une approche matérialisée. Ils sont actuellement nombreux sur le marché et concernent essentiellement le marché des entreprises. Afin d'étendre les fonctionnalités d'ISIS à l'intégration de données nous avons choisi de développer un outil spécifique : XeuTL.

6.1.2 Retour sur la conception et la réalisation

Les choix en termes d'architecture, de structures de données et de langages de programmation qui ont découlé de la phase de conception n'ont pas été remis en cause lors du développement de l'application.

Au niveau de la réalisation, quelques obstacles techniques, notamment au niveau de la charge mémoire, ont nécessité une attention particulière afin d'être levés et plusieurs prototypes ont été réalisés dans le but de valider les différentes couches de l'application à réaliser.

6.1.2.1 Phase de transformation

L'étude et la réalisation de la phase de transformation ont nécessité la réalisation de trois prototypes, décrits ci-dessous.

Prototype 1 : basé sur des transformations XSLT:
Les transformations XSLT sont couramment utilisées dans le monde du web. Le premier prototype de XeuTL est basé sur ces transformations. Voici la description du fonctionnement des phases d'extraction et de transformation de ce premier prototype :

1. L'application extrait les données brutes des bases sources.
2. Les données brutes sont stockées dans un fichier XML.
3. L'application traduit les correspondances définies par l'utilisateur en transformations XSLT.
4. L'application exécute ces transformations sur le document XML.

Cette méthode s'est avérée inefficace. Les moteurs XSLT testés (SAXON[23], Xalan[24]) chargent par défaut les documents XML en mémoire afin d'appliquer les transformations XSLT. Les documents à traiter étant de grande taille, la mémoire est rapidement saturée. Il existe une alternative permettant d'effectuer des transformations XSLT en utilisant des techniques de streaming. Néanmoins les performances estimées dans le cas de transformations complexes sont assez mauvaises.

Prototype 2 : basé sur une base XML native :
Afin de s'affranchir des problèmes de mémoire rencontrés avec le premier prototype, nous avons utilisé une base native XML. Voici la description du fonctionnement des phases d'extraction et de transformation du deuxième prototype :

1. L'application extrait les données brutes des bases sources.
2. Les données brutes sont stockées dans un fichier XML.
3. Le fichier XML est chargé dans la base native XML.
4. L'application traduit les correspondances définies par l'utilisateur en requêtes basées sur le langage XQuery.
5. L'application exécute les requêtes XQuery sur la base XML native.

Avec des requêtes retournant beaucoup d'enregistrements, la mémoire du système arrive à saturation. Les requêtes de type agrégation et de type imbriqué sont mal gérées par ce type d'environnement. Cette méthode s'avère inefficace dans notre cas.

Une technique permettant de résoudre ces problèmes de charge mémoire consisterait à découper les documents XML en documents XML de petite taille. Cependant, exécuter nos transformations XSLT (cas du premier prototype) ou nos requêtes XQuery (cas du second prototype) sur une multitude de documents XML compliquerait grandement la tâche et réduirait considérablement les performances.

Prototype 3 : basé sur la génération de requêtes SQL :
Appliquer nos transformations sur un fichier XML de données brutes ne semble pas convenir pour l'application que nous réalisons. Nous avons alors décidé d'explorer une nouvelle direction avec un troisième prototype tirant parti du langage SQL, et en intégrant la phase de transformation à celle d'extraction. Le prototype génère des ordres SQL formulés de manière à appliquer les transformations définies par l'utilisateur sous forme de correspondances. Voici la description du fonctionnement des phases d'extraction et de transformation du troisième prototype :

1. L'application traduit les correspondances définies par l'utilisateur en requêtes basées sur le langage SQL
2. L'application exécute ces requêtes sur les sources de données
3. L'application stocke les données obtenues dans un fichier XML

Le fichier XML final contient les données transformées. La phase de chargement peut s'effectuer de manière séquentielle. Nous n'observons plus de problèmes de performances liés aux recherches complexes effectuées dans un document XML. Ce troisième prototype est validé.

[23] http://saxon.sourceforge.net/
[24] http://xml.apache.org/xalan-j/

6.1.2.2 Gestion de gros volumes de données

La gestion de gros volumes de données est un problème majeur en intégration de données. XeuTL répond à cette problématique de manière efficace. Les traitements potentiellement sources d'une importante consommation mémoire (extraction, transformation, chargement) s'effectuent en flux continu.

Ces traitements reposent sur l'utilisation conjointe de certaines spécificités de l'API JDBC et de STAX :

- l'API JDBC fournit l'interface *Statement*. L'exécution d'une requête SQL nécessite d'appeler une classe spécifique au pilote utilisé implémentant cette interface. L'interface *Statement* retourne les enregistrements du résultat d'une requête par blocs. En paramétrant une faible taille de bloc les échanges entre l'application et le SGBD sont plus nombreux mais l'application traitera les données en flux continu. Ceci permet de traiter des ordres SQL retournant un très grand nombre d'enregistrements sans risquer de surcharger la mémoire du système.
- plusieurs librairies ont été testées pour la gestion de fichiers XML de grande taille. L'API STAX, s'est avérée très efficace pour le traitement de gros volumes de données. STAX ne construisant pas d'arbre XML en mémoire, sa charge reste faible quelque soit la taille du document XML à traiter.

Lors de la phase d'extraction/transformation, un ordre SQL généré est directement exécuté sur la base source et les enregistrements obtenus stockés ligne par ligne dans le fichier XML d'extraction de données (via l'API STAX). L'ordre suivant est ensuite traité de manière identique.
Lors de la phase de chargement, les ordres SQL sont directement exécutés sur la base cible au fur et à mesure qu'ils sont déduits du parcours du fichier XML de données extraites.
Ces différentes mesures permettent de conserveur une consommation mémoire tout à fait raisonnable.

6.1.3 Résultats

L'étude a participé à l'orientation de plusieurs travaux au sein du laboratoire TIMC, dont deux thèses soutenues en 2009, [Ahm09] et [Ker09], sur le thème de l'intégration de données. Elle a également été intégrée à un état de l'art sur l'intégration de données dans le cadre du projet européen NOESIS : *Platform for a wide scale integration and visual representation of medical intelligence*[25]**.** Un extrait de cette étude est présenté en annexe 2.

Un outil ETL peut proposer un très grand nombre de fonctionnalités. Un de nos objectif étant de réaliser un outil pour mettre en place des processus ETL, nous avons fait le choix de réaliser des fonctionnalités orientées vers l'intégration de données. A l'usage, celles-ci s'avèrent efficaces pour la mise en place de processus ETL standard.

[25] http://www.noesis-eu.org

6.2 Perspectives d'évolution

6.2.1 ISIS

L'équipe Osiris souhaite enrichir ISIS d'outils pour l'intégration de données. Ce stage s'inscrit dans cette perspective. Dans le cadre du mémoire nous avons implémenté une solution basée sur une approche matérialisée. Cette approche est intéressante dans certains cas d'utilisation: migration de données, alimentation de base cible ou d'entrepôt de données.
L'approche virtuelle (ou médiateur) répond à des besoins différents : recherche d'information dans un grand nombre de sources de données, nécessité de consulter des données toujours à jour, etc.

Dans le cadre du projet ISIS, il semblerait intéressant de compléter l'outil par la réalisation d'une solution de type médiateur. Pour réaliser l'intégration, un médiateur se base sur des correspondances définies entre les schémas sources et le schéma cible. XeuTL permet à l'utilisateur d'établir des correspondances entre les schémas sources et le schéma cible. La réalisation d'un médiateur dans le cadre du projet ISIS pourrait s'inspirer de la gestion des correspondances effectuée dans XeuTL. Dans le cas de XeuTL ces correspondances servent à extraire des bases sources les données transformées; dans le cas d'un médiateur elles pourraient être utilisées pour reformuler les requêtes des utilisateurs afin d'interroger les sources de données.

6.2.2 XeuTL

6.2.2.1 Evolution standard

Afin de mettre en place des processus ETL avancés, il sera intéressant de réaliser :

- un module de gestion d'erreurs avancé qui permettra d'identifier les données qui n'auraient pas été correctement traitées lors de l'exécution du processus. Il faudra envisager la possibilité de les soumettre à nouveau après application de nouvelles transformations si nécessaire.
- un module de suivi d'exécution des processus ETL, qui permettra un suivi en temps réel du déroulement de l'exécution du processus ETL. Ce module devra permettre de reprendre l'exécution d'un processus interrompu accidentellement.
- un planificateur de processus ETL, qui permettra de planifier le déclenchement de plusieurs processus ETL et d'obtenir des comptes-rendus d'exécution des processus ETL.

XeuTL est compatible avec les SGBD pour lesquels un driver JDBC est disponible, ce qui est le cas de la quasi-totalité des SGDB relationnels du marché.
Les données en entreprise ne sont pas uniquement stockées dans des bases de données relationnelles. Les sources de données sont souvent des fichiers plats exportés depuis diverses applications. Il serait intéressant de se pencher sur la question de l'intégration de ces fichiers.
Les fichiers en question doivent être un minimum structurés.
Un fichier CSV extrait de Microsoft Excel, par exemple, est potentiellement intégrable car il possède une certaine structure :

- les champs sont séparés par des ' ;',
- les enregistrement sont séparés par des retours chariot,
- selon la configuration de l'export, la première ligne peut indiquer le nom des colonnes du tableau exporté.

L'intégration de fichiers XML est aussi envisageable. XeuTL est pensé comme un ETL destiné aux bases de données relationnelles. L'intégration de fichiers XML avec XeuTL nécessiterait une modification de plusieurs couches de l'application. Le générateur de requêtes repose essentiellement sur l'utilisation du langage SQL. Il est envisageable d'enrichir le générateur afin qu'il traite les requêtes exprimées en XQuery. Le gestionnaire de transformations serait en grande partie réutilisable ainsi que le gestionnaire de communication et le gestionnaire de fichiers XML. Afin de permettre l'intégration de fichiers XML, il conviendra de se pencher sur le traitement des schémas XML. Le schéma XML est un fichier XML décrivant la structure d'un document XML. Avec XeuTL, l'utilisateur définit des correspondances entre les bases sources et la base cible en ce basant sur les schémas physiques des bases. Pour l'intégration de fichiers XML l'utilisateur devra définir ses correspondances entre les schémas XML décrivant la structure des fichiers XML sources et le schéma physique de la base cible.

Si tous les SGBD relationnels peuvent être interrogés en utilisant des requêtes SQL, il est important de noter que chaque fournisseur de SGBD propose une implémentation du langage SQL qui lui est propre. Une requête SQL valide dans un environnement ne le sera pas forcément sur l'environnement d'un concurrent. Pour s'affranchir de ce problème nous nous sommes basés sur l'API JDBC. Cette API propose un niveau d'abstraction qui permet d'interagir avec différents SGBD de manière uniforme (tout SGBD proposant un driver JDBC). Cependant, JDBC a ses limites :

- afin de rester compatible avec tous les SGDB, les requêtes SQL exécutées sur les sources de données doivent rester relativement génériques,
- la gestion des erreurs est très lourde.

L'utilisation d'un framework type Spring[26] ou Hibernate[27], permettrait d'offrir une couche d'abstraction supplémentaire : la compatibilité avec les SGBD est améliorée et la gestion des erreurs simplifiées. Cette évolution de XeuTL est donc à étudier.

Les ontologies sont souvent utilisées pour l'intégration de données dans le cadre de systèmes à base de médiateurs. Il serait intéressant d'étudier l'utilisation d'ontologies dans le cadre d'un système basé sur une approche matérialisée. En effet, l'utilisation d'ontologies pourrait être envisagée comme une aide apportée à l'utilisateur dans la phase de définition des correspondances entre les schémas sources et cible. Cela permettrait, par exemple, d'indiquer à l'utilisateur que deux schémas sources présentent des concepts proches et qu'il est potentiellement possible de combiner les données des sources concernées.

6.2.2.2 Evolution vers les entrepôts de données

Une orientation future de XeuTL pourrait être son utilisation dans le cadre de la mise en place de processus ETL visant à alimenter des entrepôts de données (cf. § 2.2.3.2). L'utilisation d'un entrepôt dans un but décisionnel nécessite, selon les besoins, des processus ETL particuliers. Ces processus reposent en règle générale sur :

- une fréquence de rafraîchissement élevée (fréquence qui peut s'approcher du temps réel selon les cas),
- d'excellentes performances dans la durée d'exécution de leur cycle (extraction, transformation, chargement),
- l'ajout d'une phase de nettoyage de données.

[26] http://www.springsource.org/
[27] http://www.hibernate.org/

Une fréquence de rafraîchissement élevée dépend notamment des performances dans la durée d'exécution des processus. Les solutions actuellement proposées pour répondre à ces deux problèmes sont la mise en place des techniques de parallélisme et de pipeline. Ces techniques sont présentées dans [NET06]. Dans le cadre d'un ETL la technique du parallélisme a pour but d'exécuter plusieurs opérations en même temps. Elle repose sur l'utilisation de plusieurs processeurs sur une même machine ou sur l'utilisation de plusieurs machines pour effectuer une opération. La technique du pipeline permet l'exécution d'une nouvelle opération avant que la précédente soit terminée.

L'entrepôt de données a pour but l'aide à la décision, et la qualité des données qu'il présente à l'utilisateur est primordiale. Le nettoyage des données permet d'identifier les données incorrectes, incomplètes, imprécises, inadaptées, …. et de les remplacer, les modifier ou les supprimer. La phase de nettoyage de données est une phase critique du processus ETL destiné à l'alimentation d'un entrepôt de données. Cette phase s'intègre à la phase de transformation et va apporter de la valeur aux données en garantissant leur exactitude et leur pertinence.
Dans ce cadre, l'ETL doit être capable de corriger, rejeter ou charger les données telles quelles et fournir à l'utilisateur des moyens permettant de définir des règles de nettoyage. Le nettoyage de données se décompose en cinq phases selon [RD00] :

- analyse de données : permet d'identifier potentiellement les données à nettoyer,
- définition de transformations : ces transformations particulières sont destinées au nettoyage des données (exemple : dédoublonnage),
- vérification : les transformations sont testées et validées afin d'en garantir l'efficacité,
- transformation : les transformations sont appliquées sur les données,
- propagation sur les sources de données : les données nettoyées sont, quand cela est possible, propagées sur les sources originales afin qu'elles bénéficient des données « propres » et que le nettoyage effectué ne soit pas à refaire lors d'une prochaine phase de nettoyage.

Dans le cadre d'une évolution de XeuTL orientée vers les entrepôts de données, nous pourrions implémenter de nouvelles fonctionnalités permettant d'effectuer l'analyse et la vérification de données, tout en proposant des transformations propres au nettoyage des données.

7 CONCLUSION

Depuis l'adoption des bases de données par les entreprises, l'intégration de données a toujours été au centre de leurs préoccupations. L'intégration de données doit faire face à la problématique de l'hétérogénéité sémantique. Il existe deux approches permettant de résoudre cette hétérogénéité : l'approche virtuelle et l'approche matérialisée. L'approche matérialisée s'appuie sur des outils appelés ETL. Ceux-ci permettent d'extraire des données de plusieurs sources de données, et de les transformer afin de les charger dans une base cible. Les ETL sont nombreux sur le marché mais la plupart des solutions sont lourdes, propriétaires et onéreuses.

L'équipe OSIRIS souhaite enrichir le projet ISIS de fonctionnalités d'intégration. Dans ce cadre une étude du domaine de l'intégration a été réalisée, et XeuTL, un outil de type ETL a été développé. Celui-ci se base sur des correspondances pour résoudre l'hétérogénéité sémantique des sources de données. L'utilisateur définit ces correspondances entre les schémas physiques des bases sources et celui de la base cible. XeuTL déduit de ces correspondances les transformations qu'il applique sur les données des bases sources afin d'alimenter la base cible. Une importance particulière a été donnée aux capacités de traitement de gros volumes de données. XeuTL est un outil d'intégration inspiré de l'approche matérialisée. Il pourrait être intéressant, dans le cadre du projet ISIS, de lui adjoindre un outil d'intégration complémentaire inspiré de l'approche virtuelle.

L'intégration de données est un vaste sujet, étudié maintenant depuis plus d'une trentaine d'années. Les recherches sont nombreuses, tout comme les produits existants. Les frontières entre le domaine de l'intégration de données est celui de l'informatique décisionnelle ou de la gestion de la qualité des données ne sont pas clairement définies. Les besoins des entreprises dans ces domaines ont donné naissance à de nombreuses solutions de type ETL, EAI[28] ou EII[29]. Là encore, la frontière entre ces différents types d'outils est floue, comme l'indique la figure 56 tirée de [Imh05].

[28] Enterprise Application Integration
[29] Enterprise Information Integration

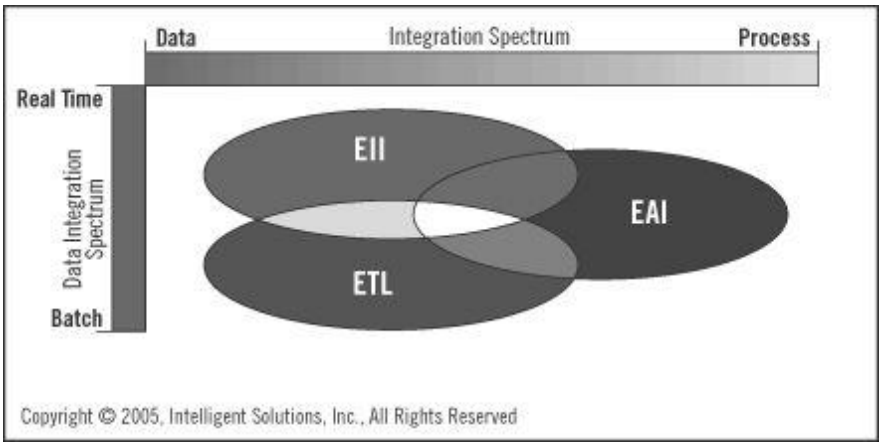


Figure 56 : les types de solution d'intégration

Assimiler et maîtriser ce vaste domaine a constitué la principale difficulté de ce stage. L'apport de mes tuteurs a été très important à ce niveau. Ils m'ont aidé à cadrer le projet.

L'intégration de données et l'utilisation d'outils ETL font partie de mon activité professionnelle actuelle. Ce stage m'a été d'un apport significatif à ce niveau : j'ai pu découvrir de nouvelles techniques d'intégration et m'enrichir de sérieuses connaissances théoriques.

La réalisation de XeuTL m'a permis de me perfectionner dans l'utilisation des technologies Java et XML et j'essaie maintenant, dans le cadre de mon travail, de pousser à l'utilisation de ces technologies lorsque celles-ci sont adaptées aux besoins. J'ai aussi amélioré ma maîtrise de différents outils de développements comme l'IDE[30] Eclipse ou le gestionnaire de sources Subversion.

[30] Integrated Development Environment (Environnement de développement intégré)

Bibliographie

[Ahm09] H. Ahmad : "*Une approche matérialisée basée sur les vues pour l'intégration de documents XML*", Thèse, Université Joseph Fourier – Grenoble I, (2009).

[BT98] M. Bokun and C. Taglienti. "*Incremental Data Warehouse Updates*". DM Review Magazine, (1998).

[CCG02] A. Calì, D. Calvanese, G. De Giacomo, M. Lenzerini: "*On the Expressive Power of Data Integration Systems*", ER 2002: 338-350, (2002).

[CKT01] C. Convey, O. Karpenko, N. Tatbul : "*Data Integration Services*", Technical Report, Brown University Computer Science Department, (2001).

[FKL04] E. Franconi, G. Kuper, A. Lopatenko, I. Zaihrayeu "*Queries and Updates in the coDB Peer to Peer Database System*", Submitted, (2004).

[Gru93] T. R. Gruber. "*A translation approach to portable ontologies*". Knowledge Acquisition, 5(2):199-220, 1993.

[HIS03] A.Y. Halevy, Z.G. Ives, D. Suciu, I. Tatarinov: "*Schema Mediation in Peer Data Management Systems*", ICDE 2003: 505-, (2003).

[HK95] Hull, R., King, R.: "*Reference architecture for the intelligent integration of information*", Rapport technique (ARPA), 1995

[Hul97] R. Hull: "*Managing Semantic Heterogeneity in Databases: A Theoretical Perspective*", In Proc. of the ACM Symp. on Principles of Database Systems (PODS), pages 51--61, (1997).

[Imh05] C. Imhoff : "*Understanding the Three E's of Integration*" DM Review, (April 2005).

[Ive03a] Z.G. Ives: "*Peer-to-Peer Schema Mediation*", University of Pennsylvania, (2003).

[Ker09] S. Kermanshahani : "*IXIA (IndeX-based Integration Approach) A Hybrid Approach to Data Integration*", Thèse, Université Joseph Fourier – Grenoble I, (2009).

[Kno03] C. Knoblock: "*View Integration*", University of Southern California 1, (2003).

[KS96] V. Kashyap, A. Sheth, "*Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies*", in Cooperative Information Systems: Current Trends and Directions, (1996).

[Len02] M. Lenzerini : "*Data Integration: A Theoretical Perspective*", (2002).

[Len03] M. Lenzerini: "*Information integration"*, Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003, Acapulco, Mexico, (2003).

[LW00] A.Y. Levy, D.S. Weld: "*Intelligent Internet systems*", Artif. Intell. 118(1-2): 1-14 (2000).

[MAB03] A. Maier, J. Aguado, A. Bernaras, I. Laresgoiti, C. Pedinaci, N. Peña, T. Smithers: "*Integration with Ontologies*", Conference Paper WM2003, April 2003, Luzern, (2003).

[MBP03] P. McBrien, A. Poulovassilis: "*Defining Peer-to-Peer Data Integration Using Both as View Rules*", DBISP2P 2003: 91-107, (2003).

[PMT03] V. Papadimos, D. Maier, K. Tufte: "*Distributed Query Processing and Catalogs for Peer-to-Peer Systems*". CIDR 2003, (2003).

[RD00] E. Rahm, H.H. Do : "*Data Cleaning: Problems and Current Approaches*", IEEE Techn. Bulletin on Data Engineering, Dec. 2000

[SK93] Sheth, A. P. and Kashyap, V. 1993. "*So Far (Schematically) yet So Near (Semantically)*", In Proceedings of the IFIP WG 2.6 Database Semantics Conference on interoperable Database Systems (Ds-5) (November 16 - 20, 1992). D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, Eds. IFIP Transactions, vol. A-25. North-Holland Publishing Co., Amsterdam, The Netherlands, 283-312.

[SL90] A.P. Sheth and J.A. Larson : "*Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases*", ACM Computing Surveys, 22(3):183-236, (1990).

[SWV01] H. Stuckenschmidt, H. Wache, U. Visser, G. Schuster: "*Methodologies for Ontology-Based Semantic Translation*", Meeting of the E-Commerce Integration Meta-Framework Group, October 2001, Brussels, (2001).

[UG96] M. Uschold and M. Gruninger: "*Ontologies: Principles, Methods and Applications*", Knowledge Engineering Review, Vol. 11, No. 2, March, 1996, p93-115, (1996).

[WVV01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner: "*Ontology-based Integration of Information - A Survey of Existing Approaches*", In: Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing, Seattle, WA, Vol. pp. 108-117, (2001).

Références des sites Internet

[NET01] http://www.ralphkimball.com/
Site du Kimball Group
Date de dernière consultation : 01/05/2010

[NET02] http://www.supinfo-projects.com/fr/2003/entrepot/3/default.asp
Date de dernière consultation : 01/05/2010

[NET03] http://webdocs.cs.ualberta.ca/~zaiane/courses/cmput690/slides/Chapter2/sld021.htm
Site du Professeur Osmar Zaiane
Cours « Data Warehouse and OLAP »
Date de dernière consultation : 01/05/2010

[NET04] http://vivantech.net/ETLEvaluation.aspx
Site de la société Vivantech
Date de dernière consultation : 01/05/2010

[NET05] http://alistair.cockburn.us
Site de Alistair Cockburn
Date de dernière consultation : 01/05/2010

[NET06] http://blog.todmeansfox.com/2009/05/18/etl-subsystem-31-paralleling-and-pipelining/
Blog de Tod McKenna
Date de dernière consultation : 01/05/2010

[NET07] http://www.journaldunet.com/solutions/0702/070221-panorama-etl/4.shtml
Site du journal du Net
« Les offres d'ETL », 2006
Date de dernière consultation : 01/05/2010

[NET08] http://grim.developpez.com/cours/businessintelligence/concepts/conception-datawarehouse/
Site « developpez.com »
« Conception d'un entrepôt de données (Data Warehouse) »
Date de dernière consultation : 01/05/2010

Glossaire

Agréger : action de réunir des éléments distincts en un tout.

Base de données relationnelle : base de données structurée suivant les principes de l'algèbre relationnelle.

Base XML native : base de données qui s'appuie sur le modèle de données fourni par XML.

Correspondance : rapport logique défini par l'utilisateur entre une ou plusieurs colonnes des bases sources et une colonne de la base cible d'un processus ETL.

Curseur : variable pointant vers un ou plusieurs enregistrement(s) dans une table.

Format pivot : format d'échange de données.

Hétérogénéité sémantique : caractérise les différences dans le sens, l'interprétation ou l'utilisation des mêmes données [KS96].

Index : structure de données améliorant la vitesse des opérations de recherche d'information dans une table d'une base de données.

Métadonnées (d'une base de données) : propriétés d'une base de données (liste des tables, des index, des clés, …).

Parseur : analyseur syntaxique destiné à récupérer les informations contenues dans un document XML.

Schéma global : schéma de la base cible dans une approche matérialisée ; schéma du médiateur dans une approche virtualisée.

Schéma local : schéma d'une base source.

Sérialisation : action de mettre des données dans un flux.

Transformation : opération résultant du traitement d'une correspondance dans XeuTL.

Triggers : événement déclenchant l'exécution d'une opération dans une base de données.

XQuery : langage de requêtes permettant d'extraire des informations d'un document XML ou d'une collection de documents XML.

Annexe 1 : Les offres ETL en 2006 [NET07]

Tableau de synthèse des solutions à suivre			
Editeur	**Solution**	**Tarification**	**Principaux clients**
IBM	Information Server	**Dès 88 000 €** (avec 1 an de maintenance comprise)	NC
Informatica	Power Center	**Dès 50 000 €**	EDF, Société Générale, Danone...
Oracle	Warehouse Builder et Sunopsis Data Conductor	**Dès 25 000 €**	Caroll, Arpège Groupe Caisse d'Epargne...
Microsoft	SQL Server Integration Services	**Dès 24 000 $** (SQL Server 2005 uniquement)	NC
Talend	Open Studio	**Offre Open-Source gratuite** (formation expertise 1000€/jour et support dès 900 €/mois)	Groupe Accor, GMF, Direction Générale de la Comptabilité Publique...
Editeur	**Solution**	**Tarification**	**Principaux clients**

Figure 57 : Synthèse des autres offres du marché - 1 [NET07]

Tableau de synthèse des autres offres du marché		
Editeur	**Solution**	**Commentaire**
Business Objects	Business Object Data Integrator	Data Integrator, intégrée ou non à la suite Business Objects XI, dispose d'une fonctionnalité de centralisation des metadonnées et un module complet de qualité des données (normalisation, correction...). Clients : **Total**...
Cognos	Data Manager	Les metadonnées sont gérées en central et stockées dans un SGBDR tiers. L'extraction des données s'effectue par batch et plusieurs connecteurs ERP sont disponibles en natif (PeopleSoft, SAPet Oracle). **Tarification : dès 72 000 €.** Clients : **Thales, URSSAF, Electrolux**...
Clover.ETL	Clover.ETL	En gérant les processus légers (*threads*) de façon indépendante, Clover.ETL partage la charge de transformation sur différents processeurs et s'adapte ainsi aux multiples configurations matérielles
Information Builder	iWay DataMigrator	DataMigrator s'interfaçe avec les principaux SGBDR du marché en utilisant une palette de connecteurs : ODBC, JDBC, XML, et IBM WebSphere MQ. L'offre est interopérable sous Windows, Unix et iSeries (AS/400) et grands systèmes. **Tarification : dès 40 000 €.** Clients : **BEL, CHEP, SMACL**...
Open Text	Livelink ECM - Data Integration	Open Text propose les briques d'ordonnancement, de gestion centralisée des metadonnées et d'administration des scénarios d'alimentation. **Tarification : dès 40 000 €.** Clients : **Dassault Aviation, Banque Populaire, BMW**...
Pentaho	Pentaho Data Integration	Construite sur la base de frameworks Java, la solution s'intègre à la suite Pentaho Open BI et supporte plus d'une vingtaine d'applications tierces aussi bien Open-Source que propriétaires (SAP, documents Office, fichiers plats...)
SAS	Data Integration Server	L'éditeur propose sa solution en mode intégré ou bien en SaaS et dispose notamment de connecteurs natifs pour les ERP SAP, PeopleSoft, Oracle et Siebel
Sybase	Sybase ETL	Le module Sybase ETL est commercialisé indépendamment de Data Integration Suite, comprenant en outre des briques de réplication et de fédération de données hétérogènes

Figure 58 : Synthèse des autres offres du marché - 2 [NET07]

Tableau comparatif des solutions à suivre			
Editeur / Solution	**Principales briques fonctionnelles**	**Extraction et type d'architecture**	**Connecteurs ERP natifs > 6**
IBM / Information Server	Ordonnancement des tâches, gestion centralisée des metadonnées (*Metadata Server*), gestion des glossaires métier (*Business Glossary*), administration des scénarios d'alimentation et qualité des données (*Datastage*)...	Extraction batch temps réel et/ou via MOM. Architecture : Hub&Spoke	SAP, Oracle, PeopleSoft et Siebel
Informatica / Power Center	Ordonnancement des tâches, gestion centralisée des metadonnées, administration des scénarios d'alimentation et qualité des données, profiling et accès aux données non structurées...	Extraction batch temps réel et/ou via MOM. Architecture : Hub&Spoke	✔ (SAP, Oracle, Siebel, PeopleSoft...)
Oracle / Warehouse Builder et Sunopsis Data Conductor	Ordonnancement des tâches, gestion centralisée des metadonnées, qualité des données...	NC	Oracle, PeopleSoft, Siebel...
Microsoft / SQL Server Integration Services	Ordonnancement des tâches, gestion centralisée des metadonnées, qualité des données...	Extraction batch temps réel et/ou via MOM	NC
Talend / Open Studio	Ordonnancement des tâches (*scheduler*), gestion centralisée des metadonnées (*MetadataManager*), gestion des glossaires métier (*Business Modeler*), administration des scénarios d'alimentation (*rAMC*) et qualité des données...	Extraction batch temps réel. Architecture : Hub&Spoke (Network Centric via MOM)	OpenBravo et SAP (en cours de développement)

Figure 59 : Synthèse des autres offres du marché - 3 [NET07]

Annexe 2 : Extrait de l'étude réalisée sur l'intégration de données dans le cadre du projet européen NOESIS[31]

J'ai réalisé cette étude dans le cadre du projet européen NOESIS: *Platform for a wide scale integration and visual representation of medical intelligence*. Cette étude a été intégrée à un état de l'art sur l'intégration de données. Cet état de l'art constituait un délivrable du projet.

Many different systems have been developed these latter years. Here are some examples of already developed data integration systems implementing the mediator approach.

1.1 A detailed example: MOMIS

1.1.1 Presentation

The MOMIS[32] (Mediator envirOnment for Multiple Information Sources) is a framework performing information extraction and integration from both structured and semi-structured data sources.

This document mainly focuses on the ontology building in MOMIS.

1.1.2 Architecture

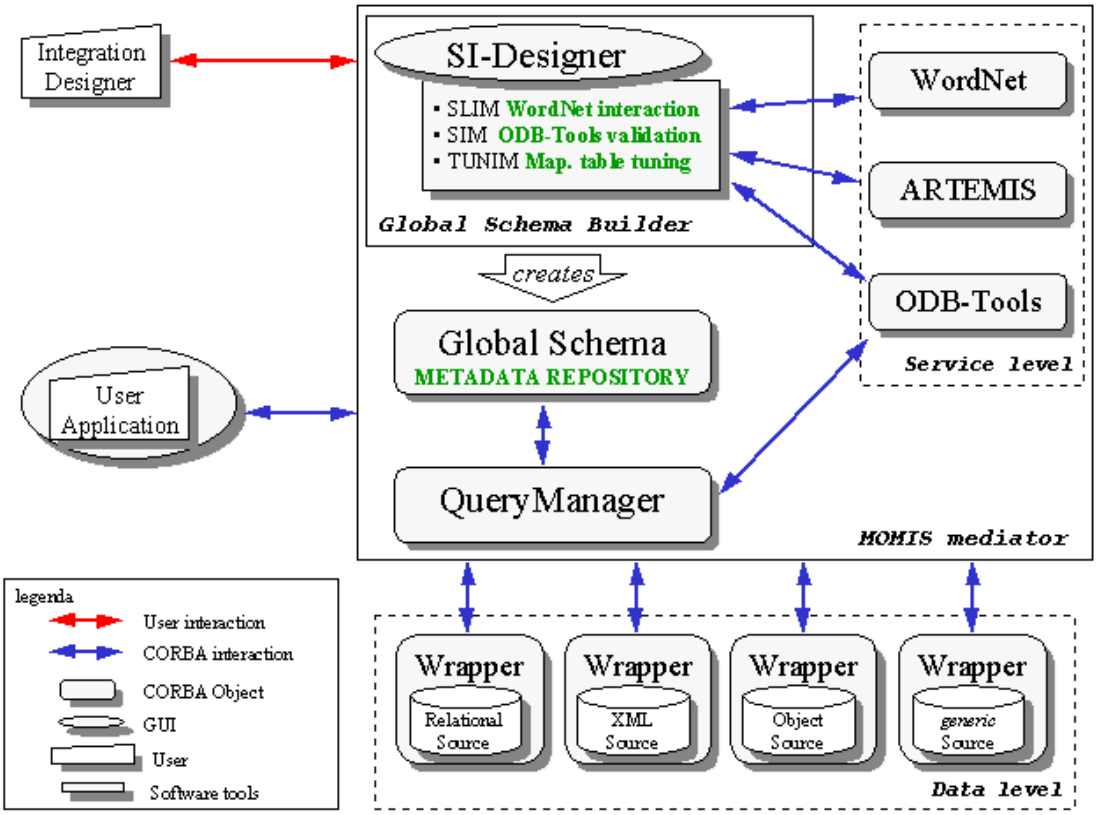
As presented in [BBC00], MOMIS follows a "semantic approach" to information integration based on the conceptual schema, or metadata, of the information sources, and on the I3[33] architecture.

It is composed of the following elements described in [BBG01] that communicate using the CORBA standard (see figure 10):

- a common data model, ODM-I3, which is defined according to the ODL-I3 language, to describe source schemas for integration purposes.
- wrappers, placed over each sources:
  - translate metadata descriptions of the sources into the common ODL-I3 representation
  - translate (reformulate) a global query expressed in the OQLI3 query language into queries expressed in the sources languages
  - export query result data set
- a mediator which is composed of two modules:
  - the SI-Designer module processes and integrates ODL-I3 descriptions received from wrappers to derive the integrated representation of the information sources
  - the Query Manager (QM) module performs query processing and optimisation. The QM generates the OQL-I3 queries to be sent to wrappers starting from each query posed by the user on the global schema. QM automatically generates the translation

[31] http://www.noesis-eu.org
[32] http://dbgroup.unimo.it/Momis/
[33] http://isse.gmu.edu/I3_Arch/index.html

of the query into a corresponding set of sub-queries for the sources and synthesizes a unified global answer for the user.



Note: please refer to the MOMIS Prototype Description[34] presented on the MOMIS website for a more complete description of the different modules. See also the examples included in this chapter for a cleaver view of the different modules.

Fig 10: MOMIS architecture

1.1.3 MOMIS originality

In order to face common problems due to the heterogeneity of the data sources, MOMIS provides the capability of explicitly introducing many kinds of knowledge for integration, such as integrity constraints, intra- and inter-sources intensional and extensional relationships, and designer supplied domain knowledge (see [BBG01]).
A *Common Thesaurus*, which has the role of a shared ontology of the source, is built in a semi-automatic way. The *Common Thesaurus* is a set of intra and inter-schema intensional and extensional relationships, describing inter-schema knowledge about classes and attributes of sources schemas. It provides a reference on which to base the identification of classes' candidate to integration and subsequent derivation of their global representation.
The idea of information integration in MOMIS mainly lies in:

- the creation of an integrated view of all sources (the Global Virtual View) in an automated way (as much as possible). This global virtual view is expressed by using XML standard for obvious interoperability reasons.

[34] http://dbgroup.unimo.it/Momis/prototipoPub/

- the possibility to perform revision and validation of the various kinds of knowledge used for the integration. For that purpose, MOMIS combines reasoning capabilities of Description Logics with affinity-based clustering techniques, by exploiting a common ontology for the sources constructed using lexical knowledge from WorldNet and validated integration knowledge.

1.1.4 Integration Process

The intelligent schema integration process in MOMIS is composed of four phases ([BBG01]):

- generation of a common thesaurus: the common thesaurus is a set of terminological intensional and extensional relationships, describing intra and inter-schema knowledge about classes and attributes of sources schemas. Inter-schema knowledge is expressed in form of terminological and extensional relationships between classes and/or attribute names.
- Affinity analysis of classes: relationships in the common thesaurus are used to evaluate the level of affinity between classes intra- and inter-sources. The concept of affinity is introduced to formalize the kind of relationship that can occur between classes from the integration point of view. The affinity of two classes is established by means of affinity coefficients based on class names, class structures, and relationships in Common Thesaurus.
- Clustering classes: classes with affinity in different sources are grouped together in clusters using hierarchical clustering techniques. The goal is to identify the classes that have to be integrated since describing the same or semantically related information.
- Generation of the mediated schema: unification of affinity clusters leads to the construction of the predicated schema. A class is defined for each cluster, which is representative of all cluster's classes and is characterised by the union of their attributes. The global schema for the analyzed source is composed of all classes derived from clusters, and is the basis for posing queries against the sources.

1.1.5 SI-Designer architecture

The SI-Designer handles the main part of the integration process (see [BBC01]).
It is composed of two modules:

- SIM (Source Integrator Module): extracts intra-schema relationships starting from a relational, object and semi-structured source. Moreover this module performs the "semantic validation" of relationships and infers new relationships by exploiting ODB-Tools capabilities.
- SLIM (Sources Lexical Integrator Module): extracts inter-schema relationships between names and attributes of ODL-I3 classes of different sources, exploiting the WordNet lexical system.

SI-Designer provides a graphical interface to interact with SIM, SLIM and ARTEMIS modules showing the extracted relationships and helping in the Common Thesaurus construction.
Once the Common Thesaurus has been built, SI-Designer uses the ARTEMIS module to evaluate a disjoint set of clusters (structural similar classes). Each cluster has a corresponding global class (a view over all similar classes belonging to the cluster) characterized by a set of global attributes and a mapping-table. SI-Designer automatically generates a set of global

attributes for each global class and a mapping table which maps each global attribute into the local attributes of the classes in the cluster.
Then, a semi-automatic interaction with the designer starts: the designer may revise the set of global attributes and the mapping table, to assign a name to each global class, so as to achieve a global schema.

1.1.6 Example

Here is an example extracted from [BCV99].
Two departments of a hospital, the cardiology department and the intensive care department, need to share information concerning their patients:

- the *Cardiology* department (CD) contains semi-structured objects about patients with ischemic heart diseases, hypertension, and about physicians and nurses who have access in the line of the duty to information concerning patient's health. Figure 11 illustrates a portion of the data.

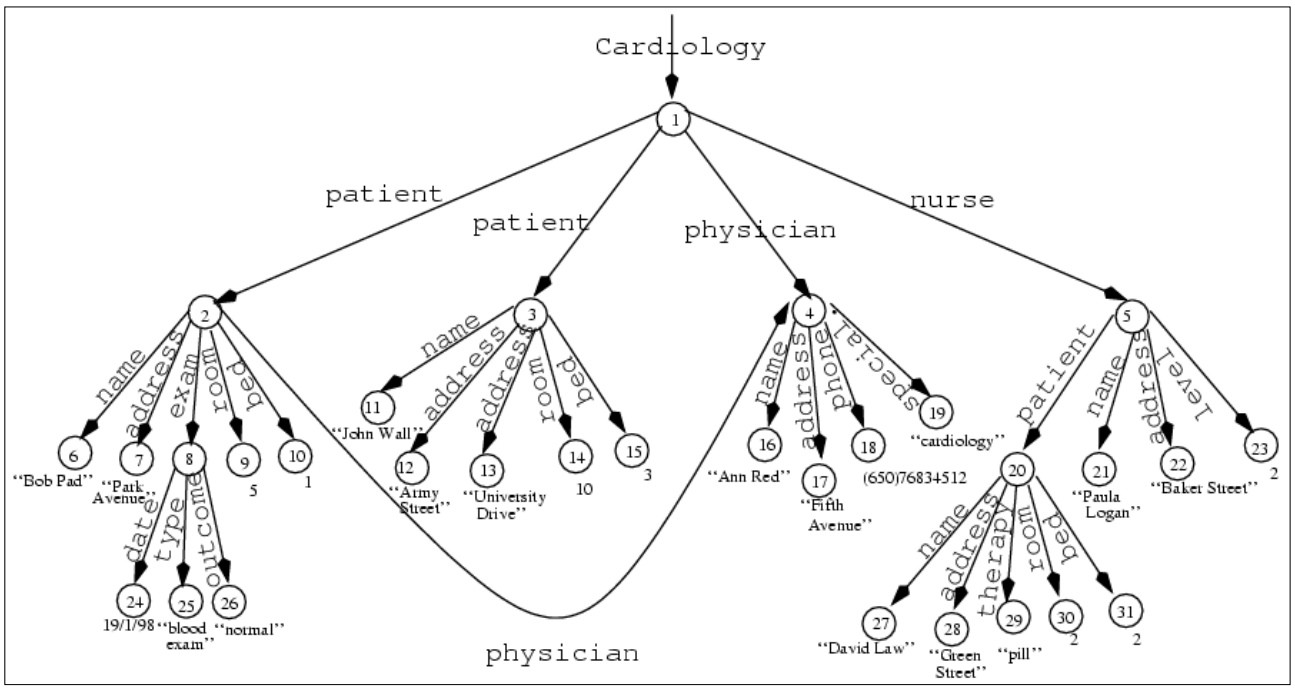


Figure 11: Cardiology department (CD)

- the *Intensive Care* department (ID) is a relational database containing information similar to information of the cardiology department: it stores data both on patients with diagnoses of trauma, myocardial infraction and on medical staff. Fig 12 illustrates the four relations forming this database. Dis_Patient contains information about discharged patients. It is a subset of Patient instance.

```
Patient (code, first_name, last_name, address, test, doctor_id)
Doctor (id, first_name, last_name, phone, address, availability, position)
Test (number, type, date, laboratory, result)
Dis_Patient (code, date, note)

Note: Dis_Patient is a subset of Patient instance
```

Fig 12: the intensive care database (ID)

For integration and query, we consider schema descriptions of the sources. For structured data sources, the schema is already available and is used. To represent semi-structured data at the intensional level, we associate an object pattern with each set of objects having the same label in the source graph. Object patterns for all the objects in our semi-structured source (the CD source) are shown in Fig 13.

```
Patient-pattern (Patient, {name, address, exam*, room, bed, therapy*,physician*})
Physician-pattern (Physician, {name, address, phone, specialisation})
Nurse-pattern (Nurse,{name, address, level, patient})
Exam-pattern (Exam, {date, type, outcome})
```

Fig 13: Object patterns for the CD source

Schemas of structured data sources (e.g., ID source) and object patterns (e.g., the CD source) are translated into ODL-I3 language. Each object pattern and each relation is translated into an ODL-I3 class.

The ODL-I3 representation of the CD.Patient object pattern is presented in figure 14.

```
interface Patient
( source semistructured Cardiology_Department )
{
attribute string            name;
attribute string  address;
attribute set<Exam>     exam*;
attribute integer           room;
attribute integer           bed;
attribute string  therapy*;
attribute set<physician> physician*;
};

Note: '*' denotes optionality
```

Fig 14: ODL representation of the CD.patient object pattern

1.1.7 Generation of a Common Thesaurus

To provide a shared ontology for the sources, a dictionary of terminological relationships describing common knowledge about ODL-I3 classes and attributes of sources schemas is constructed, called Common Thesaurus. The following kinds of relationships are represented in the common Thesaurus:
SYN (Synonym-of) is defined between two terms that are considered synonyms.

BT (Broader Terms), or hypernymy, is defined between two terms such as one has a more general meaning than the other. BT is not symmetric. The opposite of BT is NT (Narrower Terms).
RT (Related Terms), or holonymy, is defined between two terms that are generally used together in the same context.

Discovering terminological relationships from source schema is a semi-automatic activity in MOMIS. This activity is assisted by ODB-Tools and proceeds in the following steps.

Automated extraction of relationships

By exploiting ODB-Tools capabilities and semantically rich schema descriptions, an initial set of BT, NT, and RT can be automatically extracted. In particular by translating ODL-I3 into OLCD descriptions, ODB-Tools extract BT/NT relationships among classes discretely from generalization hierarchies, respectively. Other RT relationships are extracted from the specification of foreign keys in relational source schemas.

```
<ID.Patient RT ID.Doctor>
<CD.Nurse RT CD.Patient>
<CD.Patient RT CD.Physician>
<CD.Patient RT CD.Exam>
<ID.Patient BT ID.Dis_Patient>
<ID.Patient RT ID.Test>
```

Example 1.1: Set of terminological relationships automatically extracted by ODB-Tools

Another set of relationships can be automatically extracted exploiting the WorldNet lexical system. In this case, synonyms, hypernyms/hyponyms, and related terms can be automatically proposed to the designer, by selecting them according to relationships predefined in the lexical system.

```
<ID.Doctor BT CD.Physician>
<ID.Test SYN CD.Exam>
<CD.Patient.name BT ID.Patient.first_name>
<CD.Patient.name BT ID.Patient.last_name>
```

Example 1.2: Set of relationships derived from WordNet

Integration/Revision of new relationships

In addition to the terminological relationships automatically extracted, other relationships can be supplied directly by the designer, interacting with the tool, to capture specific domain knowledge on the sources schemas (e.g., new synonyms).

```
<ID.Doctor BT CD.Nurse> <ID.Patient SYN CD.Patient>
<CD.Patient.physician BT ID.Patient.doctor_id>
<CD.Nurse.level SYN ID.Doctor.position>
<CD.Exam.outcome SYN ID.Test.result>
```

Example 2: Terminological relationships supplied by the designer

Validation of relationships

In this step, ODB-Tools are employed to validate terminological relationships defined for attributes in the Thesaurus, by exploiting the virtual schema.

```
<CD.Patient.physician BT ID.Patient.doctor_id>      [0]
<CD.Patient.name BT ID.Patient.first_name>          [1]
<CD.Patient.name BT ID.Patient.last_name>           [1]
<CD.Nurse.level SYN ID.Doctor.position>             [0]
<CD.Exam.outcome SYN ID.Test.result>                [1]
```

Example 3: Output of the validation phase ([1]: valid relationship, [0]: invalid relationship)

Inference of new relationships

In this step, inference capabilities of ODB-Tools are exploited. A new set of terminological relationships is inferred by ODB-Tools, by exploiting the "virtual schema" defined in the revision/integration step and by deriving new generalization and aggregation relationships.

```
<ID.Patient RT CD.Physician>
<ID.Patient RT CD.Exam>
<CD.Patient RT ID.Doctor>
```

```
<CD.Patient RT ID.Test>
<ID.Dis_Patient RT ID.Doctor>
<ID.Dis_Patient RT ID.Test>
<ID.Dis_Patient RT ID.Doctor>
<ID.Dis_Patient RT CD.Physician>
```

Example 4: Inferred terminological relationships

Fig 15 represents the overall process that results in the Common Thesaurus generation.

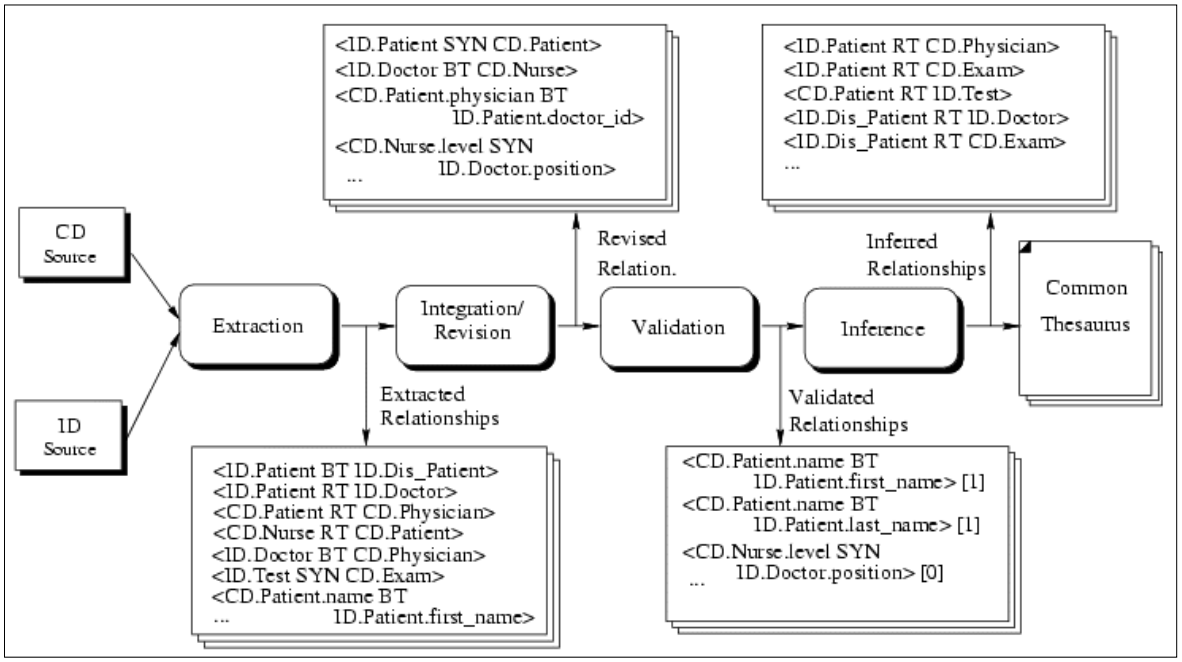


Figure 15 : the Common Thesaurus generation

Fig 16 illustrates a graphical representation of the Common Thesaurus for CD and ID departments. Solid lines represent explicit relationships (i.e., extracted/supplied), dashed lines represent inferred relationships, and superscripts indicate their kind.

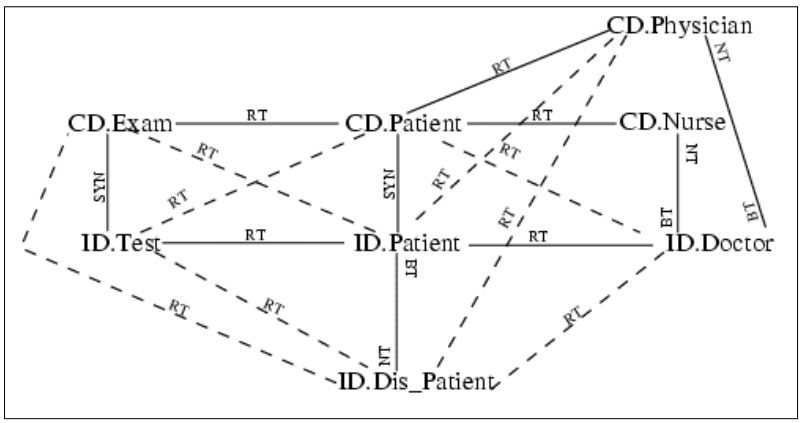


Figure 16 : The Common Thesaurus

Building the mediator integrated view

ODL-I3 classes having a semantic relationship in different sources are identified. For this purpose, *affinity coefficients* are evaluated for all possible pairs of ODL-I3 classes, based on the (valid) terminological relationships in the Common Thesaurus. Affinity coefficients determine the degree of semantic relationship of two classes based on their names (Name affinity coefficient) and their attributes (Structural Affinity coefficient). A comprehensive value of affinity, called *Global Affinity* coefficient, is finally determined as the linear combination of the Name and Structural Affinity coefficients. Global affinity coefficients are used by a hierarchical clustering algorithm, to classify ODL-I3 classes according to their degree of affinity. The output of the clustering procedure is an affinity tree, where ODL-I3 classes are the leaves and intermediate nodes have an associated value, holding for the classes in the corresponding cluster. The affinity-based evaluation and clustering procedures are performed with the help of the ARTEMIS tool environment. An affinity tree example is shown in figure 17.

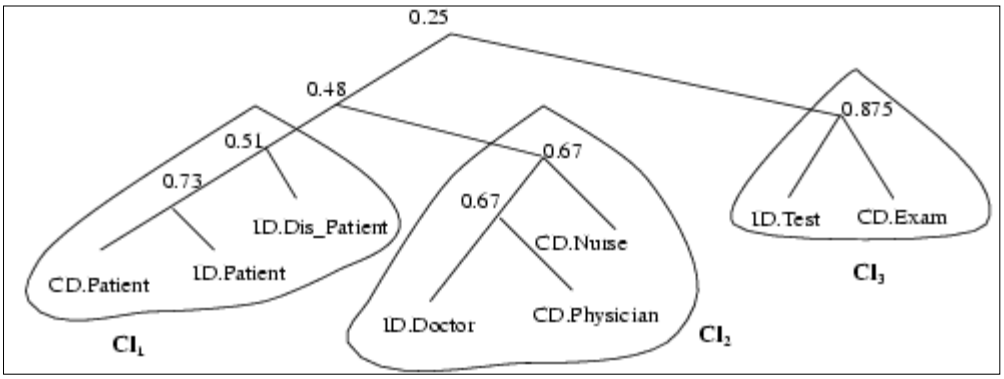


Figure 17 : The affinity tree

Clusters for integration are interactively selected from the affinity tree using a threshold based mechanism. For each selected cluster in the tree, a global class gci representative of the classes obtained in the cluster (i.e., a class providing the unified view of all the classes of the cluster) is defined. The generation of gci is interactive with the designer. Let Cli be a selected cluster in the affinity tree. First, the global schema builder component of MOMIS associates to the Gci a set of global attributes, corresponding to the union of the attributes of the classes belonging to Cli, where the attributes with a valid terminological relationship are unified into a unique global attribute in Gci. The attribute unification process is performed automatically for what concerns names according to the following rules: for attributes that have a SYN relationship, only one term is selected as the name for the corresponding global attribute in Gci; for attributes that have a BT/NT relationship, a name which is a broader term for all them is selected and assigned to the corresponding global attribute in Gci.

To complete global class definition, information on attribute mappings and default values is provided by the designer in the form of mapping rules. An example of ODL-I3 specification for the global class *Hospital_Patient* is shown in figure 18:

```
interface Hospital_Patient
{ attribute name
        mapping_rule (ID.Patient.first_name and
                        ID.Patient.last_name),
                        CD.Patient.name;
  attribute physician
        mapping_rule CD.Patient.physician,
                        ID.Patient.doctor_id
  attribute dept
        mapping_rule CD.Patient = 'Cardiology',
                        ID.Patient = 'Intensive Care',
                        ID.Dis_Patient = 'Intensive Care'
}
```

figure 18: the global class hospital

A mapping rule is defined for each global attribute A and specifies:

- information on how to map A with the corresponding class attributes of the associated cluster
- default/null values defined for A based on values of attributes of cluster classes.

For example, the mapping rule defined for the global attribute *name* in the global class Hospital_Patient above, specifies which attributes have to be considered in each class of the cluster Cl1: an *and* correspondence is defined for *name*, stating that the concatenation of the attributes *first_name* and *last_name* of *ID.Patient* have to be considered. The mapping rule defined for the global attribute *dept* specifies the value of this attribute for the instances of classes *CD.Patient*, *ID.Patient* and *ID.Dis_Patient*. The global schema of the mediator is composed of the global classes defined for all the clusters of the affinity tree.
Integrity constraint rules can also be specified for global classes of the mediator global schema, to express semantic relationships holding among the different sources.

1.1.8 SI-Designer Tool

SI-Designer is a framework that represents a unified solution for the overall integration process (see [BBG01]).
SI-Designer provides a graphical interface to reach the Global Virtual View, relating to each integration step a specific interaction with a software module. All the modules involved are available as CORBA object and interact using established idl interfaces. In particular the SI-Designer performs these steps:

- source acquisition: in this phase the user can select the sources to be integrated. A wrapper performs the translation from the source description model into ODL-I3 description model. This step involves SAM module.
- intensional relationships definition: in this phase, new relationships, schema derived, by interacting with SIM module and ODB-Tool system, lexicon derived, by interacting with the WordNet lexical database, and designer supplied are added to the Common Thesurus (see fig 19 for an example).

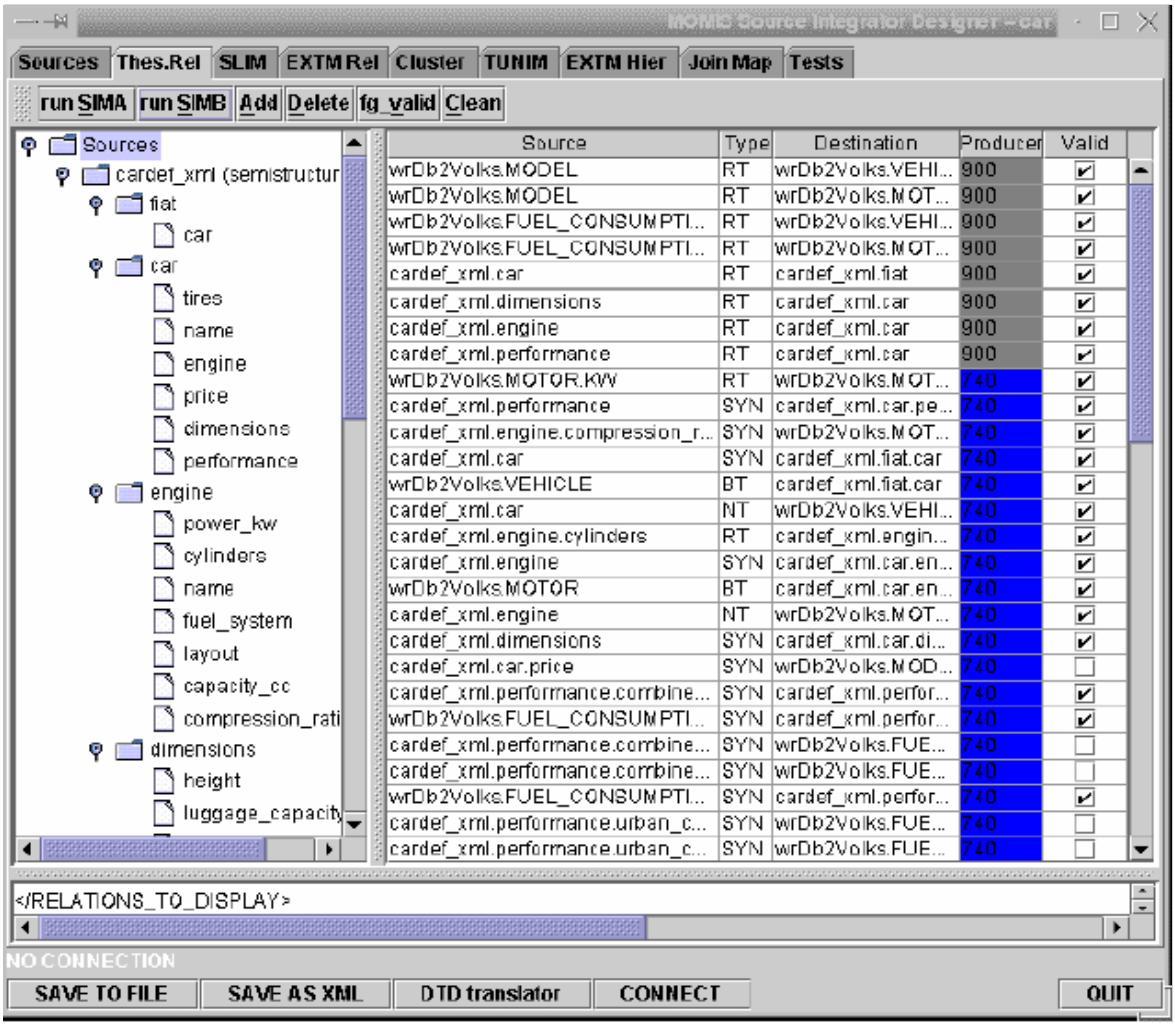


Figure 19: intensional relationships definition

- extensional relationships definitions: Extensional relationships are defined by the interaction with the integration designer. These relationships are exploited to detect extensionally overlapping classes.
- Clustering: in this phase, based on the knowledge carried in the Common Thesaurus, by exploiting ARTEMIS module, global classes are created. Fig 20 represents an example of clusters.

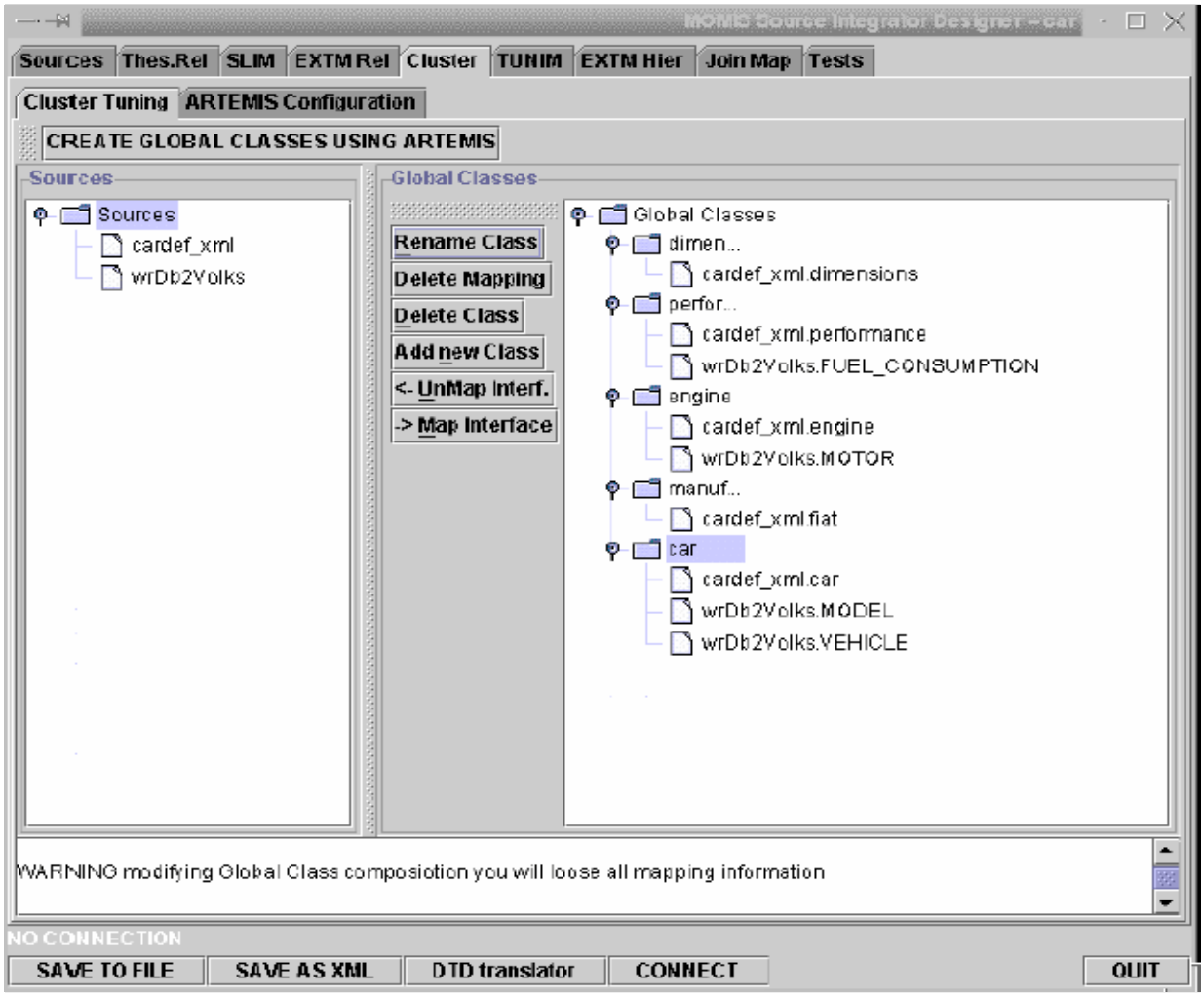


Fig 20: the clustering definition

- mapping table tuning: for each global class generated in the previous phase, the user can modify the Global Virtual View proposed automatically from the system. Fig 21 represents the mapping table tuning.

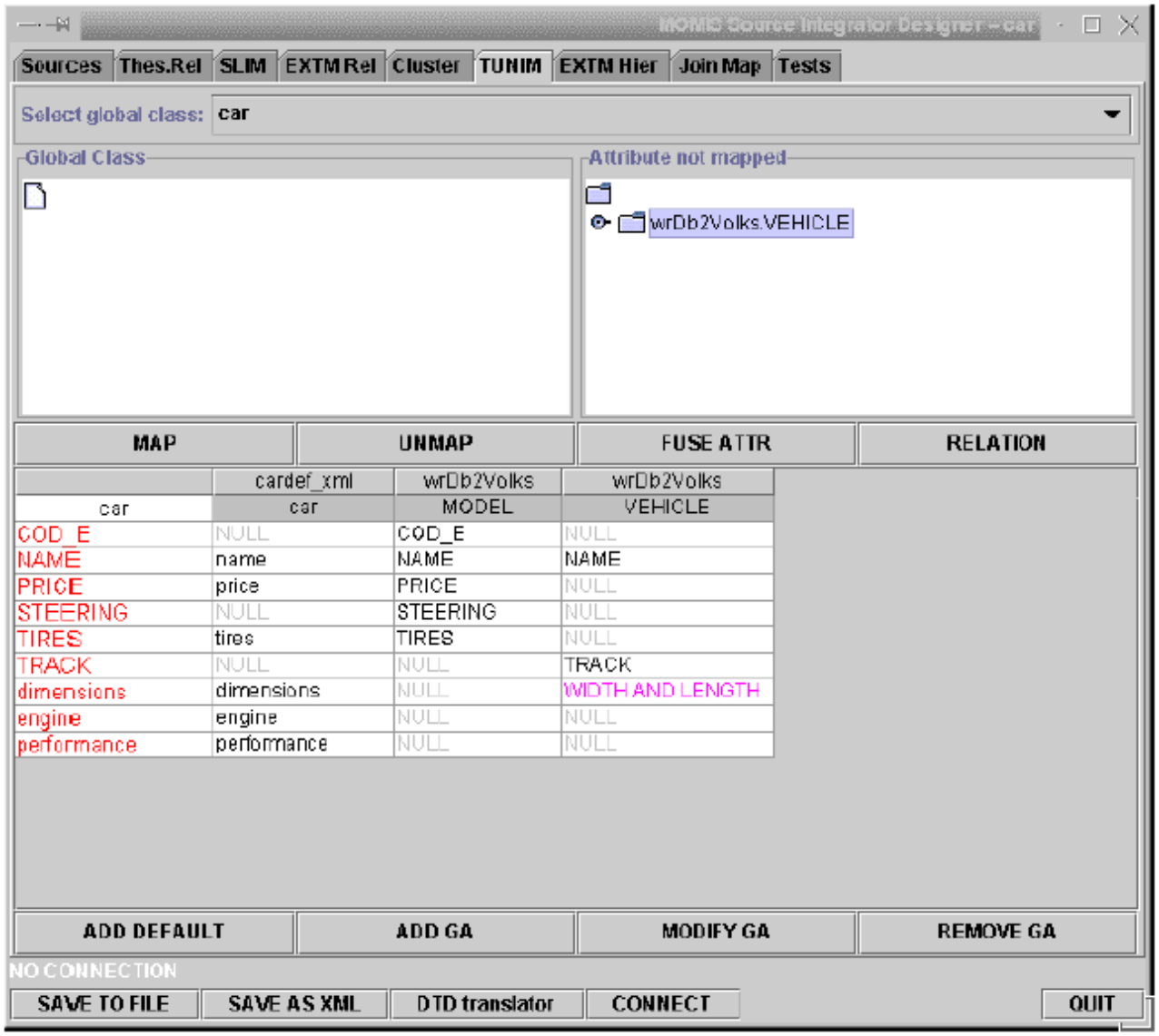


Fig 21: the mapping tuning table

The final step of the integration process provides the export of the Global Virtual View into a XML DTD, by adding the appropriate XML TAGs to represent the mapping table relationships. The use of XML in the definition of the Global Virtual View allows using MOMIS infrastructure with other open integration information system by the interchange of XML data files. In addition, the Common Thesaurus is translated into XML file, so that MOMIS may provides a shared ontology that can be used by different semantic ontology languages.

1.2 Other data integration projects in brief

1.2.1 Picsel

1.2.1.1 General Presentation

Picsel[35] ("Production d'Interfaces à base de Connaissances pour des Services En Ligne" or "Production of Interfaces based on Knowledge for On Line Services") is a data integration project developed by the LRI (Laboratoire de Recherche en Informatique - Université Paris XI) and France Télécom R&D.
The information presented in this document mainly comes from [RBC02].
The goal of Picsel is to build information integration agents related to application domains for which there exists many information sources containing closely related data.
This project is already in use in the tourism domain: it is the core of the research engine of Degriftour Travel Agency[36].

1.2.1.2 Technical presentation

Picsel is implemented in Java. It follows the LAV approach. It is based on the CARIN-ALN language to represent both the domain of application and the contents of information sources relevant to that domain.
The aim of Picsel is to guarantee the decidability of query answering and a correct and complete reasoning.

1.2.1.3 Modelisation

The first modelling step of the Picsel prototype was to build the ontology of the domain: here the tourism domain. This was achieved manually by collecting information found in tourism catalogs. Then this information has been analysed in order to bring out concepts (Ontoclass, a visual classification tool has been used to facilitate the task). Then the operational mediated schema has been modeled with the CARIN language.
The mediated schema is composed of atomic concepts and complex relations defined by rules and concept expressions. The obtained ontology, which is the support of both the user interface and the query engine, is a hierarchy of 200 concepts and 300 roles.
The second step was to build the data sources schemas (the abstract views). Each source was represented as a set of views described by a set of rules indicating the kind of data found in the source (there are as many source relations as domain relations whose instances can be found in the considered source: LAV approach), and a set of constraints on the instances of the sources relations.

1.2.1.4 Query processing

The query processing of Picsel is out of the scope of this document.

35 http://www.lri.fr/~picsel/
36 Http://www.lastminute.com

1.2.1.5 Human computer interface

The language used for querying (CARIN) is not very intuitive. Moreover the ontology built here is very rich and detailed. Therefore it is quite difficult for an unaccustomed user to query the system.
That is why the idea of a guided query formulation has been developed. It allows the user to find the relevant terms easily. It also hides the language formalism.
This is achieved by a graphical interface proposing predefined general queries, automatic query generation and a guide for refining terms.

1.2.1.6 Conclusion

Thanks to the expressive power of CARIN-ALN the mediated and sources schemas are very rich. It has been shown that this kind of formalism is well appropriate for building specialized information server over a reasonable number of data sources. It also allows complex querying although query processing is proportional in complexity. The LAV approach allows to add new sources easily (the mediated schema does not require any changing).
Nevertheless this kind of approach (based on a single mediated schema) is not flexible enough for dealing with the huge quantity of information available through the web.

1.2.2 Xyleme

1.2.2.1 Presentation

The Xyleme project was first developed by the INRIA[37] Research team. As stated in [Abi01], the project was completed at the end of 2000. A prototype has been implemented. This prototype is now being turned into a product by a start-up company also called Xyleme[38].
The goal of Xyleme is to provide a very wide-area integration of XML sources that could deal with the huge quantity of information available through the web. The Xyleme system is inspired by web search engines. Xyleme can be seen as a dynamic (interested in data evolution/changes) warehouse focused on XML, capable of storing huge quantities of data. It is neither a research engine (only index) nor a mediator (only virtual data).

1.2.2.2 The challenges

As presented in [Abi01] the challenges for Xyleme are:

- Data acquisition and Maintenance: discover data of interest and maintain it up to date
- Repository: store this data and index it so that it can be processed efficiently
- Query processing: support efficiently an SQL-style query language
- Semantic Integration: understand DTD and tags, partition the Web into semantic domains, provide a simple view of each domain
- Change control: monitor the web and offer services such as Query subscriptions
- Size of data: billions of pages
- Size of index: terabytes
- Number of customers: thousands of simultaneous queries and millions of subscriptions

37http://www.inria.fr/
38 http://www.xyleme.com/.

1.2.2.3 Architecture

Figure 22 presents the functional architecture of Xyleme.

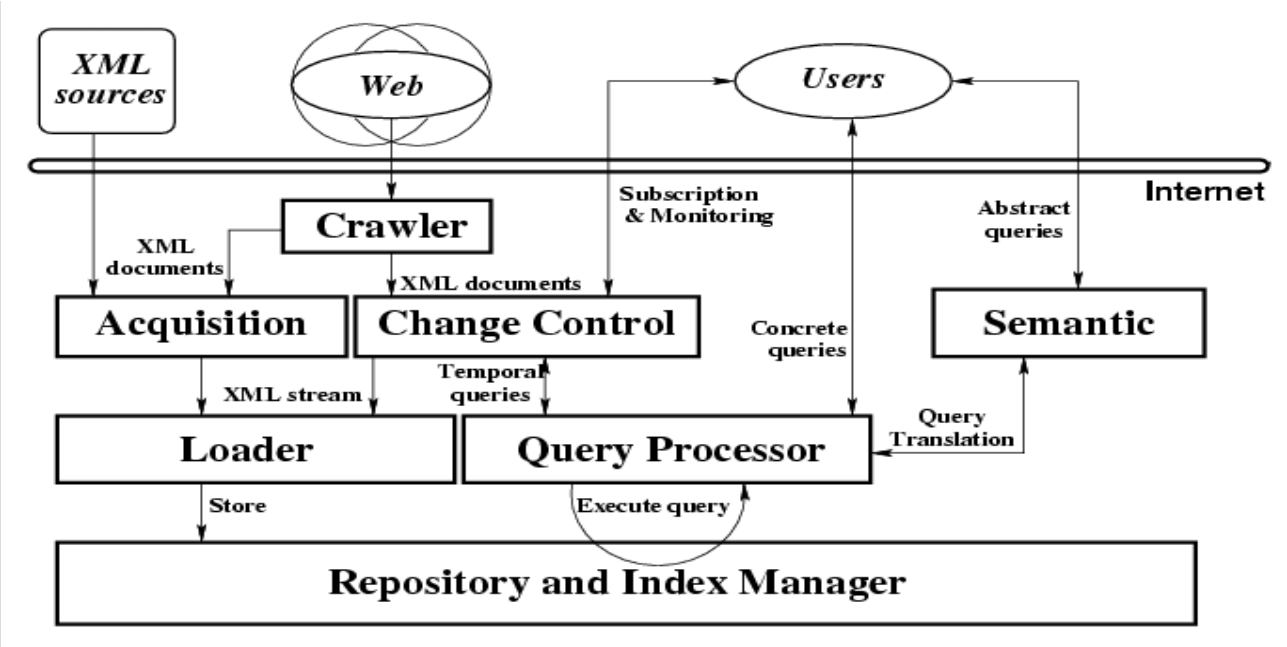


Figure 22: Xyleme functional architecture

Here is a brief description of the different modules:

- Acquisition & Crawler : in charge of inspecting the web and collecting the XML documents
- Change control : specific functionalities such as management of document changes, versions, temporal queries
- Loader : load XML documents in the repository
- Repository and Index Manager : storage and indexing of XML documents (data sources)
- Query processor : allow to query the repository as a database (transformation of queries between the semantic module and the repository)
- Semantic : provides a mediated schema

1.2.2.4 The modules

1.2.2.4.1 The sources

For Xyleme, an XML document is a data sources and can be seen as a data tree. Therefore the DTD of the document represents the data source schema and can be seen as a tree type. A data tree is an instance of a tree type. Figure 23 is an example of a data tree and figure 24 an example of a tree type.

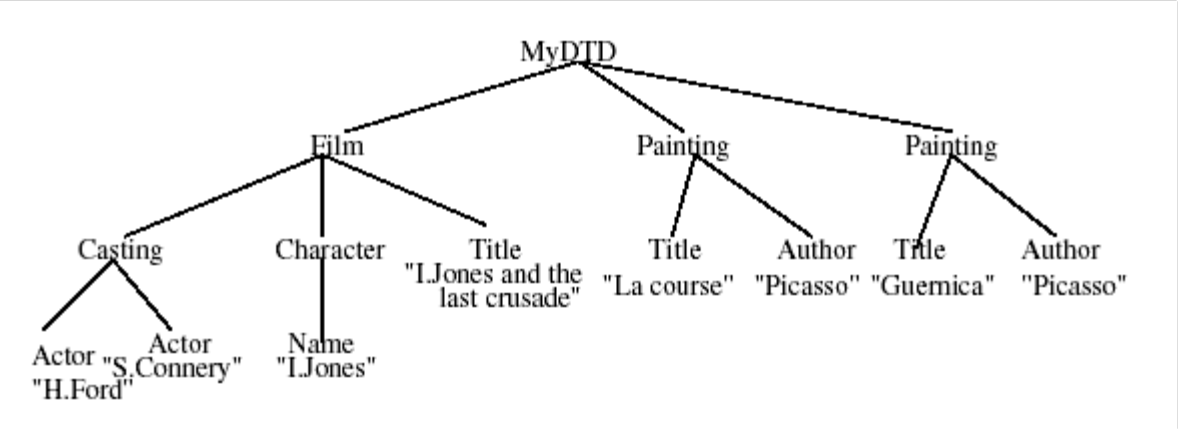


Figure 23: a data tree

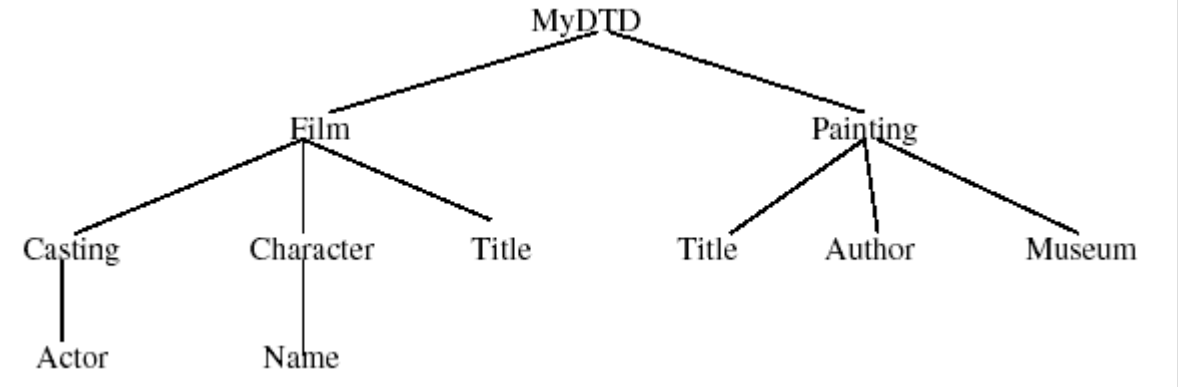


Figure 24: a tree type

1.2.2.4.2 The semantic module

The semantic module is in charge of providing a homogeneous mediated schema (global schema) over the data trees. This mediated schema is actually an abstract tree type describing domains (e.g., culture, tourism) like an abstract merger of the DTDs.
Figure 25 presents a fragment of an abstract tree type concerning the culture domain.

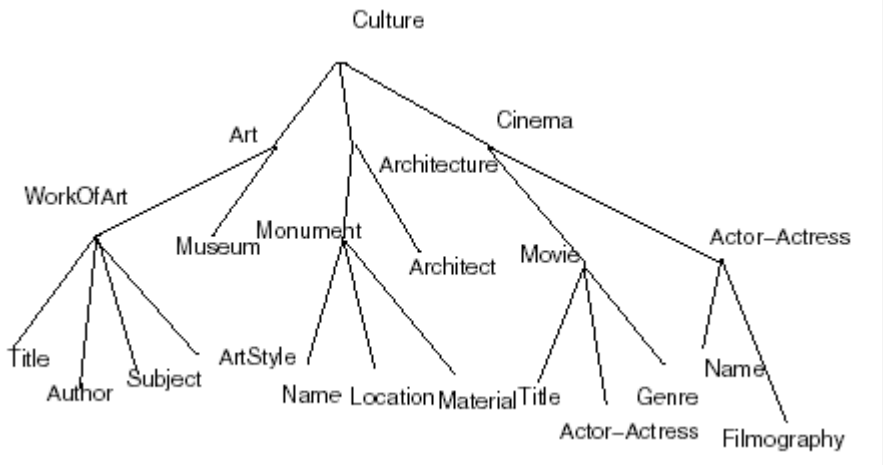


Figure 25: Abstract tree type fragment.

This kind of representation is simple to understand for the user.
The connections between the mediated schema and the tree types are made by mapping relations between paths of the abstract tree type and paths of the tree types modeling the DTDs in the repository (GAV and LAV approach). Here is an example illustrating a mapping relation in the system presented in figures 23, 24 and 25 : Culture/Cinema/Movie ↔ MyDTD/Film.
This kind of mapping relation is generated in a semi-automated manner.

1.2.2.4.3 User queries

The language used for querying the system is a tree based query language (a subset of Xquery[39]).
Figure 26 presents modeled examples of user queries:

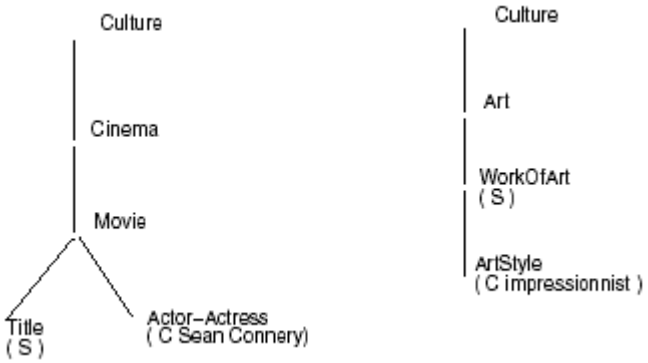


figure 26: user queries examples

S represents the nodes to be included in the result, C the nodes constrained with conditions.

1.2.2.4.4 Query processing

The query processing is performed in 2 steps:
- the abstract query is translated into a concrete query using the mapping relation
- the tree query is evaluated onto the database

Figures 27, 28 and 29 illustrate the query processing.

```
Movie <-> Film          Movie/ActorActress <-> Film/Casting
Movie/ActorActress/Name <-> Film/Character
Movie/ActorActress/Name <-> Film/Casting/Actor
Movie/ActorActress/Role <-> Film/Character/Name
```

Figure 27: Mapping relations between abstract and concrete tree types

39http://www.w3.org/TR/xquery/

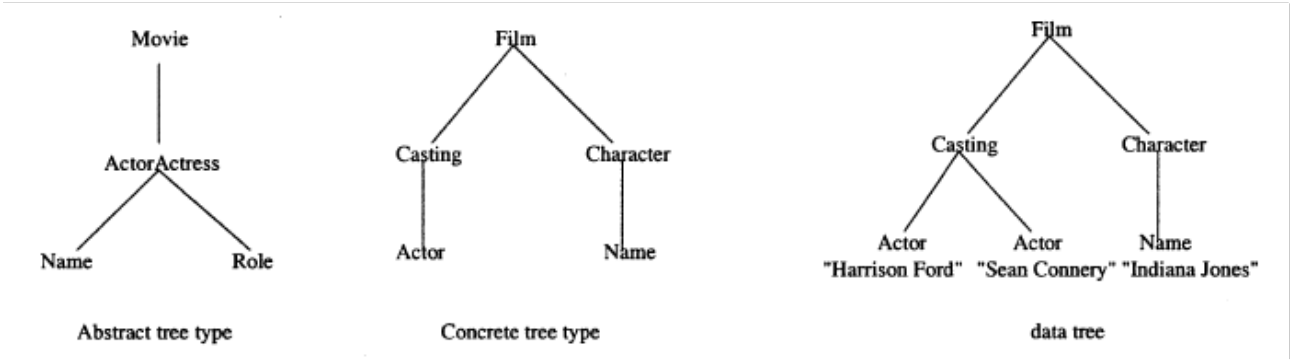


Figure 28: Tree types examples

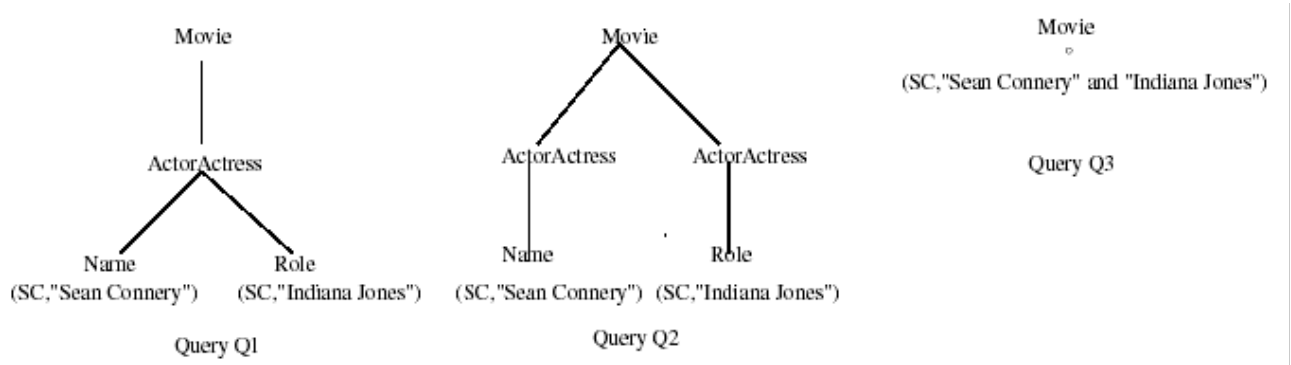


Figure 29: Tree queries examples

The necessary nodes (root, selected, conditional and join nodes) are ordered with <.
The goal is to convert abstract paths into concrete paths.
The process is the following one:

- the abstract paths leading to the necessary nodes are listed
- the nodes in the abstract path leading to the necessary nodes are ordered
- the mapping between the abstract paths and the concrete paths is achieved

1.2.2.4.5 Data acquisition

Another idea of Xyleme is the possibility to acquire data automatically. In order to perform this task, automated web agents visit the web and retrieve pages to discover new URLs. Each crawler deals with millions of pages per days, and several crawlers may be used simultaneously.

1.2.2.5 Conclusion

Xyleme aims at integrating a huge number of sources covering very broad topics. The simplicity of the global schema permits semi-automation of the integrating process. The language used for knowledge representation is not very expressive, implying simplicity of the query processing.
Nevertheless, Xyleme is based on a single global schema which is not enough for dealing with the huge quantity of information made available by the web.

MEMOIRE D'INGENIEUR C.N.A.M. en INFORMATIQUE

XeuTL, un outil pour l'intégration de données

Cédric Gueydan **Grenoble, le 30 juin 2010**

Résumé

Depuis que l'utilisation des bases de données s'est démocratisée, les entreprises traitent des volumes de données de plus en plus importants. Ces données sont souvent réparties dans différents systèmes hétérogènes. L'intégration de ces données est nécessaire afin de les exploiter efficacement. La principale problématique de l'intégration de données est l'hétérogénéité sémantique. Il existe deux approches pour traiter l'intégration et résoudre l'hétérogénéité sémantique :

- l'approche virtuelle : l'utilisateur interroge un médiateur qui manipule et traduit sa requête afin d'interroger différentes sources de données ; les données sont uniquement stockées dans les bases sources ;
- l'approche matérialisée : l'utilisateur interroge une base cible qui contient une copie des données intégrées des bases sources.

Ce travail présente la réalisation de XeuTL, un outil de type ETL indispensable dans l'approche matérialisée. XeuTL extrait les données des bases sources et les transforme avant de les charger dans une base cible. L'utilisateur définit des correspondances entre les schémas physiques des bases sources et celui de la base cible, résolvant ainsi l'hétérogénéité sémantique des données.

Mots-clés : Intégration, base de données, hétérogénéité sémantique, ETL, extraction, transformation, chargement, XML

Abstract

Since Business companies have adopted databases, they have to deal with huge volumes of data. This data is often spread across heterogeneous systems. Integrating this data is necessary in order to exploit it in an efficient way. Semantic heterogeneity is The main issue in data integration . Two approaches exist to resolve this heterogeneity:

- the virtual approach: the user sends his requests to a mediator in charge of transforming and translating a request in order to query the data sources; The data is only stored into the data sources.
- The materialized approach: the user sends his query to a target database. This database contains a copy of the integrated data of the data sources.

This paper presents the making of XeuTL, an ETL tool essential to the materialized approach. XeuTL extracts data from data sources, transforms this data and loads it into a target database. The user defines some mapping rules between the data sources physical schemas and that of the target database, thus resolving semantic heterogeneity.

Keywords : Integration, databases, semantic heterogeneity, ETL, extraction, transformation, load, XML