Consolidation d'un modèle conceptuel de données de Master Data Management

Ludovic Menet

Université de Marne-La-Vallée

Mémoire

Master Recherche
Information scientifique et technique (IST)

Consolidation d'un modèle conceptuel de données de Master Data Management

Ludovic MENET

2005-2006

Résumé

Actuellement, la majorité des Systèmes d'Information est caractérisée par une hétérogénéité en terme de données et de solutions de paramétrage. Cette diversité se situe dans les systèmes de stockage (bases de données, fichiers, annuaires…), les formats de données (tables, fichiers propriétaires, documents XML…) et les solutions proposées pour gérer les différents types de données. Cette hétérogénéité à la fois dans les données et dans les solutions existantes sur le marché a pour conséquence de rendre lourde, complexe et coûteuse la mise en œuvre et l'exploitation de ces données par les applications de l'entreprise. Utiliser un ensemble d'applications différentes afin de pouvoir gérer cette diversité dans les types de données entraîne inévitablement une redondance tant au niveau des données que des outils.
Afin de résoudre ces problèmes, Orchestra Networks a développé une solution appelée EBX.Platform, basée sur une architecture XML, permettant d'avoir une solution de Master Data Management dans un Système d'Information. L'unification des données de références, au sein d'EBX.Platform, passe par la définition d'un modèle conceptuel de données, basée sur la technologie XML Schema, et la définition d'un référentiel interne. Nous proposons donc de vérifier et de consolider le modèle conceptuel d'EBX.Platform et d'apporter des solutions aux problèmes rencontrés.

Mots Clés :

Bases de données hétérogènes, integration, langage XSD, Master Data Management, metaschéma XML, validation XML.

Abstract

Currently, the majority of the Information Systems is distinguished by heterogeneity both in datas and solutions for managing parameters. This diversity exists in storage systems (databases, files, directories…), formats of data (tables, files owners, XML documents…) and availables solutions for managing different types of datas. As a consequence this heterogeneity, both in datas and existing solutions, has to make both complex and expensive the use of these datas by enterprises' applications. Therefore, using differents applications for the same purpose entails inevitably redundancy in datas and tools.
In order to solve these problems, Orchestra Networks developed a solution called EBX.Platfom, based on a XML structure, allowing a Master Data Management solution in an Information System. Data's unification, within EBX.Platform, pass by the definition of a conceptual model based on the XML Schema technology, and the definition of an internal data warehouse. Thus we propose to check and consolidate EBX.Platform's conceptual model and bring some solutions to the encountered problems.

Keywords

Heterogeneous databases, integration, Master Data Management, XSD language, metaschema XML, XML validation.

Remerciements

Je tiens à remercier Catherine Pelachaud, directrice du laboratoire LINC; Myriam Lamolle, maître de conférence à l'IUT de Montreuil, sans qui ce stage n'aurait pu se faire qui m'a suivi durant celui-ci et qui m'a apporté une aide précieuse dans mes travaux; Amar Zerdazi, doctorant à l'IUT de Montreuil, qui m'a soutenu pendant toute la durée de mon stage ; et sans oublier toutes les personnes de l'IUT de Montreuil qui m'ont accueillis lors de ce stage.

Je souhaite également remercier la société Orchestra Networks composée de Christophe Barriolade, Martail Doré, Vincent Lajous, Eric Morel et Zhangyun Lei, qui a bien voulu m'accepter en tant que stagiaire durant ces 5 mois.

Sommaire

Tables des illustrations

Introduction

Dans le contexte de l'interopérabilité de sources de données hétérogènes, il existe deux principales approches d'intégration de données à savoir :

- L'approche virtuelle (ou par médiateur) **[24]**, approche synchone, consistant à utiliser un unique schéma de représentation des sources de données hétérogènes. Dans cette approche l'utilisateur interroge un médiateur qui a pour fonction de traduire des requêtes en sous-requêtes compréhensibles par les différentes sources de données.
- L'approche matérialisée (ou par entrepôt) **[18]**, approche asynchrone, dans laquelle l'utilisateur interroge un référentiel contenant une copie des données issues de différentes sources de données.

La société Orchestra Networks propose une implémentation de la deuxième approche par une architecture XML appelée EBX.Platform. Cette architecture permet aux entreprises, sans intervenir sur leurs bases de données et applicatifs existants, d'unifier la gestion de leurs données de référence (produits, tarifications, données légales, nomenclatures, paramètres techniques et applicatifs, etc.). Cette unification est opérée sur trois axes principaux :

- Définition du modèle de données pivot par l'intermédiaire du langage XML Schema **[1]**.
- Persistance dans un référentiel commun propre au produit, que ce soit en « file system », dans une base de données distante ou dans une base de données intégrée.
- Mise à disposition aux utilisateurs d'un outil Web générique et convivial de consultation, de mise à jour et de synchronisation du référentiel avec le système d'information de l'entreprise.

Une des plus-values majeure de EBX.Platform pour les entreprises est que le référentiel supporte le concept d'héritage d'instances. Les capacités de factorisation de données (définition unique des données communes) qu'apportent l'héritage et EBX permettent ainsi d'éviter les duplications et les problèmes qui y sont liés (coûts et risques). L'héritage est un mécanisme interne au référentiel EBX. Pour mettre en oeuvre l'héritage, la société Orchestra Networks, éditeur de EBX.Platform, a créé une ébauche de modèle conceptuel. Des difficultés particulières demeurent sur certains points, notamment : respect des contraintes d'intégrité selon les opérations effectuées (telles que création et suppression d'occurrences), prise en compte et gestion des impacts liés à l'héritage, définition des algorithmes d'optimisation automatiques par factorisation.

L'objectif de mon stage était donc de :

1. vérifier et consolider le modèle conceptuel actuel en s'inspirant des recherches existantes dans ce domaine ou des domaines connexes,
2. proposer des solutions opérationnelles aux problèmes rencontrés.

La première partie de ce document sera consacrée à la présentation des structures qui m'ont accueilli durant mon stage. La seconde partie présentera les approches virtuelle et matérialisée. La troisième partie aura pour but de découvrir EBX.Platform afin de présenter le contexte dans lequel j'allais intervenir. La quatrième partie de ce rapport sera, quant à elle, consacrée aux recherches réalisées dans le but de consolider le modèle conceptuel

d’EBX.Platform. Enfin la dernière partie abordera la problématique de l’intégration de données hétérogènes via XML.
Nous conclurons ce rapport par un bilan des travaux réalisés, ainsi que les perspectives à l’issue de mon stage.

I. Présentation des structures d'accueil

Mon stage de Master recherche s'est déroulé conjointement dans 2 structures : le Laboratoire INformatique et Communication et la société Orchestra Networks.

A. Le Laboratoire INformatique et Communication (LINC)

Le LINC est situé à l'IUT de Montreuil sous Bois (93), rattaché à l'Université de Paris 8. Ce laboratoire est composé de 7 enseignants-chercheurs ainsi que de 6 doctorants.
Les thèmes de recherche du laboratoire se placent autour des axes de la communication, en vue d'établir des modèles d'interaction, en particulier, d'interaction personnalisée et adaptée aux besoins de chaque utilisateur. Ces thèmes de recherche sont abordés sur plusieurs fronts : de l'analyse approfondie d'un corpus de comportements et d'interactions (par exemple annotation de vidéos illustrant la communication homme-homme ou homme-machine, annotation de documents (audio/vidéo/texte), à la communication avec un agent conversationnel ou bien avec une navigation adaptative sur le Web. L'approche « entreprise » est pluridisciplinaire et repose sur des travaux menés dans des domaines variés : Bases de Données Hétérogènes, méthode d'aide à la décision, logique floue, multimodalité, hypermédias, agents conversationnels.
Les activités dans cette ligne de recherche s'orientent autour de deux thèmes :

- Analyse et Représentation de l'Information Communicative : analyse et annotation de corpus multimodaux ; définition de modèles formels de représentation de l'information communicative ; modèle d'intégration d'informations hétérogènes ; étude de méthodes d'apprentissage et de classification des informations ; établissement de méthode d'aide à la décision pour pouvoir classifier de nouvelles informations.
- Communications et Adaptation : création d'interfaces adaptatives dans lesquelles le contenu de l'information présentée à l'utilisateur est adaptée à ses besoins et capacités, par exemple à travers des hypermédias adaptatifs ou des agents conversationnels.

B. La société : Orchestra Networks

Orchestra Networks est une société éditrice de logiciels, créée en 2000, spécialisée dans le Master Data Management (MDM). En tant qu'éditeur logiciel, Orchestra Networks investit énormément dans la recherche et développement, environ 30 années Hommes, lui permettant de rester leader dans ce secteur. Ainsi, Orchestra Networks a-t-elle été plébiscitée par de très grandes entreprises telles que Sanofi Aventis, Société générale, SBE, France Télécom, EDF, Cofinoga, Finaref, Maaf, Predica du Crédit agricol, Kraft, Celetem…
De plus, de son activité d'éditeur de logiciel, Orchestra Networks exerce aussi dans le secteur du conseil, notamment dans les entreprises si ce n'est pour citer Unilog, France Télécom, Novartis, Bull, Société Générale…

Le MDM est un moyen d'unifier, de gérer, et d'intégrer des données de références à travers le Système d'Information de l'entreprise. Ces données de références peuvent être de différentes natures :

- Produits, services, offres, tarifs
- Clients, fournisseurs
- Données réglementaires, données financières
- Organisations, structures, personnes
- Nomenclatures, personnes

Orchestra Networks est une des premières sociétés en France, à fournir une solution complète MDM, capable de gérer toutes sorte de données de référence, sous la forme d'un logiciel, EBX.Platform reposant sur une approche matérialisée.

II. Approche virtuelle, approche matérialisée

Les informations présentes dans un système d'information sont représentées et stockées dans une multitude de sources de données et ce de façon hétérogène. Les premières approches d'intégration de ces sources de données, pour les faire coopérer, ont été réalisées dans le cadre de systèmes de bases de données relationnelles, objets/relationnelles ou objets, au travers de la mise en place d'une fédération de bases de données. Le besoin essentiel est donc de pouvoir interroger différentes sources de données simultanément et de donner l'impression à l'utilisateur qu'il interroge une unique source de données. L'approche virtuelle et l'approche matérialisée tentent de répondre à cette problématique.

A. L'approche virtuelle

L'approche virtuelle, ou par médiateur, désigne une vision globale, par l'intermédiaire d'un unique schéma de représentation, de l'ensemble des différentes sources de données hétérogènes. Ce schéma global peut être défini automatiquement à l'aide d'outils, ou extracteurs de schémas **[26]**. Le projet *TSIMMIS* **[27]**, réalisé par des chercheurs de l'université de Stanford, se base sur cette approche. Un des objectifs de TSIMMIS est d'intégrer des sources hétérogènes, pouvant être très peu structurées et pouvant évoluer. Toujours dans les travaux de recherche abordant l'approche virtuelle, nous pouvons citer des projets tels que *Disco* **[28]** et *YAT* **[29]**.
Dans cette approche virtuelle les requêtes utilisateurs sont formulées selon la sémantique du schéma global extrait. L'éxécution de ces requêtes nécessite une traduction de celles-ci, en sous-requêtes adaptées à chacun des sous-schémas des différentes sources de données.

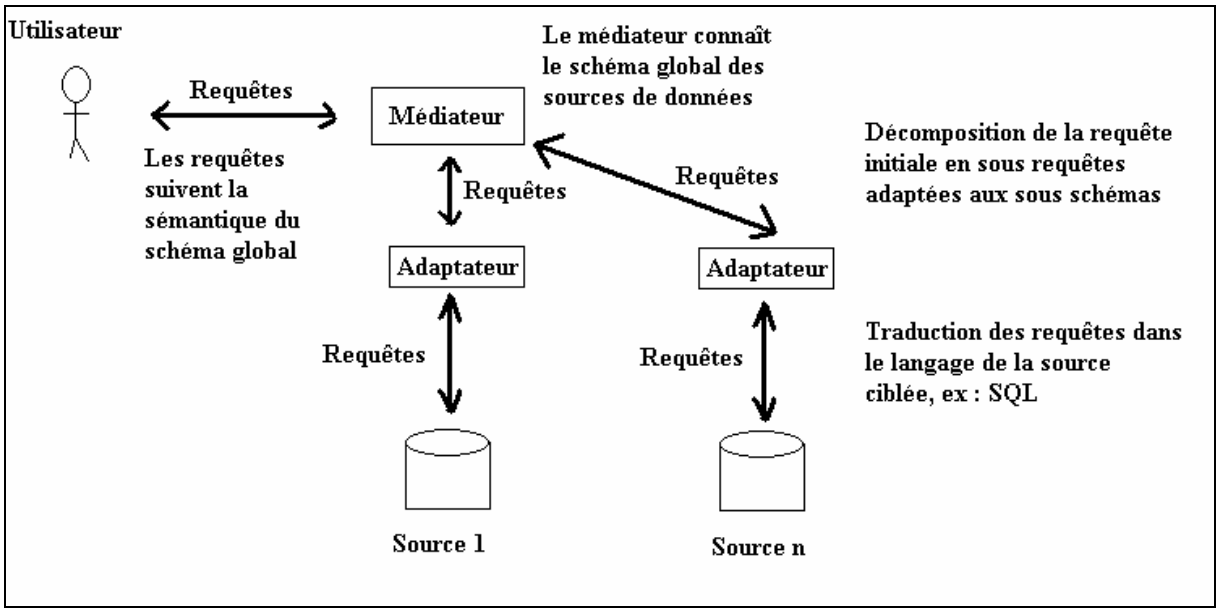


Figure 1 : illustration d'une architecture basée sur l'approche virtuelle

La figure 1 illustre une architecture basée sur une approche virtuelle. Dans cette approche les données sont stockées uniquement au niveau des sources. Les traitements sont donc synchronisés sur les sources de données. Dans cette approche le *médiateur* connaît le schéma global et possède des vues abstraites sur les sources, qui lui permettront de décomposer la requête initiale en sous-requêtes. Les *adapteurs* ont pour fonction de traduire les sous requêtes dans des langages compréhensibles par les différentes sources de données. Après le traitement de ces requêtes, par les différentes sources de données, les réponses suivent le cheminement inverse pour arriver à l'utilisateur.

B. L'approche matérialisée

Dans cette approche, les données, issues de sources hétérogènes, sont stockées dans un entrepôt de données (ou référentiel). Le projet *Xylème* **[19]** est un système d'entrepôt dynamique ayant pour but de stocker et d'intégrer de manière semi automatique toutes les ressources XML du Web. Ce stockage permet à l'utilisateur final d'avoir un accès unique et transparent à toutes les données hétérogènes. L'utilisation d'un système à base d'arbres **[23]**, contribue à faire de Xylème un système efficace pour l'évaluation de requêtes, l'intégration de données et leur maintenance.
L'approche matérialisée repose sur une copie des données dans un entrepôt, ainsi les actions sur le référentiel sont asynchrones par rapport aux sources de données. La propagation des modifications apportées au référentiel, vers les différentes sources de données, doit passer par des procédures de mises à jour.

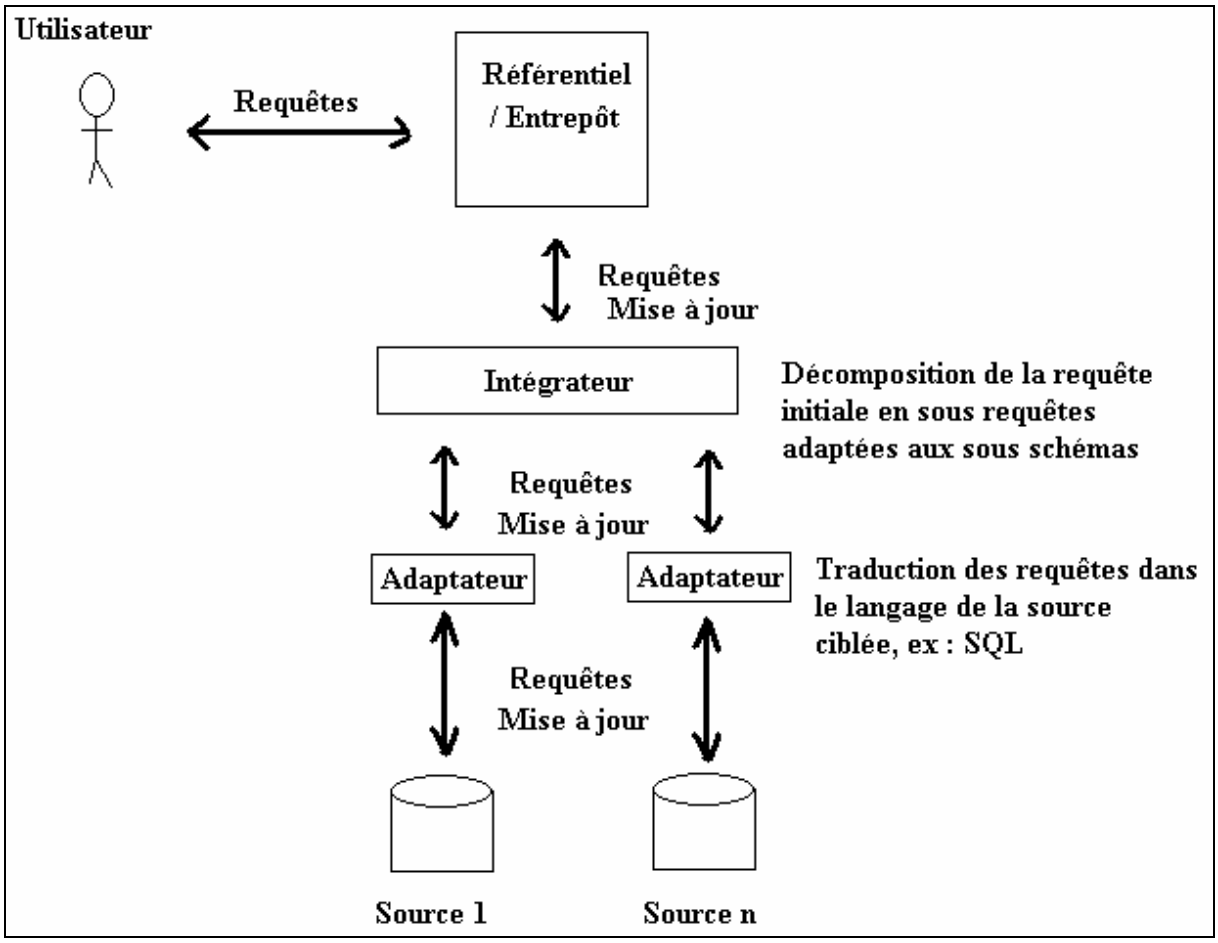


Figure 2 : illustration d'une architecture basée sur l'approche matérialisée

La figure 2 illustre une architecture basée sur une approche matérialisée. Contrairement à l'approche virtuelle, les requêtes utilisateurs sont directement exécutées dans le référentiel, sans avoir à accéder aux différentes sources de données. Dans cette approche les données du référentiel sont déconnectées de celles contenues dans les sources hétérogènes. Les mises à jour de données, du référentiel vers les sources de données ou inversement, sont déléguées à l'*intégrateur* qui a pour fonction de réaliser la correspondance entre le schéma du référentiel et les sous schémas des sources de données hétérogènes. Les *adaptateurs* traduisent les requêtes dans le langage des sources de données.

C. Bilan

L'approche virtuelle et l'approche matérialisée représentent deux solutions aux problèmes d'hétérogénéité des sources de données. Dans les deux méthodes, les différentes sources de données sont transparentes pour l'utilisateur. Le tableau suivant illustre les avantages et les inconvénients des deux approches :

	Points forts	Points faibles
Virtuelle	- Très bonnes performances en terme de volume, les données sont directement manipulées dans les sources. - Mises à jour rapides	- Performances, toute requête doit être traduite pour être interprètées par les différentes sources de données. - Gestion difficile de l'historique.
Matérialisée	- Performances, les actions sont directement effectuées, sans traduction, dans le référentiel. - Possibilité d'historisation des données au sein du référentiel - Systèmes de stockages efficaces (arbres…)	- Volume, les données sont à la fois dans le référentiel et dans les sources de données - Mise à jour nécessitant la copie des données du référentiel vers les sources de données ou inversement.

Nous pouvons constater que ces deux approches possèdent des points forts et des points faibles, l'approche à adopter dépend donc des besoins rencontrés.

La société Orchestra Networks propose un logiciel de Master Data Management appelé EBX.Platform, reposant sur une approche matérialisée.

III. EBX.Platform

Basé sur Java et XML Schema, EBX.Platform est une solution standard et non intrusive qui permet aux entreprises d'unifier et de gérer leurs données de référence et leurs paramètres à travers leurs systèmes d'information. EBX.Platform repose sur une approche « matérialisée » (cf. II), possédant ainsi de très bonnes performances au niveau des traitements (performances que nous présenterons plus loin), du fait de l'efficacité de son référentiel.

A. Pourquoi une solution MDM

Actuellement, la majorité des Systèmes d'information est caractérisée par une hétérogénéité en terme de données et de solutions de paramétrage. En effet, cette hétérogénéité se présente principalement sous trois aspects :

- Diversité des systèmes de stockage (bases de données, fichiers, annuaires…)
- Diversité des formats de données (tables, fichiers propriétaires, documents XML…)
- Diversité des solutions proposées pour gérer les différents types de données.

Cette hétérogénéité à la fois dans les données et dans les solutions existantes sur le marché a pour conséquence de rendre lourde, complexe et coûteuse la mise en œuvre et l'exploitation

de ces données par les applications de l'entreprise. Utiliser un ensemble d'applications différentes afin de pouvoir gérer cette diversité dans les types de données entraîne inévitablement de la redondance tant au niveau des données que des outils.

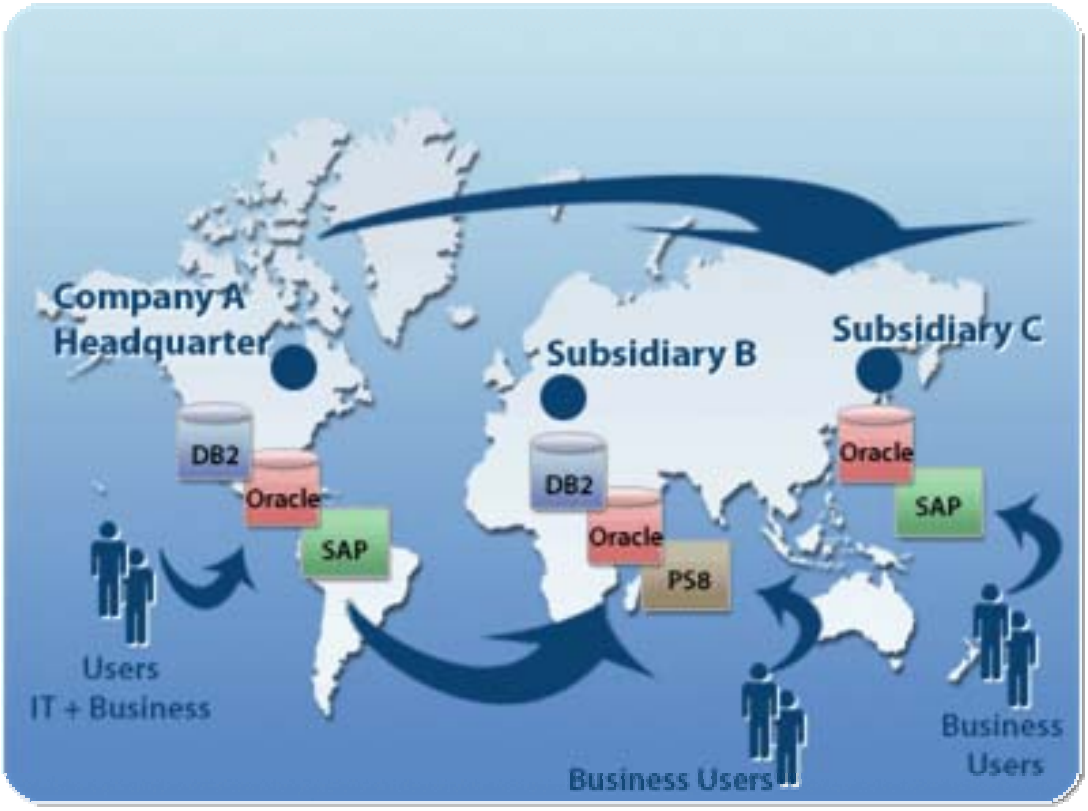


Figure 3 - Illustration d'un Système sans MDM

La figure ci-dessus présente un Système d'Information sans MDM, présentant les aspects suivants :

- Pas de vue unifiée des données de références
- Données dupliquées entre plusieurs systèmes
- Pas de cohérence entre les sièges et les filiales
- Pas d'outils uniques pour les utilisateurs métier

Les conséquences d'un système sans MDM sont les suivantes :

- Coût de maintenance plus élevé
- Manque de réactivité
- Risques réglementaires

Afin de résoudre ces problèmes, Orchestra Networks a développé une solution appelée EBX.Platform permettant d'avoir une solution MDM dans un Système d'Information.

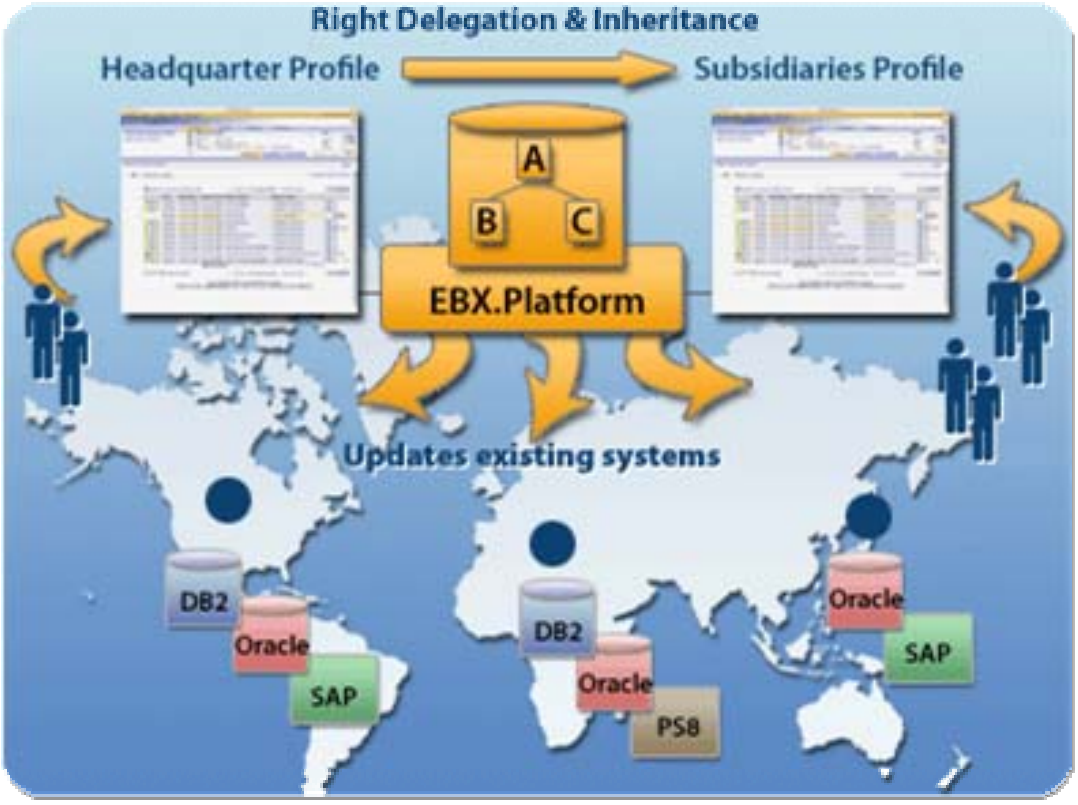


Figure 4 - Illustration d'un Système MDM utilisant EBX.Platform

La Figure 4 présente un Système d'Information MDM utilisant EBX.Platform et présentant les aspects suivants :

- Données unifiées dans un référentiel central
- Règles d'héritage entre les filiales et le siège
- Intégration des données de référence à travers le Système d'Information
- Disposition d'un outil unique pour tous les utilisateurs métier

L'utilisation de EBX.Platform apporte les avantages suivants :

- Abaissement du coût de maintenance
- Meilleure réactivité
- Respect des règles légales

B. L'architecture d'EBX.Platform

Basé sur le standard XML Schema, EBX.Platform simplifie la définition de modèles qui ont pour objectif d'unifier les données de référence d'une entreprise. En utilisant la technologie XML Schema, ces modèles peuvent être de tous types (simples, complexes) et de toutes natures (métiers, techniques, graphiques). Un des principaux avantages de XML Schema est de permettre la définition de modèles de données structurées, typées et ayant de puissantes propriétés de validation.

La Figure 5 schématise l'architecture d'EBX :

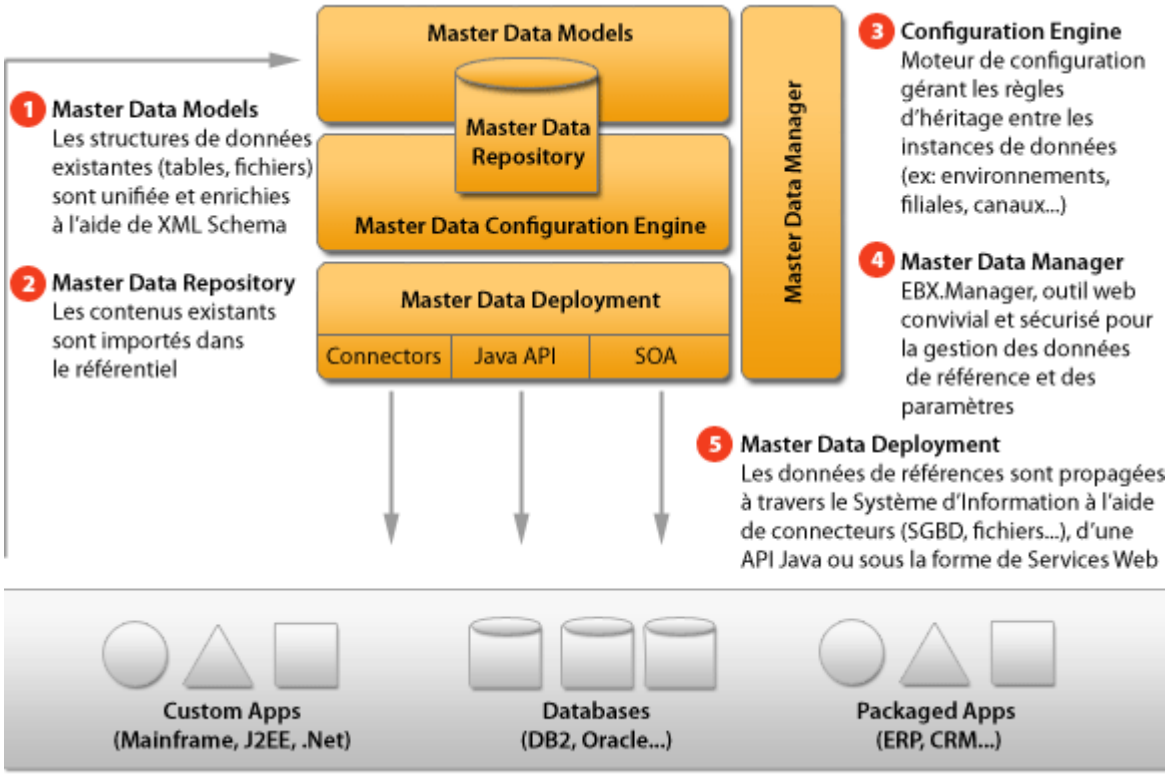


Figure 5 - Architecture d'EBX.Platform

Son architecture générale est notamment constituée de :

- Les modèles EBX (Master Data Models) permettent de créer une définition unifiée et enrichie des données de référence et paramètres. Ces modèles sont des documents XML Schema conformes à la recommandation du W3C [1].
- Le référentiel EBX (Master Data Repository) permet de gérer plusieurs instances de données de référence dans un référentiel unique.
- Le moteur de configuration (Master Data Configuration Engine) est conçu pour supporter de larges volumes de données et des mécanismes de factorisation des données tout en assurant une interaction avec l'utilisateur qui est automatisée, intuitive et sécurisée (interface homme-machine générée par introspection du modèle, libellés et descriptions, messages interactifs de validation, rapports de validation complets).

Le langage XML Schema apporte de nombreux avantages parmi lesquels on peut citer :

- une riche bibliothèque de types de données de base pleinement spécifiés ;
- une structure hiérarchique de base, simple à appréhender pour le développeur et l'utilisateur final ;
- un standard largement reconnu par l'industrie ;
- la spécification immédiate des formats d'échanges avec les sources de données hétérogènes (instances XML).

C. Concepts d'EBX.Platform

EBX.Platform repose sur 2 principes :

- **Les modèles d'adaptation** qui sont des documents XML Schema définissant la structure des données de références.
- **Les adaptations** qui sont les instances XML des modèles d'adaptation, représentant le contenu des données de références.

Les modèles d'adaptation représentent donc les modèles des données de références sous la forme d'un document conforme au formalisme XML Schema. L'utilisation de XML Schema permet de préciser que chaque nœud du modèle de données correspond à un type de données existant et conforme au standard du W3C.
EBX.Platform supporte la majorité des types de données utilisés pas XML Schema, des types complexes multi occurencés ; en outre, des types métiers supplémentaires sont aussi gérés.
Par ailleurs, le formalisme d'XML Schema permet de spécifier pour chaque nœud, du schéma, des contraintes (énumération, longueur, bornes inférieures et supérieures…), des informations relatives à l'adaptation et à l'instanciation (connecteurs d'accès, classe d'instanciation Java, restriction d'accès…) et des informations de présentation (libellé, description, formatage…).

Une adaptation est, au sens XML, une instance du modèle d'adaptation. A tout nœud du modèle d'adaptation, déclaré instanciable, correspond un nœud dans l'adaptation, l'instance XML.
Si un modèle d'adaptation possède plusieurs adaptations, alors on considère qu'un arbre d'adaptation est manipulé (cf. Figure 6). Au sein d'une adaptation, chaque nœud possède les propriétés suivantes :

- **Une valeur d'adaptation** : Si cette valeur n'est pas définie dans l'adaptation courante alors elle est héritée des ascendants (l'adaptation mère). Si aucune adaptation ascendante ne définit une valeur, alors la valeur est héritée par défaut du modèle de données (XML Schéma).
- **Un droit d'accès pour les descendants** : Le nœud d'adaptation peut être soit caché (aux descendants), soit en lecture seule (pour les descendants), soit en lecture/écriture (pour les descendants). Cette propriété peut ne pas être définie, auquel cas la valeur est héritée de l'adaptation mère.

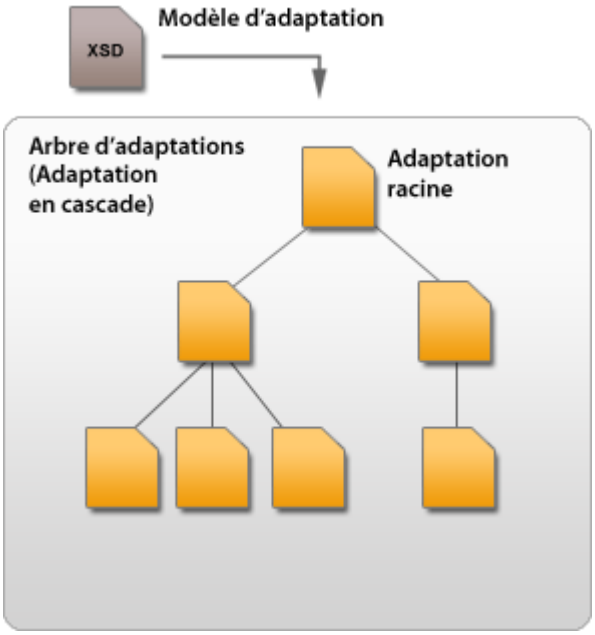


Figure 6 - Illustration d'un modèle d'adaptation et de ses instances

Comme nous avons pu le voir, un modèle d'adaptation est un modèle de données défini au moyen de XML Schema et ce pour les mêmes raisons citées précédemment, mais doit également suivre certains principes pour être interprétés par EBX.Platform.

D. Principes du modèle d'adaptation

Pour être accepté par EBX.Platform, un schéma XML doit au moins posséder une déclaration d'un élément racine et cette racine doit avoir l'attribut *osd:access="--"* (voir Figure 7).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:osd="urn:ebx-schemas:common_1.0"
xmlns:fmt="urn:ebx-schemas:format_1.0">

  <xs:import namespace="urn:ebx-schemas:common_1.0"
schemaLocation="http://schema.orchestranetworks.com/common_1.0.xsd"/>

  <xs:element name="root" osd:access="--">

 </xs:element>
</xs:schema>
```

Figure 7 – Exemple de déclaration d'un modèle d'adaptation contenant une seule racine

Dans les modèles d'adaptation défini à l'aide de XML Schema, nous pouvons considérer principalement cinq types de nœuds : les nœuds simples, simples multi occurencés, complexes, complexes multi occurencés et tables.

1) Les nœuds simples

Les nœuds simples sont définis à l'aide de la balise *« element »* (Figure 8) définie par XML Schema :

```
<xs:element name="noeud" type="xs:integer">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Exemple de nœud simple</osd:label>
          </xs:documentation>
        </xs:annotation>
    </xs:element>
```

Figure 8 – Exemple de déclaration d'un nœud simple

2) Les nœuds simples multi occurencés

Les nœuds simples multi occurencés sont définis de la même manière qu'un nœud simple, à l'exception près qu'ils permettent de définir une liste composée d'éléments identiques :

```
<xs:element name="noeud" type="xs:integer" minOccurs="0" maxOccurs= "100" >
        <xs:annotation>
          <xs:documentation>
            <osd:label>Exemple de nœud simple multi occurencé</osd:label>
          </xs:documentation>
        </xs:annotation>
    </xs:element>
```

Figure 9 - Exemple de déclaration d'un nœud simple multi occurencé

Les facettes *minOccurs* et *maxOccurs* (Figure 9) indiquent le nombre d'éléments minimal et maximal composant la liste.

3) Les nœuds complexes

Les nœuds complexes sont similaires aux nœuds complexes définis par XML Schema par la balise *xs:complexType* (Figure 10). En d'autres termes, ce sont des nœuds contenant d'autres nœuds (simples, complexes ou tables) :

```
<xs:element name="noeudComplex" >
   <xs:complexType>
      <xs:sequence>
         <xs:element name="element1" type="xs:integer">
            <xs:annotation>
               <xs:documentation>
                  <osd:label>Element simple 1</osd:label>
               </xs:documentation>
            </xs:annotation>
         </xs:element>
         <xs:element name="element2" type="xs:string">
            <xs:annotation>
               <xs:documentation>
                  <osd:label> Element simple 2</osd:label>
               </xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figure 10 – Exemple de déclaration d'un nœud complexe, contenant 2 éléments

4) Les nœuds complexes multi occurencés

De la même manière que les éléments simples multi occurencés, il est possible de définir des éléments complexes multi occurencés (Figure 11) :

```
<xs:element name="noeudComplex" minOccurs="0" maxOccurs= "100">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="element1" type="xs:integer">
            <xs:annotation>
               <xs:documentation>
                  <osd:label>Element simple 1</osd:label>
               </xs:documentation>
            </xs:annotation>
         </xs :element>
         <xs:element name="element2" type="xs:string">
            <xs:annotation>
               <xs:documentation>
                  <osd:label> Element simple 2</osd:label>
               </xs:documentation>
            </xs:annotation>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figure 11 – Exemple de déclaration d'un complexe multi occurencé, contenant 2 éléments

5) Les nœuds tables

Dans EBX.Platform il est possible de définir des tables de la même manière que dans les bases de données relationnelles où les attributs de la table relationnelle sont déclarés comme une séquence d'éléments simples dans EBX.Platform (cf. Figure 12).
La déclaration d'une table s'effectue de la façon suivante :

```
<xs:element name="product" minOccurs="0" maxOccurs="unbounded"">
   <xs:annotation>
      <xs:documentation>
         <osd:label>Table des Produits</osd:label>
         <osd:description>Liste des produits du catalogue</osd:description>
      </xs:documentation>
   </xs:annotation>
   <xs:complexType>
   <xs:annotation>
      <xs:appinfo>
         <osd:table>
            <primaryKeys>/productRange /productCode</primaryKeys>
         </osd:table>
      </xs:appinfo>
   </xs:annotation>
      <xs:sequence>
         <xs:element name="productRange" type="xs:string"/><!-- clé -->
         <xs:element name="productCode" type="xs:string"/><!-- clé -->
         <xs:element name="productLabel" type="xs:string"/>
         <xs:element name="productDescription" type="xs:string"/>
         <xs:element name="productWeight" type="xs:int"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Figure 12 – Exemple de déclaration d'une table contenant 5 champs

Les contraintes sur les clés primaires sont définies à l'aide de facettes étendues spécifiques à EBX.Platform que nous présenterons par la suite.

Si l'on défini un schéma XML avec les exemples de nœuds précédents, nous obtenons le modèle d'adaptation de la Figure 13 dans EBX.Platform.

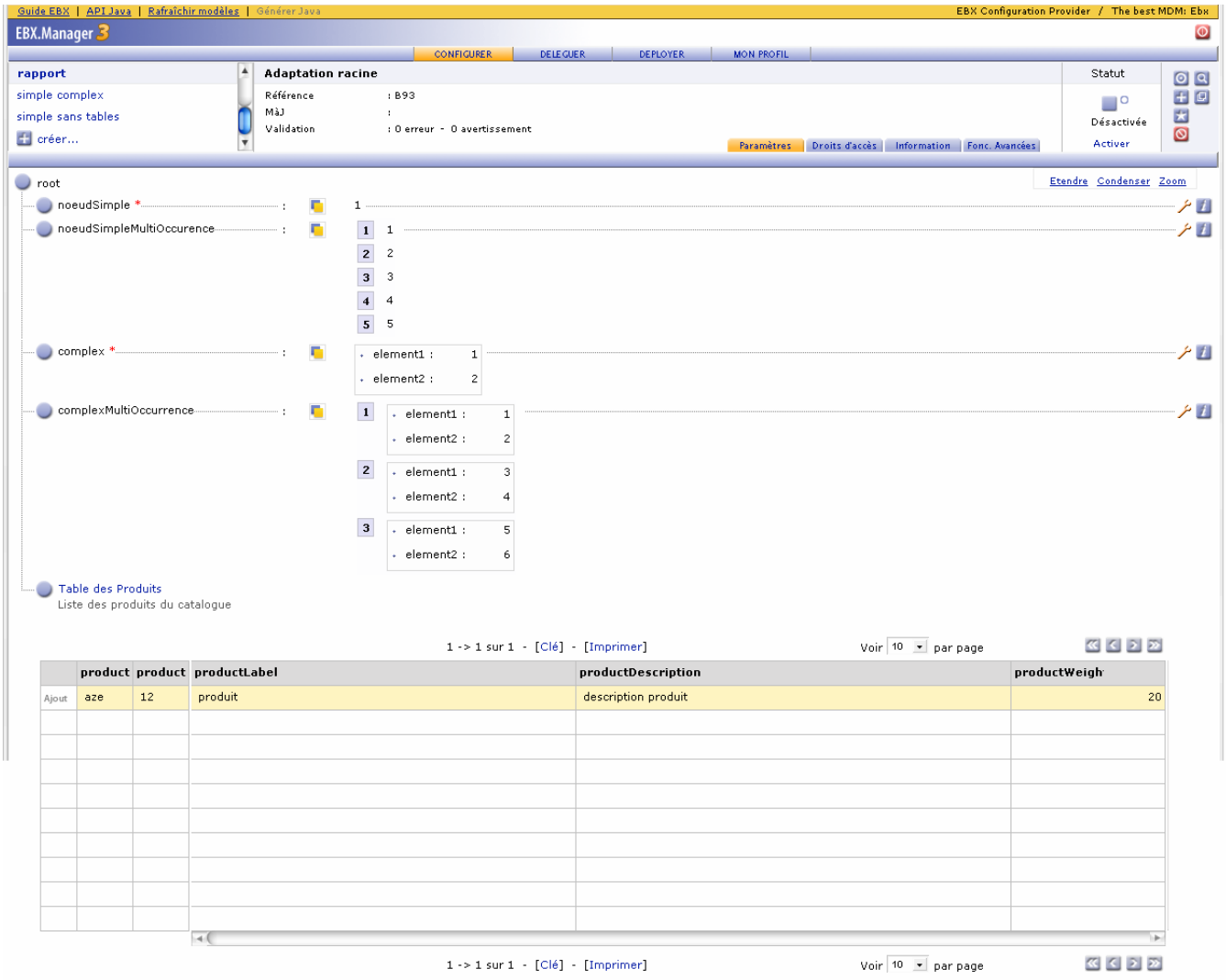


Figure 13 – Exemple d'utilisation d'un modèle d'adaptation

6) Facettes étendues

EBX.Platform propose des contraintes étendues, interprétées en JAVA, lorsque XML Schema ne permet pas de spécifier des contrôles avancés.
Afin de conserver la conformité du document par rapport à la norme XML Schema, ces facettes sont définies sous l'élément *annotation/appinfo/otherFacets*. Nous reviendrons sur la validité de ces balises dans la quatrième partie de ce document.
Les facettes suivantes ne sont qu'une partie des facettes disponibles dans EBX.Platform. Nous présentons quelques facettes permettant d'appréhender les extensions apportées à XML Schema.

a) Facettes dynamiques

Les facettes dynamiques permettent d'ajouter des contraintes par rapport à un élément défini dans le modèle d'adaptation ou bien dans le même contexte et peuvent être de natures suivantes :

- Length : contrainte sur la longueur d'un élément
- minLength : contrainte sur la longueur minimale d'un élément
- maxLength : contrainte sur la longueur maximale d'un élément
- maxInclusive : contrainte sur la borne maximale incluse d'un élément
- maxExclusive : contrainte sur la borne maximale exclue d'un élément
- minInclusive : contrainte sur la borne minimale incluse d'un élément
- minExclusive : contrainte sur la borne minimale exclue d'un élément

Par rapport aux balises standard définies par XML Schema, le nom de l'élément est conservé, cependant la facette statique *value* est remplacée par la facette *path*.

```
<xs:element name ="amount">
        <xs:annotation>
                <xs:appinfo>
                        <osd:otherFacets>
                                <osd:minInclusive path ="/path "/>
                        </osd:otherFacets>
                </xs:appinfo>
        </xs:annotation>
        …
</xs:element>
```

Figure 14 – Exemple d'utilisation d'une contrainte dynamique

Dans l'exemple de la Figure 14, la borne de la facette *minInclusive* est définie dynamiquement, la valeur de la borne est détenue par le nœud donné en paramètre.

b) Contrainte d'intégrité sur les tables (clés étrangères)

De même que pour la définition de clés primaires rattachées à une table, EBX.Platform permet de définir des clés étrangères, au même sens que dans le domaine des bases de données relationnelles .Une référence à une clé primaire de table est définie par une balise étendue *tableRef*.

Element	Description	Obligatoire
container	référence de l'instance qui contient la table.	Non, si cet élément n'est pas spécifié, l'instance est celle contenant la table.
tablePath	Expression XPath qui spécifie la table.	Oui.
labelPaths	Expressions XPath qui spécifient la composition du libellé.	Non, le libellé par défaut est la clé primaire.

displayKey	"true" ou "false", spécifie si la clé primaire est affichée en préfixe du libellé.	Non, la valeur par défault est "true".

```
<xs:annotation>
        <xs:appinfo>
                <osd:otherFacets>
                        <osd:tableRef>
                                <tablePath>../catalog</tablePath>
                                <labelPaths>/productLabel /productWeight</labelPaths>
                                <displayKey>true</displayKey>
                        </osd:tableRef>
                </osd:otherFacets>
        </xs:appinfo>
</xs:annotation>
```

Figure 15 – Exemple de définition de clés étrangères

La balise *tableRef* est interprétée comme étant une énumération. La clé étrangère formatée est la valeur de l'élément. Le libellé peut être composé en ajoutant les chemins d'autres champs des occurrences référencées par l'élément *labelPaths*.

E. Bilan

Nous avons vu précédemment qu'EBX.Platform est une solution performante, permettant de répondre aux besoins du Master Data Management. Tout modèle d'adaptation EBX.Platform est conforme à la norme XML Schema. De plus, EBX.Platform tire profit des possibilités d'extensions proposées par XML Schema, notamment en définissant des facettes étendues. Cependant la validation d'un modèle d'adaptation, plus précisément la validation des extensions introduites par EBX.Platform, est réalisée en interne et non par les mécanismes XML Schema ou d'autres technologies. Au-delà de ces contraintes de validation, une des plus-values d'EBX.Plafeform est que le référentiel interne supporte l'héritage au niveau instances. Cette notion d'héritage est conforme au concept de même nom du domaine objet. Il reste toutefois certains concepts objets qui pourraient être introduits dans les modèles d'adaptations.
Nous proposons donc dans la partie suivante de consolider le modèle conceptuel actuel et la validation des données.

IV. Consolidation du modèle conceptuel et de la validation des données

Dans cette partie il est plus précisément question de présenter des spécificités objets qui peuvent être ajoutées au modèle conceptuel, de proposer un méta-schéma d'un modèle d'adaptation ; enfin, de présenter un framework de test XML pour la validation des données.

A. Norme objet et modèle d'adaptation

Le référentiel d'EBX.Platform peut être persisté de différentes manières à savoir en file system (document XML) ou dans une base de données. C'est ce dernier type de persistance qui nous intéresse plus particulièrement. En terme de persistance, EBX.Platform gère différents types de bases de données : Oracle, DB2, HSQL… Cependant, la persistance en base de données est non standard. EBX.Platform étant développé avec un langage de programmation orienté objet (JAVA), la persistance pourrait être réalisée avec ce même langage en suivant des standards (sérialisation JAVA, norme ODMG…).

1) L'Object Data Management Group (ODMG)

L'Object Data Management Group, a pour fonction depuis 1993 de fournir des spécifications concernant la persistance d'objets dans des bases de données. Ainsi, l'ODMG propose trois types de spécifications objet : un modèle objet, un langage de spécification d'objet et un langage de requêtage. Nous présentons ici qu'une partie des spécifications de l'ODMG, qui permet d'appréhender la norme ODMG. Pour une description plus complète nous vous renvoyons au **[2]**.

- Le modèle objet :

L'ODMG défini quatre propriétés rattachées aux objets : un identifiant, un nom, une durée de vie et une structure.
L'identifiant de l'objet (*oid*) est utilisé par l'ODMG afin de pouvoir distinguer un objet d'un autre, notamment lors de requêtes pour rechercher un objet précis.
Un objet ODMG peut avoir un nom qui lui a été assigné par le développeur, à des fins particulières. L'ODMG fournit des moyens pour effectuer un mapping entre les noms d'objets et les objets eux-mêmes.
En terme de cycle de vie, l'ODMG gère 2 types d'objets : les objets persistants (persistent) et les objets transitoire (transient). Le cycle de vie d'un objet permet de spécifier à sa création la manière d'allouer la mémoire nécessaire à cet objet pour qu'il soit persisté.
La structure de l'objet permet d'indiquer sa composition (type, méthodes, attributs…) et la manière dont sera instancié cet objet par l'intermédiaire de constructeurs.

L'ODMG définit un autre de type de structure : les *littéraux*. L'ODMG défini un littéral comme étant une valeur sans *oid*. Un littéral peut être de nature simple ou complexe (nous pouvons rapprocher la notion de littéral à celle d'un nœud XML Schema qui peut aussi être simple ou complexe). Trois types de littéraux peuvent être définis : atomiques

(types de base tels que integer, float, char, string…), collections (List, Array, Bag…) et structurés (tuple()).

De même qu'en JAVA, l'ODMG défini les notions d'interface et d'héritage sur les objets.

• Le langage de spécification : Object Description Language (ODL)

L'ODMG a défini un langage pour spécifier la structure d'un schéma, dit ODL. Il faut noter qu'ODL est uniquement un langage de définition d'objets et non un langage de programmation. ODL possède une syntaxe précise permettant la définition de classes, d'interfaces, de relations, etc. L'ODMG a aussi mis en place un formalisme graphique de représentation d'un schéma. Cette représentation graphique (Figure 16) peut s'apparenter à UML**[6]** :

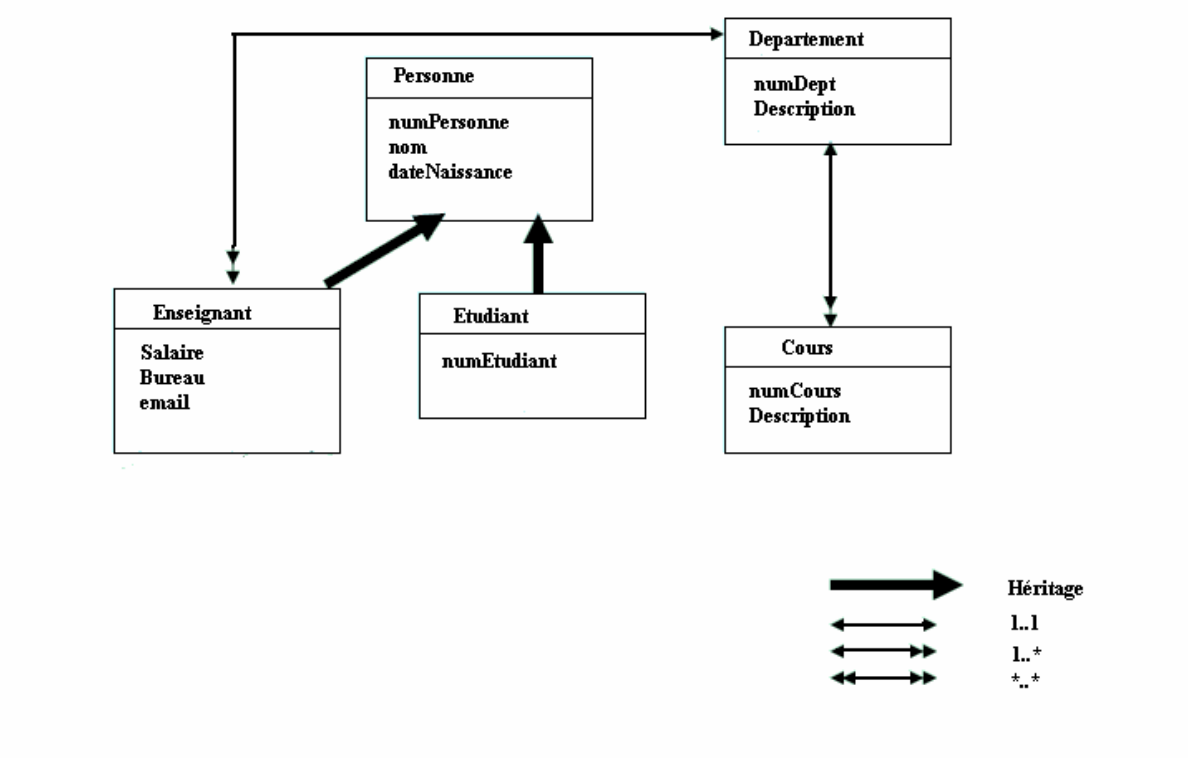


Figure 16 – Représentation d'un schéma ODMG

```
Class Personne (extent Personnes key numPersonne)
{
        attribute string numPersonne ;
        attribute struct nomComplet( string nomFamille, string prenom) nom ;
        attribute Date dateNaissance ;
};
Class Enseignant extends Personne (extent Enseignants)
{
        attribute float salaire ;
        attribute string bureau ;
        attribute string email ;
        relationship Departement associeAu Inverse Departement :: aDesEnseignants ;
};
Class Etudiant extends Personne (extent Etudiants)
{
        attribute string numEtudiant ;
```

```
};
Class Departement (extent Departements key numDept)
{
        attribute string numDept ;
        attribute string description;
        relationship Enseignant aDesEnseignants Inverse Enseignant :: associeAu ;
        relationship Cours aCours Inverse Cours :: aLieu ;
}
Class Cours (extents lesCours key numCours)
{
        attribute string numCours ;
        attribute string description ;
        relationship Departement aLieu Inverse Departement :: aCours ;
} ;
```

Figure 17 – Schéma ODL de la figure 14

• Le langage de requêtage : Object Query Language (OQL)

L'ODMG définit un langage d'interrogation d'objets, appelé OQL, très proche du langage SQL (Structured Query Langage). OQL permet de réaliser des requêtes qui ont pour but d'accéder aux informations situées dans le référentiel de persistance.
Tout comme le langage SQL, une requête OQL de sélection est construite à partir de certains mots clés tels que *select, from, where, like, having, order by, groupe by*…
La clause *from*, analogue à la même clause SQL, peut porter sur plusieurs modèles d'objets ou plusieurs requêtes OQL.
OQL dispose de différents opérateurs afin de réaliser des requêtes plus ou moins complexes :

- Comparaison : != , < , <= , > , >=
- Arithmétiques : +, -, *, /, ^
- Booléens : &&, ||
- Agrégation : count(X), min(X), max(X), sum(X), avg(X)
- Union : X+Y
- Intersection : X*Y
- Différence : X-Y

Ci-dessous, des exemples de requêtes OQL :

Noms des enseignants : select E.nom from Enseignants E ;
Nombre d'enseignants : select count(E) from Enseignants E ;
Description d'un cours donné : select C.description from Cours C where C.numCours = 'cours1' ;

L'ODMG a défini un standard de persistance pour des bases de données s'inscrivant dans la norme Object DataBase Management System (ODBMS).

2) Bases de données ODBMS

A titre d'exemple, nous allons présenter dans cette partie deux bases de données ODBMS libres.

- Orient ODBMS : http://lnx.orientechnologies.com/cms

Orient est une base de données orienté objet 100% compatible avec la norme ODMG 3.0., supportant les langages ODL, pour la spécification de schémas, et OQL pour l'interrogation. Orient est compatible avec JAVA via la technologie *Java Data Object* (JDO), C++ via les frameworks fournis par ODMG 3.0. Dans notre cas, ce qui nous intéresse est l'interaction entre la base de données et JAVA.
A titre d'exemple d'utilisation de cette base, nous reprendrons le schéma ODL illustré dans la Figure 17. Deux modes d'utilisation existent afin d'exploiter cette base de données orientée objets. Le premier moyen pour un utilisateur est d'accéder à l'interface utilisateur fournie avec la base de données. L'interface utilisateur est idéale dans le cas où l'on souhaite réaliser une exploitation manuelle de la base de données. Cependant ce que l'on recherche est l'exploitation de cette base au travers d'un langage de programmation orienté objet. Pour ce faire, Orient met à disposition une API JAVA qui permet de créer un « pont » entre le langage de programmation et la base de données. L'avantage d'exploiter ainsi la base de données est de pouvoir réaliser des traitements automatiques à partir de n'importe quelle application. Dans notre cas, ce qui nous intéresse, est de persister des objets dans la base. Notre but, ici, n'est pas de fournir un manuel complet d'utilisation de cette API, mais de montrer par l'intermédiaire d'un exemple simple, qu'il est aisé d'exploiter cette base directement avec un langage de programmation.
Dans notre exemple, nous considérons que le schéma ODL, présenté précédemment, a été importé dans la base de données, définissant ainsi la structure de nos objets et leurs relations.
L'exemple de la Figure 18 montre la création d'un objet de type *Enseignant* et va le persister dans la base de données.

```
//Connexion à la base
Properties props = new Properties();
props.setProperty("javax.jdo.PersistenceManagerFactoryClass",
                                        "com.orientechnologies.jdo.oPersistenceManagerFactory");
props.setProperty("javax.jdo.option.ConnectionURL", "nomBDD");
PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory(props);
PersistenceManager pm = pmf.getPersistenceManager();
pm.currentTransaction().begin();

//Création d'un objet
Enseignant ens = new Enseignant();
ens.setNomComplet("nom", "prenom");
ens.setSalaire(1234);
ens.setEmail("adresse@domaine");
ens.setBureau("bureau1") ;

pm.makePersistent(ens);

//Validation vers la base
pm.currentTransaction().commit();
```

Figure 18 – Exemple de persistance vers la base Orient

Comme toute base de données, il est aussi possible d'effectuer des requêtes :

```
Query query=pm.newQuery(Enseignant.class);
Collection collection = (Collection) query.execute();
Iterator cursor = collection.iterator();
while(cursor.hasNext())
{
        Enseignant city = (Enseignant) cursor.next();
        System.out.println( ens );
}
```

Figure 19 – Exemple d'interrogation de la base Orient

L'ODBMS Orient représente une solution pour la persistance d'objets, et est conforme aux normes ODMG 3.0. De plus, avec une base de ce type, la persistance s'effectue de façon transparente et simple.

- Database for objects (db4o) : www.db4o.com

Db4o est une base de données objet native en JAVA et .Net qui possède de fortes capacités de persistance et d'interrogation d'objets, directement par l'intermédiaire d'une API ou par un manager graphique. Nous allons, de la même manière que nous l'avons fait avec Orient, présenter comment exploiter simplement cette base de données.
L'exemple de la Figure 20 montre comment créer une connexion vers db4o et comment persister un objet dans la base.

```
// Connexion à la base
ObjectContainer db = Db4o.openFile("chemin de la Base de données");

//Création d'un objet de type enseignant
Enseignant ens = new Enseignant();
ens.setNomComplet("nom", "prenom");
ens.setSalaire(1234);
ens.setEmail("adresse@domaine");
ens.setBureau("bureau1") ;

//Persistance de l'objet
db.set(ens) ;


// Fermeture de la base
db.close();
```

Figure 20 – Exemple de persistance vers la DB4o

Comme nous pouvons le constater la persistance d'un objet vers db4o s'effectue de manière très simple. Précisons aussi que les objets que l'on souhaite persister peuvent être de toutes sortes et n'ont pas à implémenter une interface spécifique à db4o. La seule obligation pour

qu'un objet puisse être persisté dans la base est que chaque champ de la classe, décrivant l'objet, doit posséder une méthode publique permettant d'y accéder (en lecture et écriture).

Un des aspects importants de cette base de données est de proposer un mode d'interrogation très puissant et directement utilisable via l'API fournie avec la base de données. Plus précisément db4o défini 2 manières, plus où moins performantes et simples, permettant d'effectuer des requêtes.

1. Le premier mode de requête est basé sur la méthode *Query By Exemple(QBE)*, créée par Moshe Zloof **[25]** pour le compte de la compagnie IBM, en 1977. Cette méthode consiste à définir un prototype d'objet que l'on souhaite obtenir à la suite d'une requête. L'avantage que l'on entrevoit dans cette méthode est que l'utilisateur n'a pas à connaître un langage de requêtage particulier. L'exemple ci-dessous illustre une requête simple QBE.

```
Enseignant enseignantPrototype = new Enseignant (« nomRecherché ») ;
ObjectSet result = db.get(enseignantPrototype);

While ( result .hasNext() )
{
        //Traitements à effectuer sur chaque enregistrement
        …….
}
```

Figure 21 – Exemple d'interrogation QBE

Nous pouvons constater que QBE est pratique pour des requêtes simples. Cependant, avec ce type de requête, il n'est pas possible de réaliser des opérations booléennes telles que le ET logique, le OU logique, la négation… Le second type de requête proposé par Db4o comble ces limites.

2. *Native Query* (NQ) a pour but d'utiliser le langage de programmation comme langage de requête. Le développeur a donc besoin de connaître seulement un langage. La syntaxe de la requête peut ainsi être contrôlée directement à la compilation et par un IDE tel qu'Eclipse par exemple. Le « white paper » de William R. Cook et Carl Rosenberger **[3] ([4])** illustre plus en détails cette méthode de requêtage.

```
List <Enseignant> list = db.query(new Predicate<Enseignant>()
                        {
                                public boolean match(Enseignant ens)
                                {
                                        return ens.getSalaire() > 1000;
                                }
                        });
```

Figure 22 – Exemple d'interrogation NQ

Figure 22 illustre avec un exemple l'utilisation de NQ. Dans cet exemple, nous recherchons tous les enseignants ayant un salaire supérieur à 1000. NQ utilise le principe de la définition de classe anonyme en JAVA pour définir une requête. Pour que cette classe soit considérée par Db4o comme une requête NQ, elle doit définir la méthode *match* où seront définis les critères de sélection. DB4o itèrera sur chaque objet de type *Enseignant* afin de répondre à la requête. La requête étant définie à l'aide du langage de programmation, le développeur peut décrire toutes sortes de requêtes, et ce de manière très simple.

3) Bases ODBMS et EBX.Platform

La Figure 23 illustre la persistance des instances au sein d'EBX.Platform en utilisant des bases de données classiques, telles que Oracle et DB2, et des bases de données ODBMS telles que Orient et Db4o. Nous avons pu voir précédemment que les bases de données ODBMS sont parfaitement adaptées à la persistance d'objet et possèdent des techniques de requêtage performantes, ce qui pourrait être profitable en termes de développement et d'exploitation. En effet EBX.Platform ne disposant pas de moyen permettant d'effectuer des requêtes sur des adaptations, l'utilisation des solutions ODBMS pourraient être une solution à ce manque. De plus, l'utilisation de bases ODBMS permettrait de persister des relations objets entre concepts. Cependant, la gestion de l'héritage est réalisée au cœur même d'EBX.Platform ; il s'agit d'un héritage au niveau données, non au niveau structurel, et non défini dans le modèle d'adaptation. Il pourrait s'avérer alors important d'intégrer des métadonnées objets dans le modèle d'adaptation.

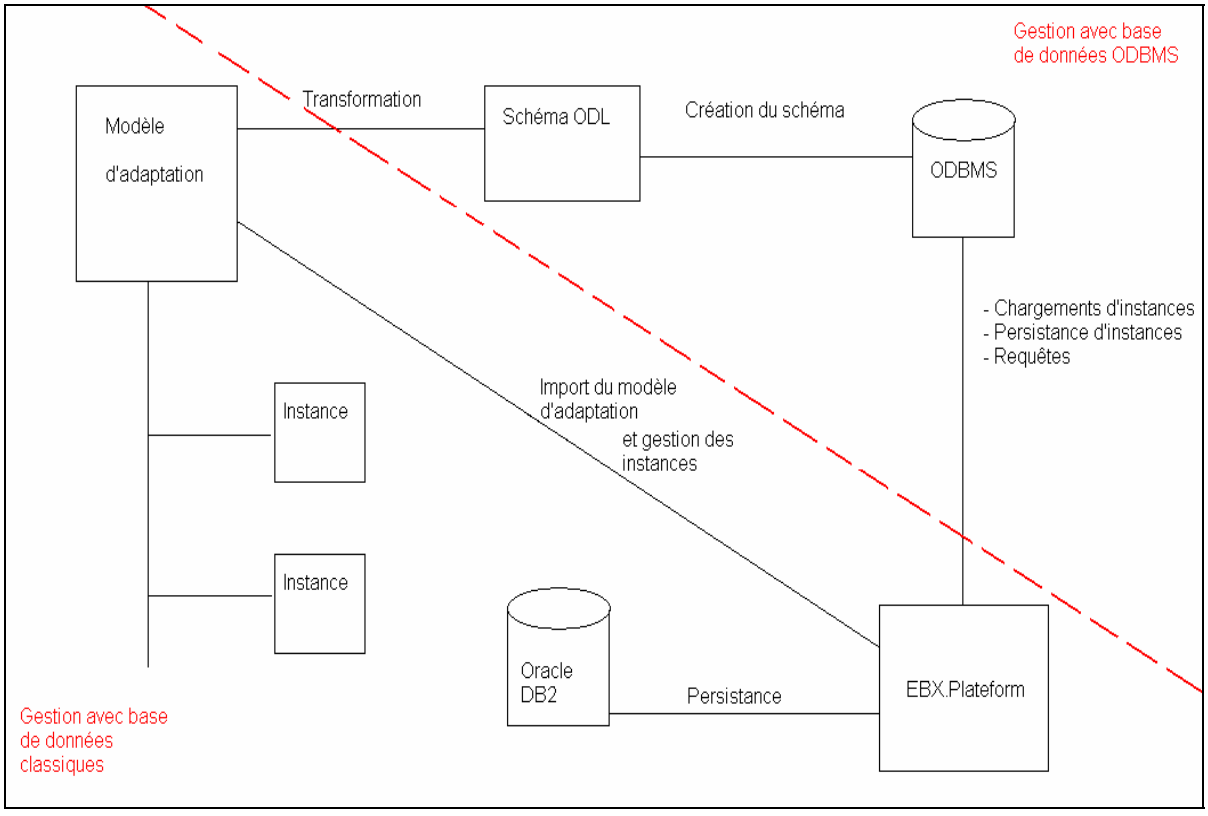


Figure 23 – Schéma de persistance ODBMS et Base de données classiques

4) Propositions d'ajout de métadonnées objet dans le modèle d'adaptation

En termes de relations entre concepts objet, nous pouvons citer les notions suivantes : la *généralisation*, la *spécialisation* et la *composition*.

- La *généralisation* peut se définir comme étant le regroupement d'attributs, d'opérations et d'association, communs à plusieurs concepts. Par exemple la figure 14, que nous avons vu précédemment, illustre la notion de généralisation des concepts *Etudiant* et *Enseignant* par l'intermédiaire du concept *Personne* qui a pour fonction de mutualiser les attributs communs (dans notre exemple). A l'inverse, la *spécialisation* est l'affinement d'un concept en sous-ensembles. En reprenant le même exemple, les concepts *Etudiant* et *Enseignant* sont des spécialisations du concept *Personne*. En effet, les concepts *Etudiant* et *Enseignant* possèdent des propriétés supplémentaires telles qu'un numéro d'étudiant pour le concept *Etudiant*, et un numéro de bureau et un salaire pour le concept *Enseignant*.
- La notion de *composition* indique qu'un concept appartient à un autre concept. Par exemple, si l'on considère une université composée de différents départements, alors on peut définir le concept *Université* comme étant une composition de départements. L'hypothèse de composition entre concepts a pour conséquence de créer une dépendance entre ceux-ci. En effet, la notion de composition sous-entend qu'il ne peut y avoir d'instances du concept *Département* sans instances du concept *Université*.

La Figure 24 illustre l'utilisation de ces concepts par l'intermédiaire d'un diagramme UML.

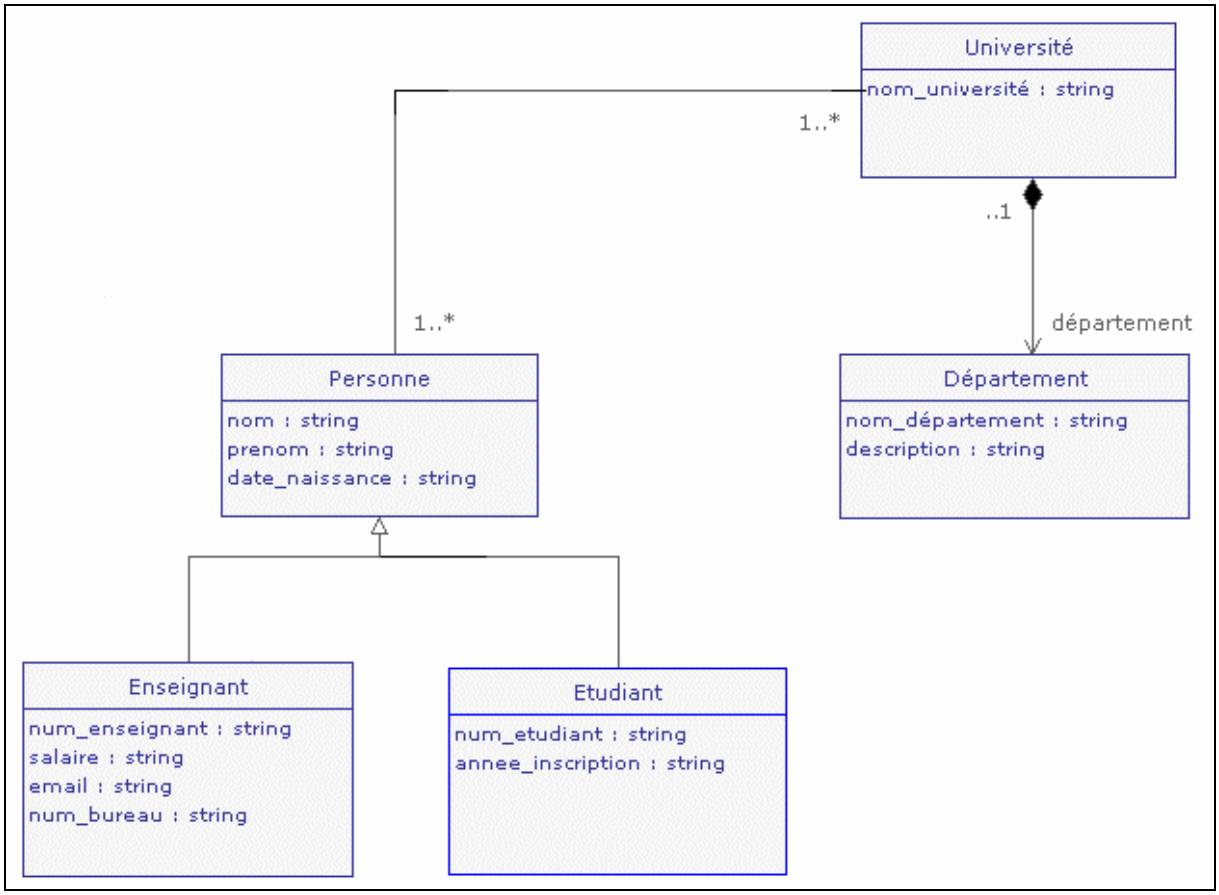


Figure 24 – Diagramme UML illustrant les notions d'héritage et de composition

Ce digramme présente les concepts *Personne, Enseignant, Etudiant, Département* et *Université*. Les concepts *Etudiant* et *Enseignant* héritent du concept *Personne*. Ces 2 premiers concepts sont donc des spécialisations du concept *Personne*. A l'inverse, ce dernier est une généralisation des concepts *Etudiant* et *Enseignant*. Le concept *Université* est défini comme étant une composition d'instances du concept *Département*.
De la même manière que *Lamolle et Zerdazi* **[5]**, nous proposons donc des métadonnées à inclure dans le schéma XML, afin de matérialiser ces concepts.
Comme le recommande le W3C, les extensions XML Schema que nous présentons seront définis dans l'élément « *appInfo* ». Nous allons donc décrire les balises suivantes :

- Généralisation :

```
<xs:annotation>
    <xs:appinfo>
            <osd:generalisation>
                    <osd:conceptPath>Path du ou des sous concepts</osd:conceptPath>
            </osd:generalisation>
    </xs:appinfo>
</xs:annotation>
```

- Spécialisation :

```
<xs:annotation>
    <xs:appinfo>
            <osd:specialisation>
                    <osd:conceptPath>Path du ou des super concepts</osd:conceptPath>
            </osd:specialisation>
    </xs:appinfo>
</xs:annotation>
```

- Composition :

```
<xs:annotation>
    <xs:appinfo>
            <osd:composition>
                    <osd:conceptPath minOccurs='0' maxOccurs='unbounded'>Path du
                    concept constituant la composition </osd:conceptPath>
            </osd:composition>
    </xs:appinfo>
</xs:annotation>
```

Remarque : la composition sera considérée comme une liste d'instances du concept associé.

En introduisant ces relations dans le modèle d'adaptation associé à notre exemple, nous obtenons le schéma **A1** situé en annexes.

L'ajout de ces métadonnées dans le modèle d'adaptation permet de mettre en évidence des relations entre certains concepts. Ces métadonnées contribuent à optimiser certains traitements tels que la factorisation de données, l'optimisation d'arbres, la suppression d'instances devenues inutiles. En effet, si l'on considère, dans notre exemple, la notion de d'agrégation entre les concepts *Université* et *Département*, une instance du concept *Département* ne peut pas exister sans instance du concept *Université*. De ce fait, lors de la suppression d'une instance, appartenant au concept *Université*, toutes les instances dépendantes de celle-ci seront supprimées.
Nous avons vu qu'il était possible d'ajouter des métadonnées dans un modèle d'adaptation en utilisant les caractéristiques d'extension de XML Schéma. Ces balises sont de manière générale utilisées comme suit :

```
<xs:annotation>
        <xs:appInfo>
                <osd:XXX>   </osd:XXX>
        </xs:appInfo>
</xs:annotation>
```

Ainsi ont été définies de nombreuses facettes et concepts interprétables par EBX.Platform, comme nous avons pu le constater dans les exemples précédents. En définissant les extensions XML Schéma dans les balises *annotation* et *appInfo*, il n'est pas nécessaire de fournir un schéma permettant de définir la structure de celles-ci. Un des problèmes résultant de ce constat est que la validation de ces extensions est déléguée intégralement au moteur de validation d'EBX.Platform, et non au moteur de validation de XML Schéma. Nous proposons alors de définir un métaschéma d'un modèle d'adaptation, décrivant la structure des concepts apportés par EBX.Platform.

5) Méta-schéma d'un modèle d'adaptation

Ce métaschéma est défini en annexes **A2** sous la forme d'un schéma XML. Le métaschéma défini, il est possible d'utiliser le moteur de validation de XML Schéma. Pour utiliser ce métaschéma XML, la déclaration suivante doit être faite dans le modèle d'adaptation que l'on souhaite définir :

<xs:import namespace="urn:ebx-schemas:common_1.0" schemaLocation="path_schema"/>.

L'identifiant de l'espace de nom est spécifié dans le métaschéma par l'entête suivant :

<xs:schema xmlns:osd="urn:ebx-schemas:common_1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:ebx-schemas:common_1.0" elementFormDefault="qualified" attributeFormDefault="qualified">

B. Définition d'un profil UML

La définition d'un modèle d'adaptation repose sur la technologie XML Schéma. L'utilisation de XML s'avère adaptée aux besoins d'EBX.Platform, mais implique une connaissance étendue de ce langage. Il existe de nombreux outils XML Schema, mais d'une part le modélisateur peut utiliser des caractéristiques de XML Schema qui ne sont pas implémentées par EBX et d'autre part il n'est pas guidé en ce qui concerne les extensions EBX (il doit alors passer en mode texte XML verbeux et non assisté). Il découle de ce constat qu'un formalisme doit être utilisé afin de rendre cette modélisation plus aisée.
L'apparition des langages orientés objets ont donné naissance à différents langages de modélisation : OMT **[20]**, Booch **[21]**, OOSE **[22]**… Chacune de ces méthodes décrit un formalisme pour construire des modèles objets. Dans les années 90, l'Unified Modeling Language (UML) **[6]** est né de la fusion des modèles OMT, Booch et OOSE.
UML est un langage de modélisation objet de plus en plus utilisé. Par nature, un modèle UML ne peut pas être productif (dans le sens où cette méthode est un formalisme de modélisation et non un langage de programmation), compte tenu de la multitude de langage existant et d'une sémantique trop générale. Il en découle le besoin de spécialiser, de profiler, UML pour des domaines précis.

Un profil permet de spécialiser le formalisme UML pour un domaine particulier. De nombreux profils ont été développés pour des domaines ciblés ou des technologies particulières. Par exemple, les profils les plus répandus sont les profils CORBA et EJB. Dans le cas du profil EJB, il est par exemple possible de préciser qu'une classe au sens UML est une *EJB Session* ou un autre concept de ce domaine. Pour plus d'informations sur le formalisme UML nous vous recommandons l'ouvrage **[6]**.
UML est basé sur une architecture quatre couches présentées dans la Figure 25 :

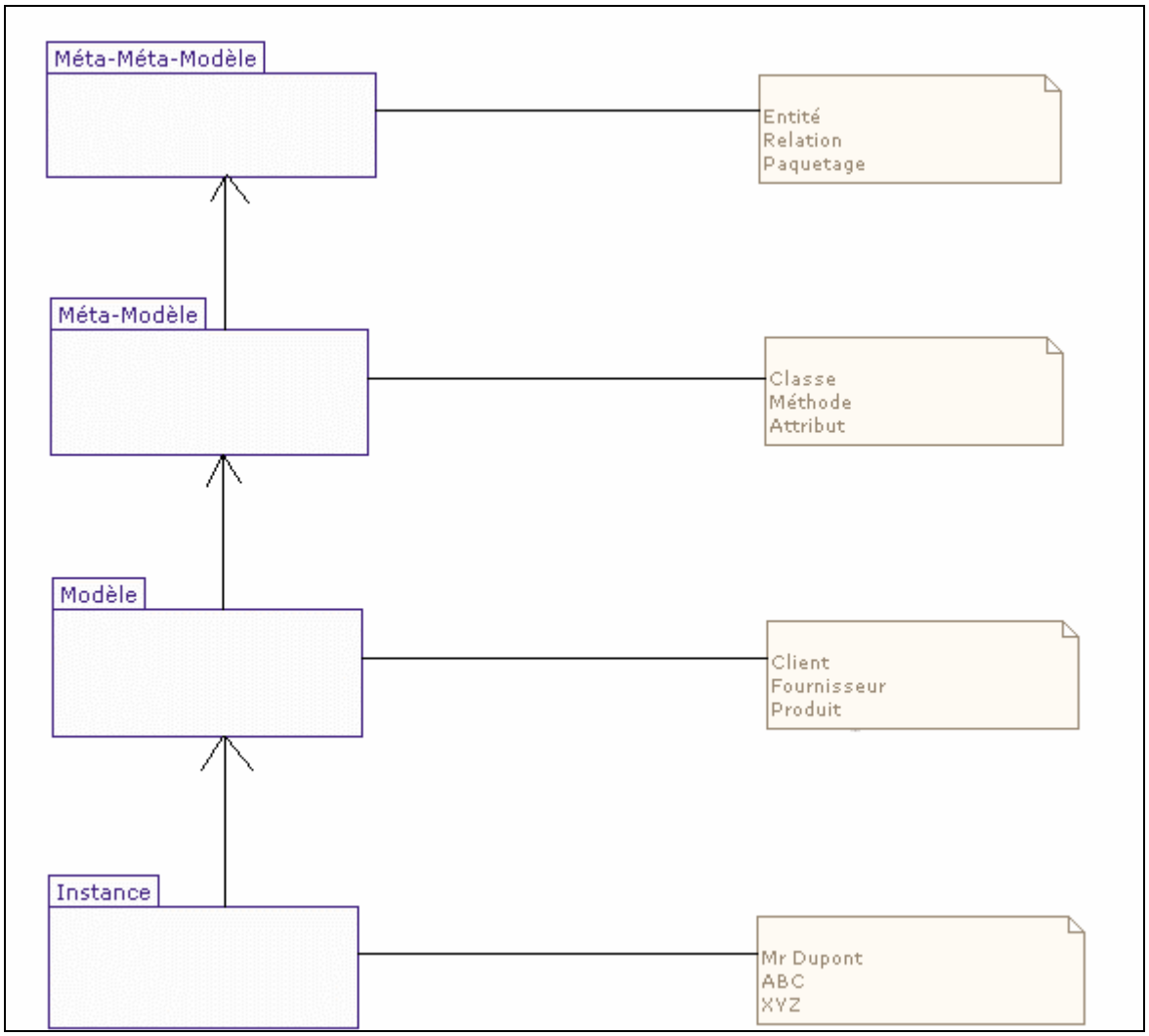


Figure 25 – Architecture 4 couches UML

Le méta-modèle UML (représenté partiellement dans la Figure 26) est défini en UML en accord avec l'architecture précédente :

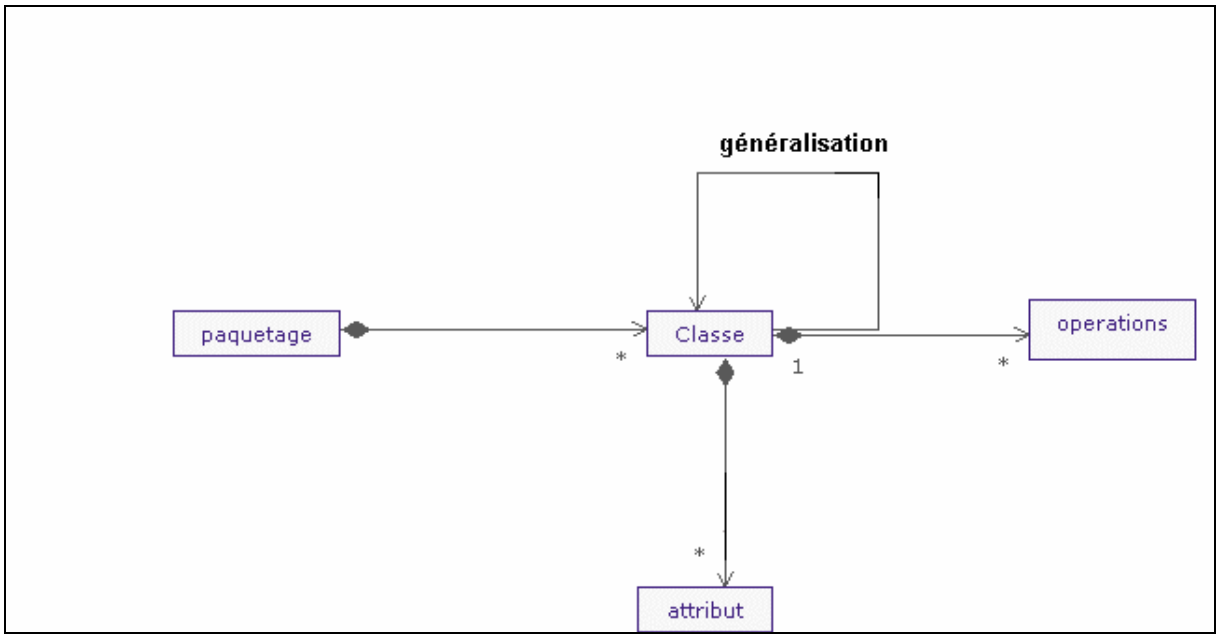


Figure 26 – Extrait du méta-modèle UML

La possibilité de définir des profils UML a donné lieu à différents travaux tel que le projet VUML sur l'analyse et la conception par point de vue, réalisé par *Nassar Mahmoud* durant sa thèse **[7]**. Dans VUML, il est question de proposer une modélisation orientée point de vue. Nassar Mahmoud introduit par exemple le concept de classe *multivue* ayant pour but de stocker et de restituer l'information selon le profil d'un utilisateur (gestion de droits d'accès, changement dynamique de point de vue, vérification de la cohérence de différents points de vue).

La démarche adéquate à la définition d'un profil est la suivante :

- Définir le domaine du profil, c'est-à-dire le méta-modèle définissant les concepts et relations entre ces concepts.
- Définition technique du profil en établissant les correspondances entre les concepts UML et les concepts définis dans le profil.
- Définir un exemple simple et concret du profil.

1) Profil du méta-modèle du modèle d'adaptation

Après étude du métaschéma XML réalisé précédemment (cf annexe A2), nous pouvons en déduire le profil UML définissant les relations entre les différents concepts introduits par EBX.Platform.

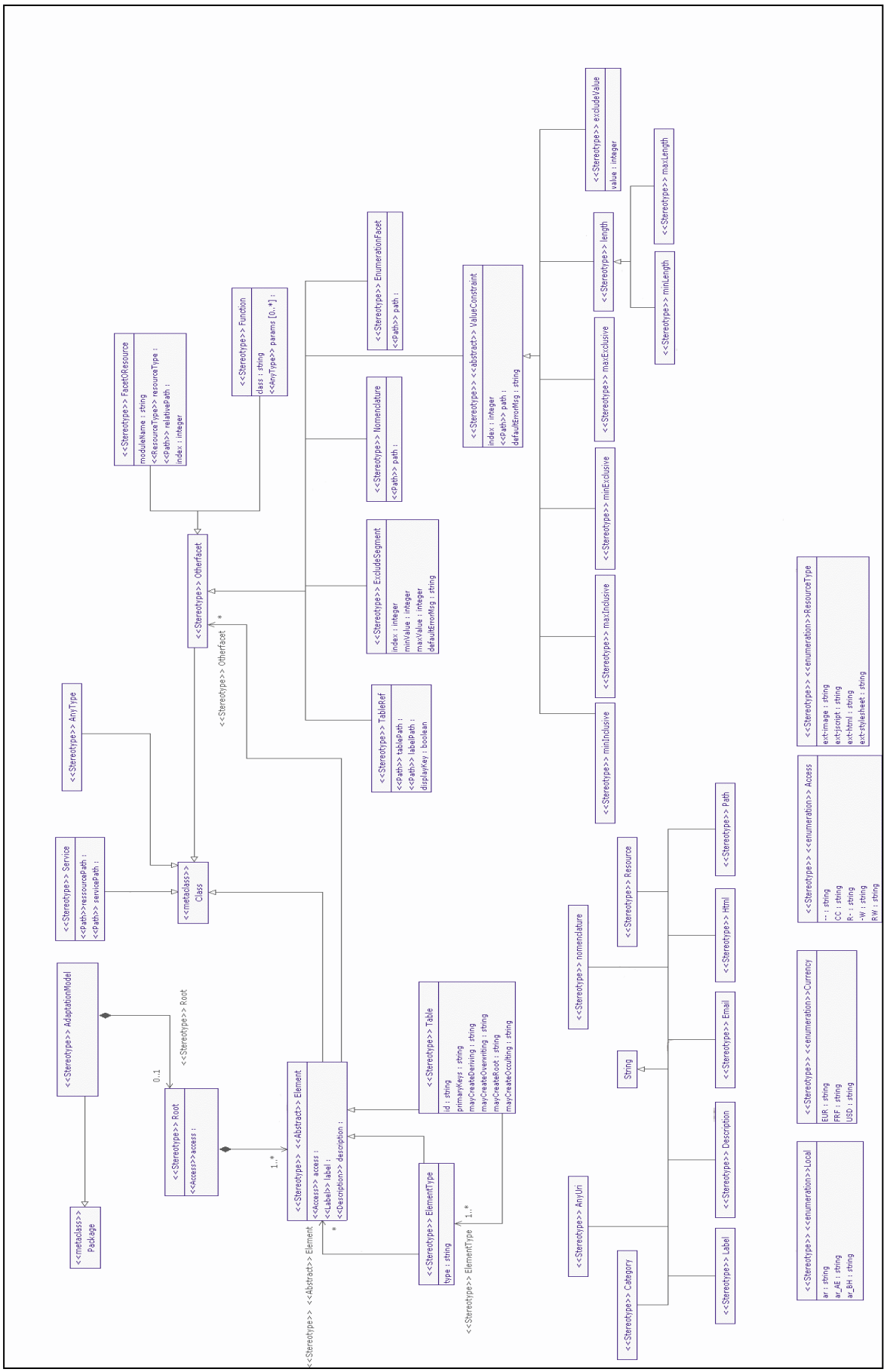


Figure 27 – Profil UML représentant le méta modèle d'EBX.Platform

2) Définition technique du profil

Le tableau suivant présente la description technique des éléments, stéréotypes, définissant le profil UML de la Figure 27. Pour chaque stéréotype défini figure le type d'élément UML auquel celui-ci peut être appliqué, ainsi qu'une brève description :

Stéréotypes	Appliqué à	Description
AdaptationModel	Package	Le package ciblée représente un modèle d'adaptation
Root	Class	La classe ciblée représente la racine d'un modèle d'adaptation
Element	Class	La classe ciblée est abstraite et représente un élément ou une table
Table	Class	La classe ciblée représente une table
ElementType	Class	La classe ciblée représente un élément simple ou complexe au sens XML Schema
OtherFacets	Class	La classe ciblée défini des contraintes sur un élément
ValueConstraint	Class	La classe ciblée est abstraite et représente une contrainte générique sur un élément
MinInclusive	Class	La classe ciblée représente une contrainte de type min inclusive sur un nombre
MaxInclusive	Class	La classe ciblée représente une contrainte de type max inclusive sur un nombre
MinExclusive	Class	La classe ciblée représente une contrainte de type min exclusive sur un nombre
MaxExclusive	Class	La classe ciblée représente une contrainte de type max exclusive sur un nombre
Length	Class	La classe ciblée représente une contrainte sur la longueur d'une chaîne de caractères
MinLength	Class	La classe ciblée représente une contrainte sur la longueur minimale d'une chaîne de caractères
MaxLength	Class	La classe ciblée représente une contrainte sur la longueur maximale d'une chaîne de caractères
ExcludeValue	Class	La classe ciblée représente une contrainte de type exclusion sur un nombre
ExcludeSegment	Class	La classe ciblée représente une contrainte de type exclusion sur un intervalle de valeurs
TableRef	Class	La classe ciblée représente une contrainte sur un élément, l'élément une clé étrangère
Service	Class	La classe ciblée représente un service EBX
Nomenclature	Class	La classe ciblée représente une contrainte générique de type Nomenclature.
Constraint	Class	La classe ciblée représente une contrainte générique de type Constraint.
EnumerationType	Class	La classe ciblée représente une contrainte indiquant que la classe est alimentée par une autre classe
Function	Class	La classe ciblée représente un ensemble de valeurs calculées par une fonction
FacetOResource	Class	La classe ciblée représente un élément de type FacetOResource
Label	Attribute	Chaine de caractère représentant un libelé
Description	Attribute	Chaine de caractère représentant une description d'un élément
AnyURI	Attribute	Chaine de caractère représentant une URI
Resource	Attribute	Chaine de caractère représentant une ressource
Locale	Attribute	Enumeration contenant des codes pays ex : ar, ar_AE, ar_BH
Currency	Attribute	Enumeration contenant les valeurs "EUR FRF USD"
Email	Attribute	Chaine de caractère représentant une adresse mail
Html	Attribute	Chaine de caractère représentant du code HTML
Path	Attribute	Attribut de type Xpath
Access	Attribute	Enumeration contenant les valeurs "-- CC R- -W RW"
Category	Attribute	Chaine de caractère représentant une catégorie
ResourceType	Attribute	Enumeration contenant les valeurs "ext-image ext-jscript ext-html ext-stylesheet"
Nomenclature	Attribute	Définition d'une valeur calculée

3) Exemple d’utilisation du profil EBX.Platform

Nous allons illustrer l’utilisation du profil défini précédemment par un exemple. Le modèle d’adaptation, situé en annexes **A3**, défini un modèle utilisant les types de base définis par EBX.Platform. Ce modèle d’adaptation est composé d’un nœud complexe contenant des éléments de type simple et complexe et d’une table. Schématiquement le modèle d’adaptation présenté est articulé de la manière suivante :

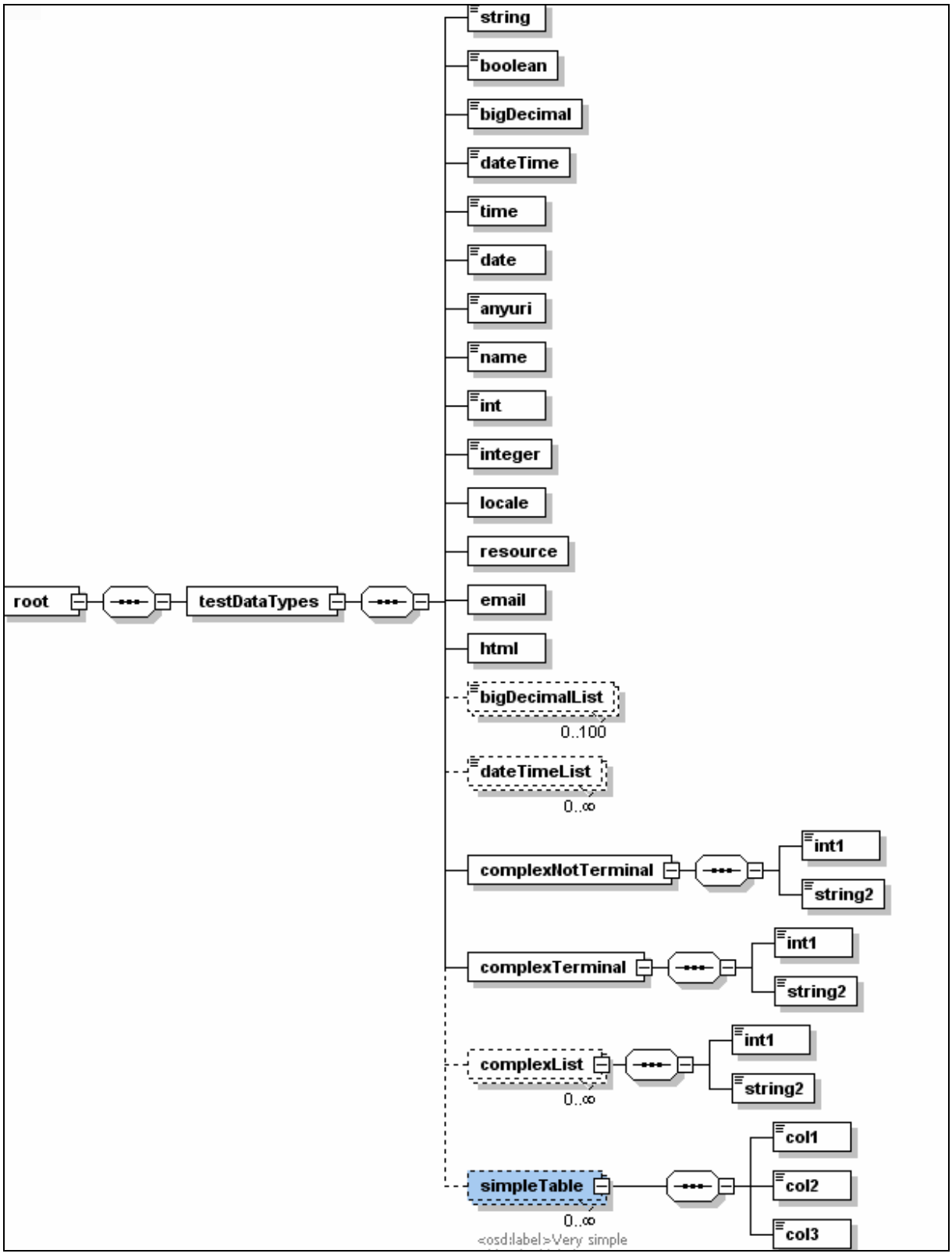


Figure 28 – Schéma Modèle d'adaptation présentant les types de base d'EBX.Platform

Après application du profil EBX.Platform déterminé précédemment, le modèle d'adaptation peut être défini sous la forme du diagramme UML de la Figure 29 :

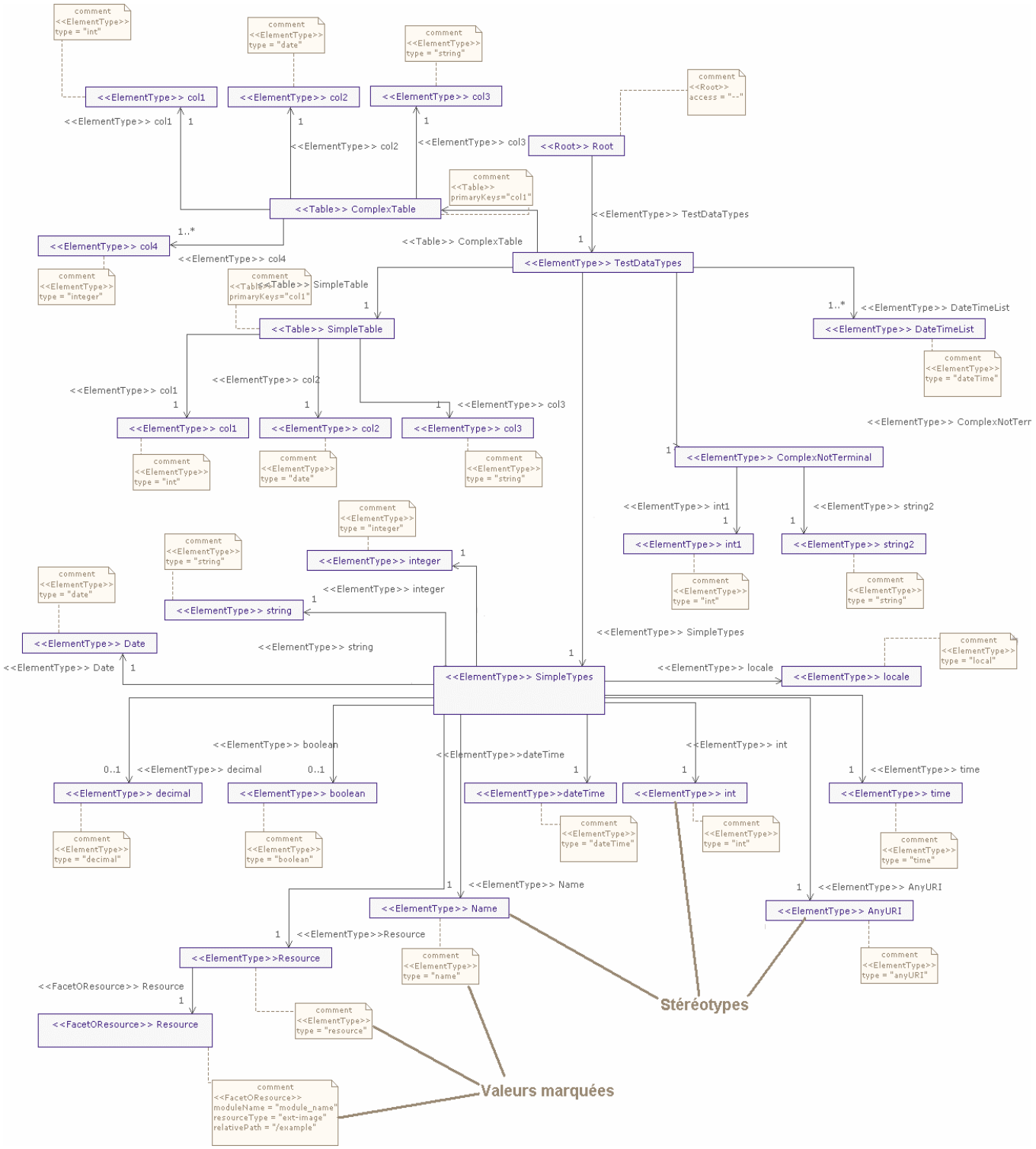
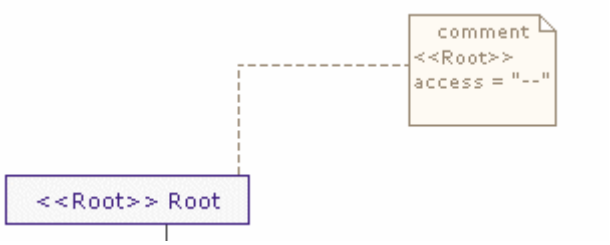
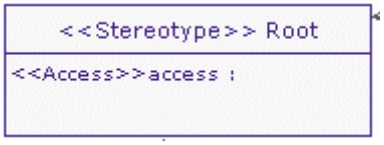


Figure 29 – Diagramme UML définissant un modèle d'adaptation

Le diagramme UML présenté, comporte des stéréotypes et des valeurs marquées. Les stéréotypes correspondent au type des éléments définis, plus précisément s'il s'agit d'un élément de type simple ou complexe ou d'une table. Les valeurs marquées correspondent aux valeurs données par l'utilisateur aux champs définis dans le stéréotype utilisé. Par exemple, dans le diagramme ci-dessous, nous avons la définition de l'élément racine du modèle d'adaptation par l'utilisation du stéréotype « Root ».



Si nous revenons sur la définition du méta-modèle, le stéréotype est défini de la manière suivante :



Le stéréotype *Root* possède un attribut *Access*. La valeur marquée dans le diagramme UML correspond à la valeur qui est assigné à cet attribut.

La définition d'un profil UML pour le méta-modèle d'EBX.Platform, associé à un générateur de code XML Schéma, permet de rendre la définition d'un modèle d'adaptation plus simple et plus sûre. En effet, en spécifiant un tel méta-modèle, l'utilisateur est contraint d'utiliser la sémantique définie par EBX.Platform, évitant ainsi certaines spécificités de XML Schéma qui ne sont pas gérées par EBX.Platform.

C. Framework de tests XML

Le référentiel d'EBX.Platform est persisté à l'aide du standard XML ou de bases de données. En tant que solution de MDM, EBX.Platform dispose de moyens pour propager des données dans le système d'information d'une entreprise, mais aussi pour importer des données. Cette propagation peut se faire sous la forme d'un document XML. Il demeure actuellement un manque dans les tests de validité de données. Pour combler ce manque, des outils de validations XML ont dû être réalisés.

1) Etat de l'art

Diverses technologies de tests et de manipulations de documents XML ont été développées et sont dans le domaine du logiciel libre. Nous allons présenter plusieurs solutions :

- Test Suite W3C : http://www.w3.org/XML/2004/xml-schema-test-suite/schemaframework.html.

  Le W3C défini un framework de test pour la validation de documents XML par rapport à la norme XML Schema.
  Les tests de ce framework sont définis dans un fichier *metadata* sous la forme d'un document XML. Les informations suivantes, pour chaque test, sont définies dans ce fichier metadata :
  - Un ou plusieurs schémas et instances sont définis
  - Le résultat attendu est indiqué
  - Documentation sur le test
  - Un historique des précédents tests est conservé

  La Figure 30 illustre un jeu de tests :

```
<testSet contributor="Microsoft" name="MS-attribute2002-01-16" xmlns="TestSuite"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="TestSuite AnnotatedTSSchema.xsd"
xmlns:xlink="http://www.w3.org/1999/xlink">
    <testGroup name="attA001">
        <annotation>
            <documentation>Syntax Checking for Attribute Declaration (form) Specs section: 3.2.3
Constraints on XML Representations of Attribute Declarations - A Test attribute declaration with optional attribute form
= 'qualified' at top level</documentation>
        </annotation>
        <documentationReference xlink:href="http://www.w3.org/TR/2004/REC-xmlschema-1-
20041028/#d0e2403"/>
        <schemaTest name="attA001">
            <schemaDocument xlink:href="./attribute/attA001.xsd"/>
            <expected validity="invalid"/>
            <current status="accepted" date="2002-01-16"/>
        </schemaTest>
    </testGroup>
    <testGroup name="attA002">
        <annotation>
            <documentation> </documentation>
        </annotation>
        <documentationReference xlink:href="http://www.w3.org/TR/2004/REC-xmlschema-1-
20041028/#d0e2403"/>
        <schemaTest name="attA002">
            <schemaDocument xlink:href="./attribute/attA002.xsd"/>
            <expected validity="invalid"/>
            <current status="accepted" date="2002-01-16"/>
        </schemaTest>
    </testGroup>

</testSet>
```

Figure 30 – Extrait d'un jeu de test défini par Microsoft

Après exécution de ce jeu de tests, via le logiciel fourni par le W3C, un rapport de tests est généré, comme suit :

XML Schema Test Descriptions for Microsoft tests.

The display below organizes test descriptions into groups of schema and instance tests. The structured description for these tests is stored in an XML file which conforms to the TS Schema for Test Submission. The column labeled "TestId" displays test identification information. The column labeled "Expected" displays the outcome expected by the contributor. The column labeled "Status" indicate test's accuracy in testing the feature it is intended to test. The "Description" column is provided by the test contributor. For the test file(s) present which has/have extension .xsd, its/their conformance to XML Schema REC's definition of valid XML representations of XML Schemas is what is at issue. When a test file with extension .xml is present as well, its schema-validity is at issue as well.

Test Set Summary
Test set name:MS-attribute2002-01-16
Contributor:Microsoft
Total number of test groups:254
Number of schema tests:254
Number of instance tests:107

TestId	Expected	Status	Description
1 attA001 Test type:<>schemaTest (1) Test files:1.attA001 Test type:<>instanceTest (0)	=validity Expected:invalid	=status Status:accepted as of 2002-01-16	=documentation Syntax Checking for Attribute Declaration (form) Specs section: 3.2.3 Constraints on XML Representations of Attribute Declarations - A Test attribute declaration with optional attribute form = 'qualified' at top level
2 attA002 Test type:<>schemaTest (1) Test files:1.attA002 Test type:<>instanceTest (0)	=validity Expected:invalid	=status Status:accepted as of 2002-01-16	=documentation Syntax Checking for Attribute Declaration (form) Specs section: 3.2.3 Constraints on XML Representations of Attribute Declarations - A Test attribute declaration with optional attribute form = 'unqualified' at top level

- XML unit : http://xmlunit.sourceforge.net

XML unit est une extension de JUnit (http://www.junit.org/index.htm), framework de tests JAVA, développé afin de proposer des méthodes de tests sur des documents XML. XMLunit propose les tests suivants :

- Validité d'une instance
- Egalité de 2 instances
- Inégalité entre 2 instances
- Similarité entre 2 instances
- Différences entre 2 instances
- Egalité structurelle entre 2 instances
- Validité de chemins XPath
- Test de valeurs obtenir par un chemin XPath
- Test du nombre de nœud dans une instance
- Test de transformation xslt

Les tests de validation peuvent porter sur une DTD ou un schéma XML.
L'utilisation de ce framework se fait de manière très simple. Pour définir une classe JAVA de test, celle-ci doit hériter de la classe *XMLTestCase*.

```
        //Teste la validité d'une instance XML
    public void testValidateSchemaValid() throws Exception
    {
            Validator validator = new Validator(new FileReader("tutoriel_voitures_valide.xml"));
            validator.useXMLSchema(true);
            assertFalse("test de validation d'une instance avec un schema xml", validator.isValid());

    }


    //Teste l'égalité de 2 instances XML
    public void testForEquality() throws Exception
    {
            String myControlXML = "<msg><uuid>0x00435A8C</uuid></msg>";
            String myTestXML = "<msg><uuid>0x00435A8C</uuid></msg>";
            assertXMLEqual("test de l'égalité de 2 instances", myControlXML, myTestXML);
    }

    //Teste l'inégalité de 2 instance XML
    public void testForInequality() throws Exception
    {
            String myControlXML = "<msg><uuid>0x00435A8C</uuid></msg>";
            String myTestXML = "<msg><id>12345</id></msg>";
            assertXMLNotEqual("test xml not similar to control xml", myControlXML, myTestXML);
    }

    //teste la similarité de 2 instances
    public void testSimilarity() throws Exception
    {
            String myControlXML = "<struct><int>3</int><boolean>false</boolean></struct>";
            String myTestXML = "<struct><boolean>false</boolean><int>3</int></struct>";
            Diff myDiff = new Diff(myControlXML, myTestXML);
            assertTrue("pieces of XML are similar " + myDiff, myDiff.similar());
    }
```

Figure 31 – Exemple de tests XMLUnit

En terme de manipulation de documents XML, nous allons présenter 3 solutions : DOM, SAX et Castor.

- SAX (Simple API for XML) : http://www.saxproject.org/

  SAX est une API java événementielle permettant de parcourir un document XML. Un avantage de cette solution est que SAX ne construit pas un arbre, correspondant au document XML, en mémoire. Ainsi, l'utilisation de la mémoire et la rapidité du parsing sont indépendantes de la taille physique du document XML. A l'inverse, SAX étant événementiel, cette solution n'est pas adaptée pour effectuer des accès aléatoires à l'intérieur du document XML. Pour utiliser SAX en JAVA, il faut définir une classe étendant la classe *ContentHandler* définie par SAX. Cette dernière est un gestionnaire (handler) d'événements, gérant les événements tels que le début et la fin d'un document XML, le début et la fin d'un élément, et la présence d'un flux de caractères dans un élément. L'exemple de la Figure 32 illustre la définition de ce gestionnaire :

```
public class MyHandler extends ContentHandler
{
   //détection d'ouverture de balise
   public void startElement(String uri, String localName,
                            String qName, Attributes attributes) throws
SAXException
```

```
    {
      //Traitements du début d'un élément
    }
    //détection fin de balise
    public void endElement(String uri, String localName, String qName)
                throws SAXException
    {
      //Traitements de la fin d'un élément
    }
    //détection de caractères
    public void characters(char[] ch,int start, int length)throws SAXException
    {
      //Traitements d'une suite de caractères
    }
    //début du parsing
    public void startDocument() throws SAXException
    {
      //Traitements du début d'un document XML
    }
    //fin du parsing
    public void endDocument() throws SAXException
    {
      //Traitements de la d'un document XML
    }
}
```

Figure 32 – Exemple de Handler SAX

Pour appeler cet handler cela se fait de la manière suivante en JAVA :

```
SAXParserFactory fabrique = SAXParserFactory.newInstance();
SAXParser parseur = fabrique.newSAXParser();
File fichier = new File("./exemple.xml");
ContentHandler handler = new MyHandler();
parseur.parse(fichier, handler);
```

Figure 33 – Exemple d'appel d'un parseur SAX

SAX est donc à utiliser dans des cas bien précis, tels que pour la lecture d'un document volumineux ou pour un accès séquentiel.

- DOM (Document Object Model)

A l'inverse de SAX, DOM est une API qui réalise une représentation en mémoire sous la forme d'un arbre d'un document XML. DOM a donc l'avantage de proposer une structure d'arbre qui permet naturellement de naviguer dans un document XML, que ce soit en lecture ou en écriture. A partir de cet arbre, il est possible d'accéder aléatoirement au contenu du document XML.
Voici un exemple de parcours d'un document XML avec DOM :

```
DOMParser parser = new DOMParser();
parser.parse(filename);
Document doc = parser.getDocument();
//Obtenir la racine du document
Element root = (Element)doc.getDocumentElement();
// Démarrer le parcours à partir de la racine
parcoursArbre (root) ;
public static void parcoursArbre(Node node)
{
        // afficher le nom et la valeur contenu dans le nœud courant
        System.out.println(node.getNodeName()+":"+node.getNodeValue());

        // Parcours des fils si le nœud en possède
        if (node.hasChildNodes())
        {
                NodeList nodeList = node.getChildNodes();
                int size = nodeList.getLength();
                for (int i = 0; i < size; i++)
                        parcoursArbre (nodeList.item(i));
        }
  }
```

Figure 34 – Exemple de parcours d'un document XML avec DOM

L'inconvénient de cette représentation est la quantité potentielle de mémoire à réserver pour un document XML de taille importante.

- Castor : http://www.castor.org

Castor est un outil permettant de sérialiser et de désérialiser des documents XML en objets JAVA. Pour ceci faire, Castor propose un compilateur traduisant des schémas XML en classes JAVA. Ce compilateur est utilisable en ligne de commande (pour plus d'informations cf. : http://java.sun.com/webservices/docs/1.6/tutorial/doc/index.html).
Castor possède donc un avantage par rapport à des technologies telles que SAX et JDOM (http://www.jdom.org). En effet, après compilation du schéma XML, le compilateur définit des classes et interfaces JAVA permettant de manipuler, en lecture et écriture, l'arbre qui sera créé à partir d'un document XML. Castor offre la possibilité de généraliser et d'automatiser la compilation de schémas à l'aide de script ANT. Il est possible de définir un document XML définissant des bindings entre des nœuds XML et des classes JAVA afin d'éviter d'éventuels conflits de nom.

```
<binding>
<elementBinding name = « Data/Purchase/Item »>
<java-class name = « PurchaseItem »/>
        </elementBinding>
        <elementBinding name = « Data/Order/Item »>
<java-class name = « PurchaseOrderItem »/>
        </elementBinding>
</binding>
```

Figure 35 – Exemple de binding avec Castor

Castor est un compromis entre SAX et DOM. En effet, Castor utilise la technologie SAX pour parcourir le schéma XML à compiler et utilise DOM afin de pouvoir naviguer dans

les objets JAVA représentant un document XML. Après un court état de l'art, nous avons utilisé ces technologies afin de réaliser un framework de tests pour des documents XML.

2) Développement d'un outil de test XML

L'approche qui a été abordée est la combinaison des technologies présentées précédemment. Des tests sont définis dans un document XML. La structure de ce fichier s'inspire du document metadata du framework TestSuite W3C. Quelques fonctionnalités sont reprises du TestSuite W3C, et les fonctionnalités de XMLUnit sont utilisées.
Ci-dessous la structure du document définissant un jeu de tests (Figure 36) et un exemple de jeu de tests (Figure 37) :

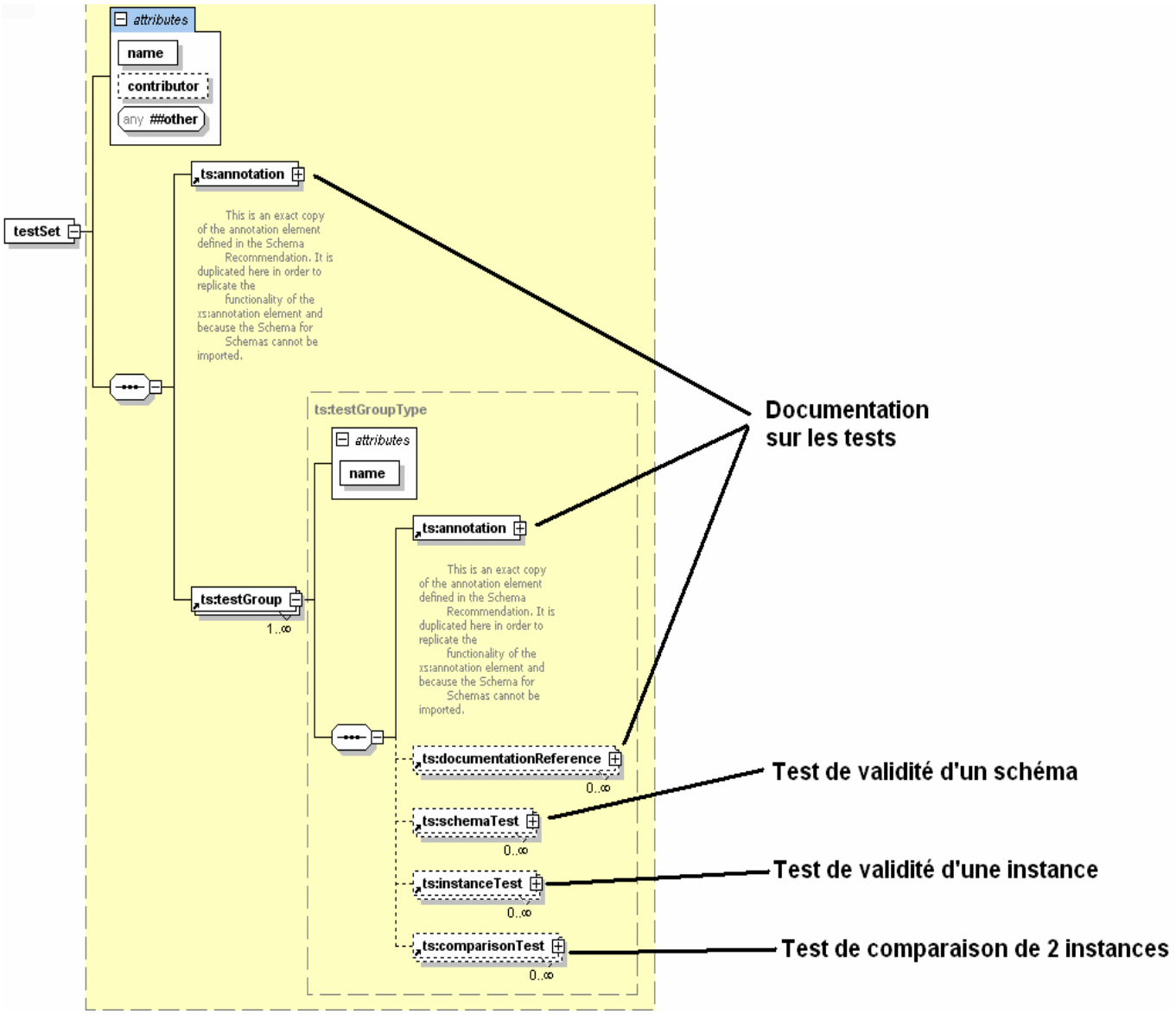


Figure 36 – Structure du schéma de tests

```
<testSet contributor="Moi" name="nom série de tests" xmlns="TestSuite"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="TestSuite AnnotatedTSSchema.xsd"
xmlns:xlink="http://www.w3.org/1999/xlink">
	<annotation>
		<documentation>première série de test</documentation>
	</annotation>
	<testGroup name="test tutorialCar">
		<annotation>
			<documentation>Test des schemas et instances du tutorial Car</documentation>
		</annotation>
		<schemaTest name="tutorialCarParts">
			<schemaDocument path="./schemasDeTest/tutorialCarParts.xsd"/>
			<expected validity="valid"/>
		</schemaTest>
	</testGroup>
	<testGroup name="test02">
		<annotation>
			<documentation>docs 02</documentation>
		</annotation>
		<schemaTest name="schema002">
			<schemaDocument path="./schema2.xsd"/>
			<expected validity="valid"/>
		</schemaTest>
		<instanceTest name="tutoriel_voitures">
			<instanceDocument path="./schemasDeTest/tutoriel_voitures.xml"/>
			<expected validity="valid"/>
		</instanceTest>
	</testGroup>
	<testGroup name="test03">
		<annotation>
			<documentation>test d'une instance non valide</documentation>
		</annotation>
		<instanceTest name="tutoriel_voitures_nonvalide">
			<instanceDocument path="./tutoriel_voitures_nonvalide.xml"/>
			<expected validity="valid"/>
		</instanceTest>
	</testGroup>
	<testGroup name="test04">
		<annotation>
			<documentation>test de comparaison de 2 instances</documentation>
		</annotation>
		<comparisonTest name="test1 de comparaison de 2 instance">
			<instanceDocument path="./tutoriel_voitures.xml"/>
			<instanceDocument path="./tutoriel_voitures_nonvalide.xml"/>
			<expectedComparison value="inequals"/>
		</comparisonTest>
	</testGroup>
</testSet>
```

Figure 37 – Exemple de définition d'un jeu de tests

L'outil de test XML a été réalisé sous la forme d'un plugin Eclipse (Environnement de développement JAVA) et propose les fonctionnalités suivantes :

- Action dans la toolbar
- Action dans le menu popup associée à un fichier *.testSet.xml
- Validation du fichier définissant le jeu de tests
- Validation de schémas XSD
- Validation d'instances XML
- Comparaison d'instances XML (similarités, différences, égalité, inégalité)
- Gestion des chemins relatifs et absolus dans la localisation des documents à tester

- Rapport des résultats dans un fichier XML
- Présentation des résultats à l'aide d'une feuille de style XSL
- Log d'exécution des tests
- Rapport sur les différences entre les instances

Remarque : Le site internet http://www-igm.univ-mlv.fr/~dr/XPOSE2004/vforel/plugeclipse.html propose des explications concernant le développement d'un plugin Eclipse.

Le fonctionnement général du plugin (Figure 38) est le suivant :

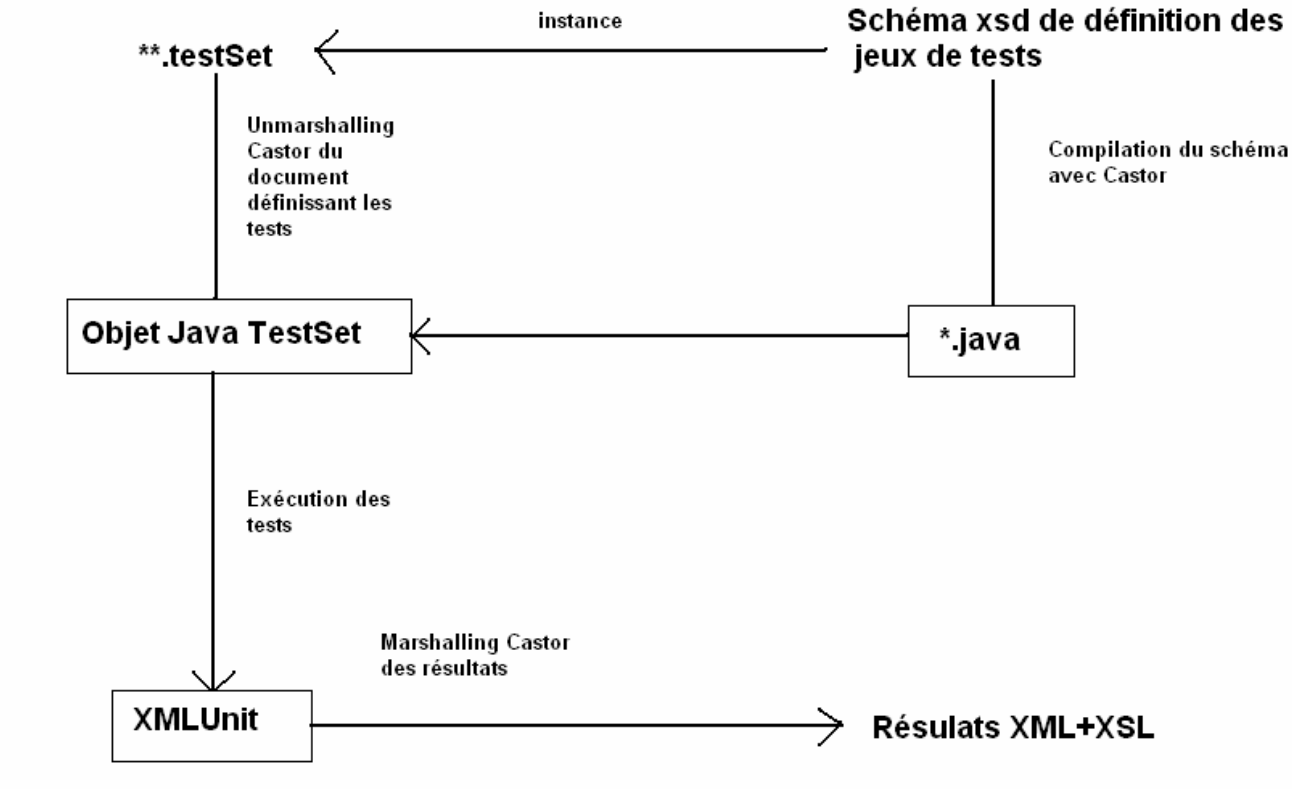


Figure 38 – Schéma du fonctionnement du plugin de test XML

Dans un premier temps le schema XML, définissant la structure des jeux de tests, est compilé à l'aide de Castor pour obtenir des classes Java qui permettront de manipuler d'une manière simple les instances de définition de tests.

Remarque : à chaque modification du schéma XML du fichier de test, les classes Java doivent être regénérées afin de prendre en comptes les changements apportés au schéma.

A partir des classes Java générées, le document XML spécifiant les tests peut être chargé en mémoire et peut être parcouru dans le but d'exécuter les tests définis. A la fin du déroulement des tests, un rapport d'exécution est généré dans lequel figure :

- Des informations générales sur le jeu de test qui a été exécuté
- Pour chaque test, les informations associées telles que le résultat attendu, le résultat obtenu, le lien vers le document testé et un lien vers un rapport présentant les différences entre deux instances dans le cas d'un test de comparaison d'instances.

Les résultats peuvent être les suivants : échoué, réussi, échoué – fichier non trouvé, instances égales, instances inégales, instances similaires.
Les figures suivantes illustrent l'utilisation de cet outil dans éclipse :

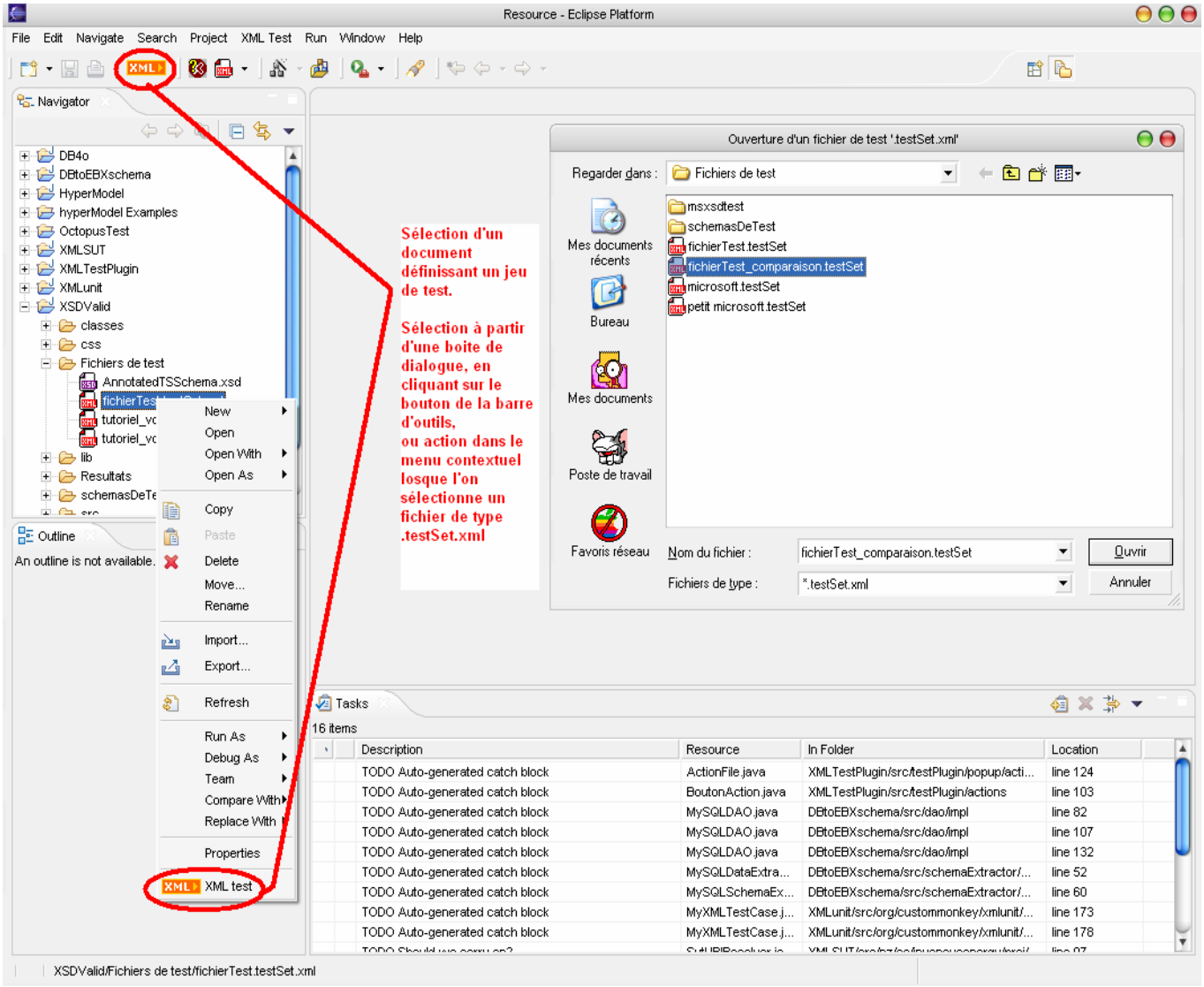


Figure 39 – Sélection d'un fichier définissant un jeu de tests

Figure 40 – Sélection du répertoire de destination des résultats

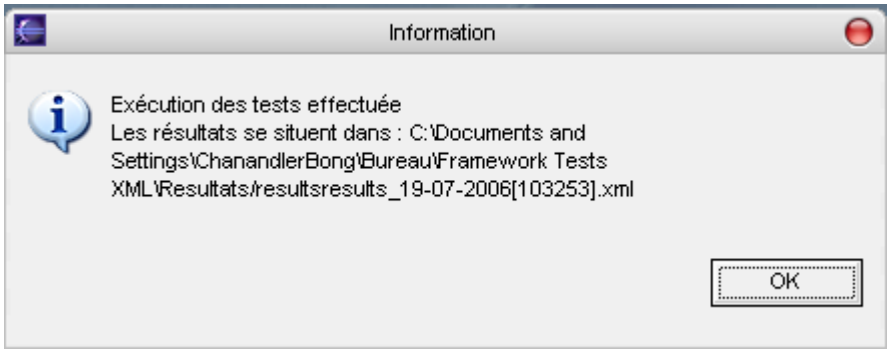


Figure 41 – Message de confirmation de fin d'exécution du jeu de tests

Après exécution du jeu de tests, une fenêtre présentant les résultats est affichée dans Eclipse.

Figure 42 – Résultats d'exécution d'un jeu de tests

L'outil de test réalisé présente des avantages par rapport à des frameworks de tests existants. En effet, dans cet outil, la notion de comparaison d'instances a été introduite, par l'intermédiaire d'XMLunit. De plus, l'exécution d'un jeu de tests donne lieu à un rapport complet dans lequel figure les résultats et dans le cas d'un test de comparaison d'instances, si ces dernières sont différentes, un rapport présentant les différences est accessible par un lien.
Il est aussi possible d'étendre, de manière très simple, les types de tests en modifiant le métaschéma du jeu de tests et en le recompilant via Castor.

Nous avons vu jusqu'à présent que nous pouvions apporter des spécificités objets aux modèles d'adaptation et qu'il était possible de mieux assurer leur création, par l'intermédiaire d'un profil UML. EBX.Platform s'inscrivant dans des solutions de MDM, les problématiques d'intégration de données provenant de sources hétérogènes se posent.

V. Intégration de données provenant de sources hétérogènes via XML

La majorité des entreprises exploite des bases de données en vue de stocker les informations de leurs systèmes. En choisissant d'utiliser EBX.Platform, l'entreprise aura la possibilité d'unifier la gestion de ses données via un modèle d'adaptation. Cependant, la définition de ce modèle d'adaptation doit être réalisée par un expert du domaine, ce qui a pour conséquence d'entraîner un certain coût (financier et temps). Nous proposons donc de définir un extracteur de modèle d'adaptation pour bases de données.

A. Extraction de schémas à partir de bases de données hétérogènes

La définition d'un extracteur de schéma générique pour base de données se réalise par la définition d'une interface Java décrivant les méthodes nécessaires au rapatriement de métadonnées. Le diagramme de classes de la Figure 43 illustre la définition de l'interface décrivant les méthodes à implémenter pour dialoguer avec une base de données, et ses implémentations pour les bases MySQL et Oracle :

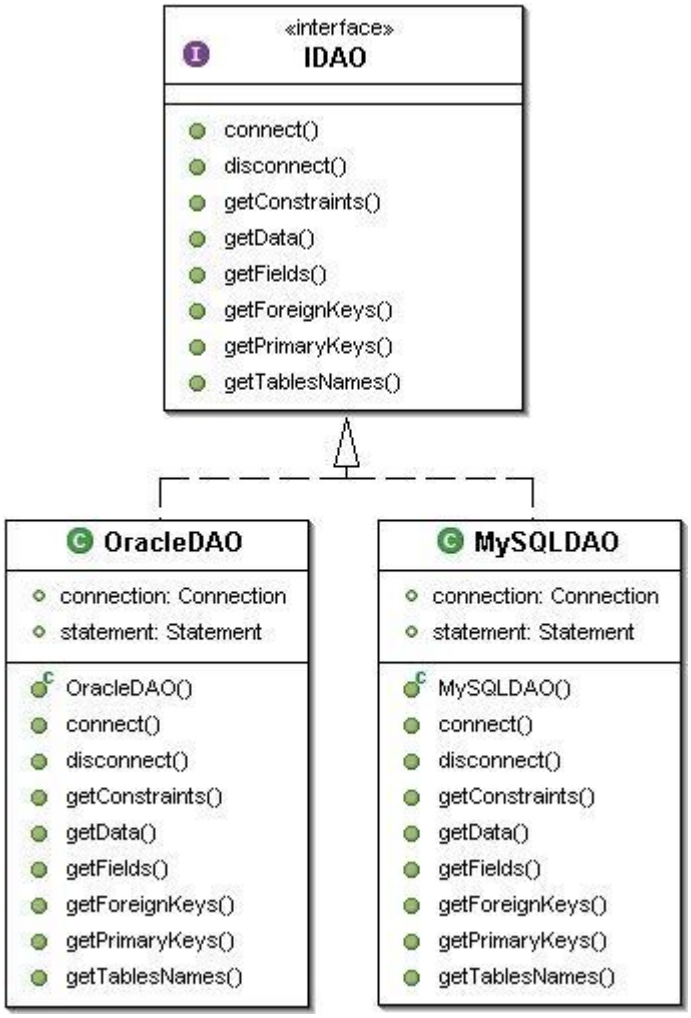


Figure 43 – Diagramme de classes de l'interface d'extraction de schémas et de données, ainsi que ses implémentations MySQL et Oracle

Nous devons préciser qu'afin de pouvoir extraire le schéma d'une base de données, celle-ci doit disposer d'un catalogue système décrivant la struture de la base, d'une DTD ou un schéma XSD dans le cas de bases de données XML. Pour la base de données MySQL, l'implémentation a été réalisée pour la version 5.xx, faute de disposer d'un catalogue système assez riche dans les versions antérieures.
L'interface IDAO défini un moyen d'accès vers une base de données, en décrivant les méthodes suivantes :

- *Connect*, réalise une connexion vers une base de données.
- *Disconnect*, réalise une déconnexion de la base.
- *GetTablesNames*, extrait les noms des tables composant la base.
- *GetConstraints*, extrait les contraintes sur les tables et champs.
- *GetFields*, extrait les champs d'une table données.
- *GetPrimaryKeys*, extrait les clés primaires d'une table donnée.
- *GetForeignKeys*, extrait les clés étrangères d'une table donnée.
- *GetData*, extrait les données d'une table.

Cette implémentation étant réalisée pour une base de données précise, celle-ci peut être utilisée pour extraire le schéma et le transformer en modèle d'adaptation. Il en va de même pour l'extraction de données qui donnera lieu à la création d'un document XML. Le diagramme de classes de la Figure 44 illustre l'architecture de l'implémentation MySQL d'extraction de schéma et de données et de la transformation respectivement en un modèle d'adaptation et en document XML :

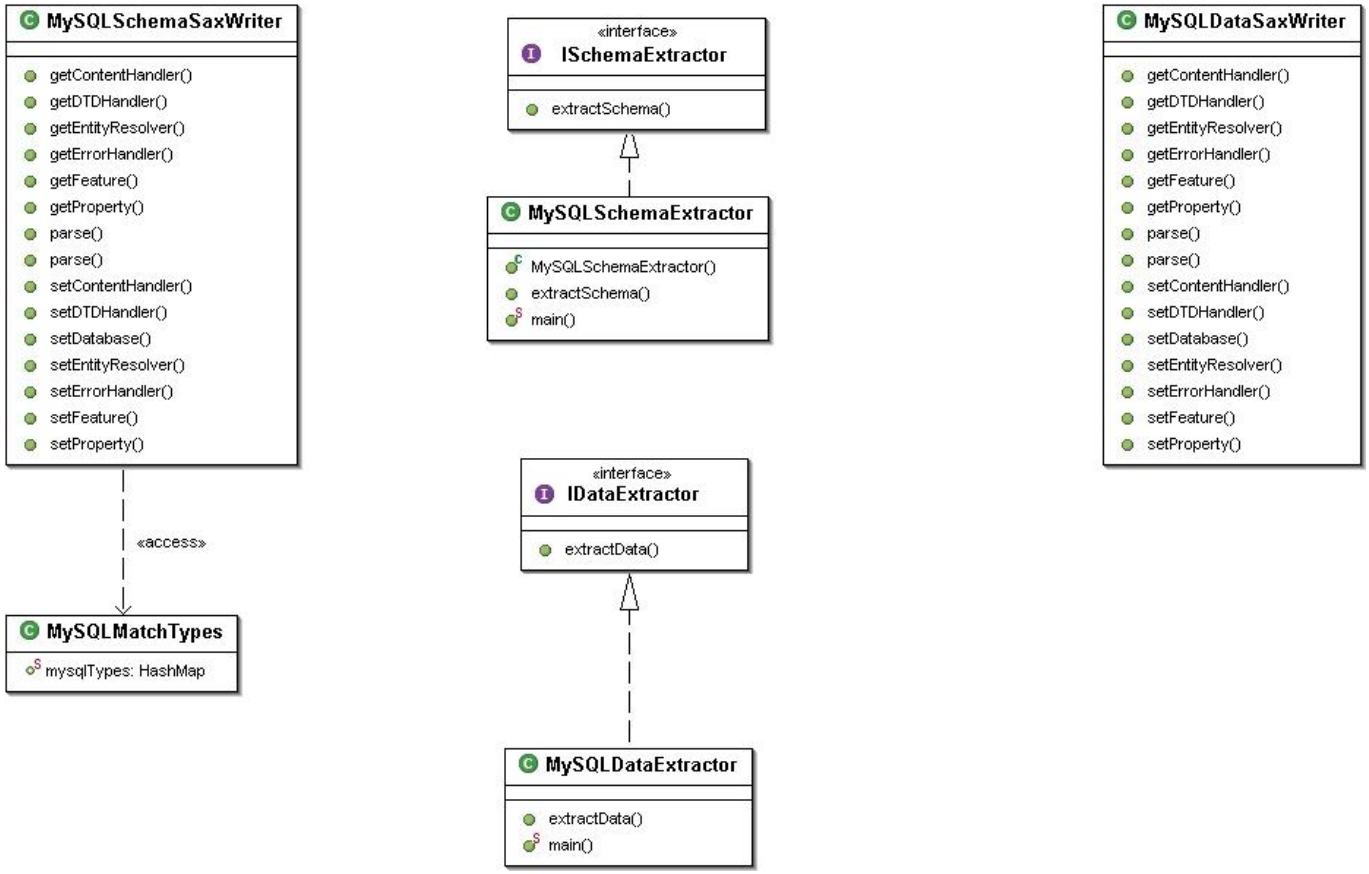


Figure 44 – Diagramme de classes de l'implémentation MySQL d'extraction de schéma de données.

En définissant une interface commune décrivant les méthodes à implémenter, il est très simple d'étendre la « boîte à outils » d'extraction vers d'autres bases de données. Nous pouvons aussi préciser que dans le processus de transformation d'un schéma d'une base de données en un modèle d'adaptation défini en XML Schema, un binding de types est réalisé afin de respecter les normes du W3C.

Afin de présenter un exemple d'extraction d'un schéma à partir d'une base de données, nous allons utiliser la base de données de test, *Mondial* mise à disposition par Peter McBrien (http://www.doc.ic.ac.uk/~pjm/databases/index.html), chercheur à *l'Imperial College* de Londre.

Dans le modèle entités-relations, cette base est composée de 14 entités et 22 relations, et dans le modèle relation, elle est composée de 27 tables.
Ci-dessous, la Figure 45 présente le modèle entités-relations de la base *Mondial*:

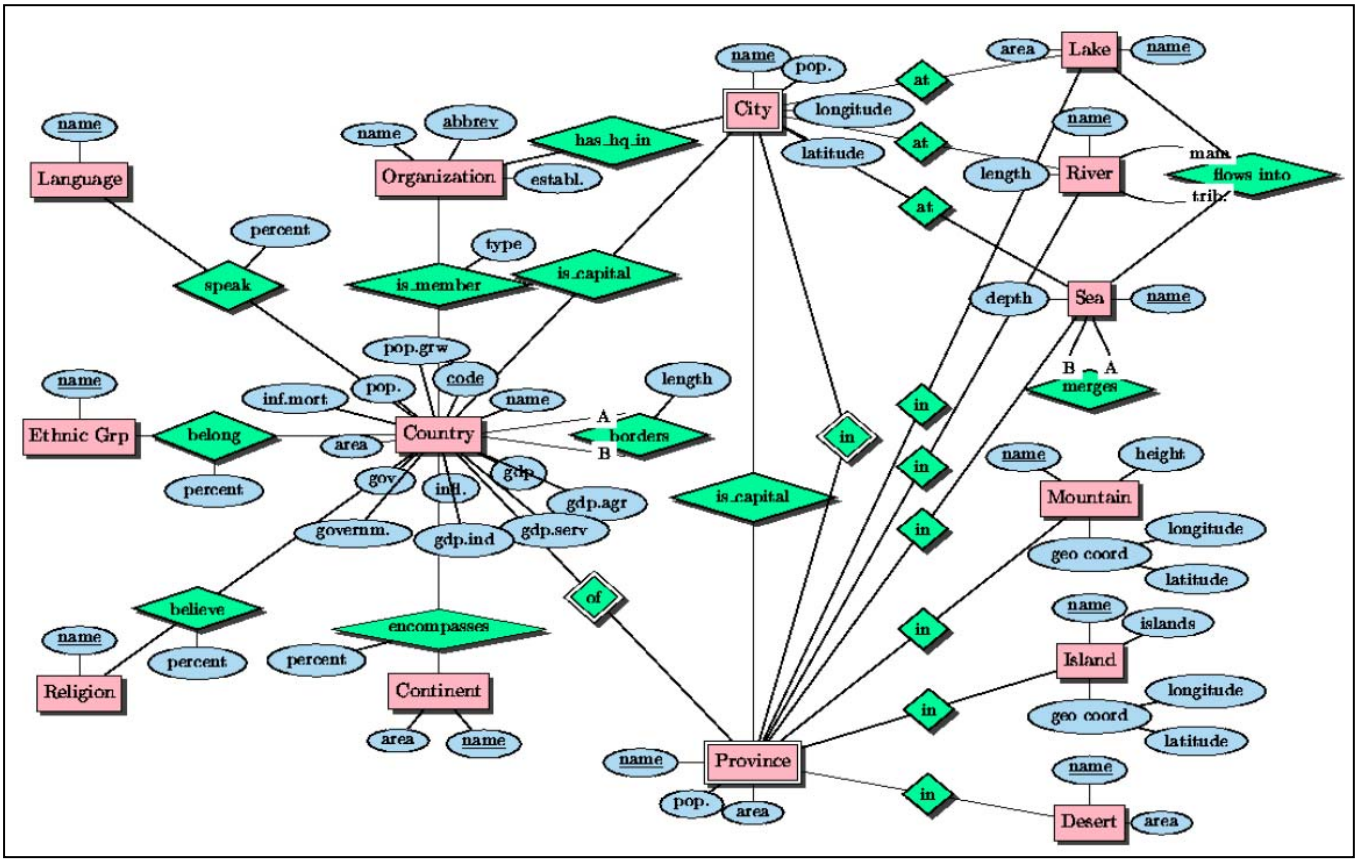


Figure 45 – Schéma entités-relations de la base Mondial définie par McBrien

Après importation de la base *Mondial* dans un système de gestion de base de données, nous allons procéder à son exploitation. L'exploitation de cette base s'effectue à l'aide de l'interface suivante :

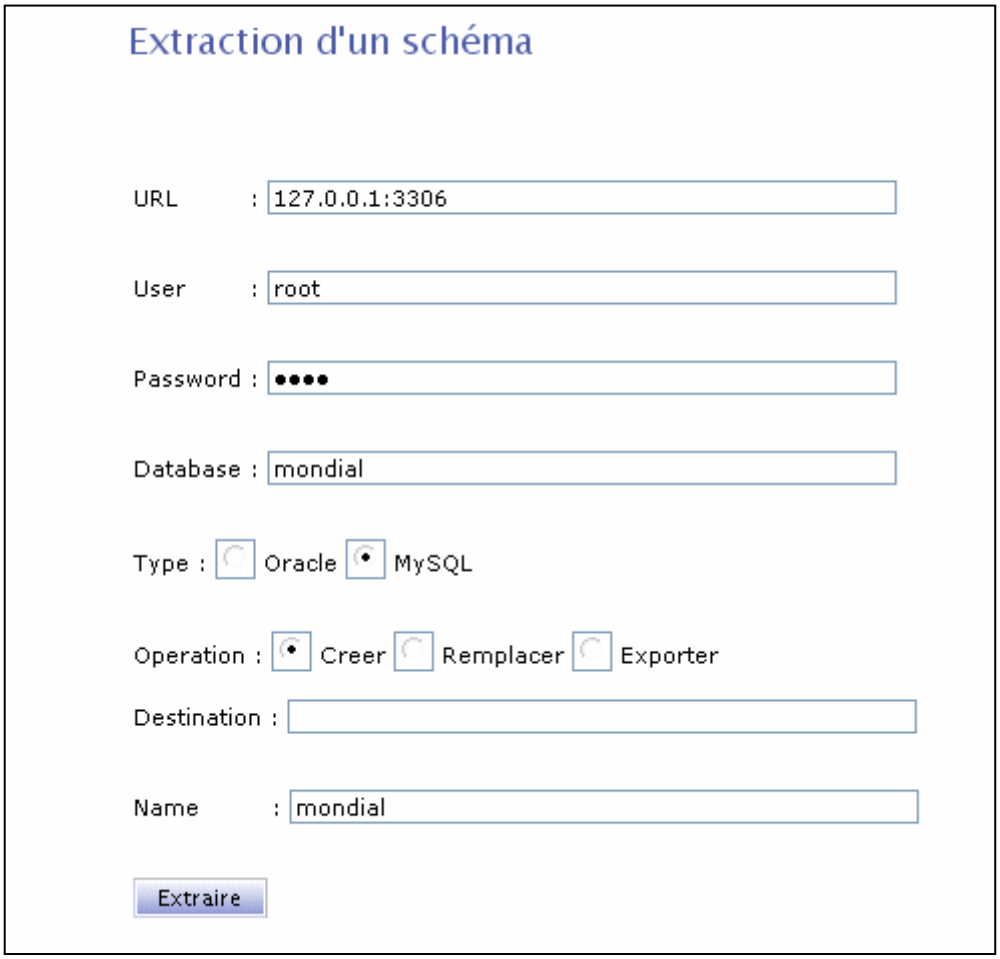


Figure 46 – Interface d'extraction d'un schéma

Cette interface permet de saisir les paramètres de connexion à la base, de choisir le type de la base de données et le mode d'extraction (s'il s'agit d'extraire le schéma simplement dans un document XML Schema, ou alors de créer l'adaptation dans EBX.Platform).

Après extraction, nous obtenons le modèle d'adaptation suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:fmt="urn:ebx-schemas:format_1.0" xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:ebxbnd="urn:ebx-schemas:binding_1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="mondial" osd:access="--">
<xs:complexType>
<xs:sequence>
<xs:element name="borders" type="bordersType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="city" type="cityType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="continent" type="continentType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="country" type="countryType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="desert" type="desertType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="economy" type="economyType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="encompasses" type="encompassesType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="ethnic_group" type="ethnic_groupType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="geo_desert" type="geo_desertType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="geo_island" type="geo_islandType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="geo_lake" type="geo_lakeType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="geo_mountain" type="geo_mountainType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="geo_river" type="geo_riverType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="geo_sea" type="geo_seaType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="is_member" type="is_memberType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="island" type="islandType" minOccurs="0" maxOccurs="unbounded"/>
```

```
<xs:element name="lake" type="lakeType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="language" type="languageType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="located" type="locatedType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="merges_with" type="merges_withType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="mountain" type="mountainType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="organization" type="organizationType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="politics" type="politicsType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="population" type="populationType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="province" type="provinceType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="religion" type="religionType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="river" type="riverType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="sea" type="seaType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name="cityType">
        <xs:annotation>
        <xs:appinfo>
        <osd:table>
        <id>pk_city</id>
        <primaryKeys> /Name /Country /Province</primaryKeys>
        </osd:table>
        </xs:appinfo>
        </xs:annotation>
        <xs:sequence>
                <xs:element name="Name" type="xs:string"/>
                <xs:element name="Country" type="xs:string" />
                <xs:element name="Province" type="xs:string"/>
                <xs:element name="Population" type="xs:integer"/>
                <xs:element name="Longitude" type="xs:decimal"/>
                <xs:element name="Latitude" type =xs:decimal"/>
        </xs:sequence>
</xs:complexType>
Etc…..
```

Figure 47 – Extrait du modèle d'adaptation de la base Mondial

L'extracteur présenté précédemment permet de créer un modèle d'adaptation, automatiquement, à partir d'une base de données. Sur le même principe, nous pouvons extraire des données et les importer dans EBX.Platform.

B. Extraction et import de données

Dans le cadre d'EBX.Platform, deux cas d'import de données existent :

- Import de données à partir d'un fichier XML, très complexe à n niveaux. Cet import doit s'appuyer sur le modèle d'adaptation afin de pouvoir gérer tous les niveaux de données.
  L'exemple de la Figure 48 illustre la complexité d'un tel document.

```
<root xmlns:xml="http://www.w3.org/XML/1998/namespace">
        <testDataTypes>
                <simpleTypes> Nœud complexe contenant des types simples
                        <string>poregperj</string>
                        <boolean>true</boolean>
                        <decimal>10.23</decimal>
                        <dateTime>2006-05-17T13:35:00.000</dateTime>
                </simpleTypes>
                <veryComplex>  Nœud complexe contenant d'autres nœuds complexes
                        <beanTerminal>
                                <simpleInt>1</simpleInt>
                                <simpleString>s1</simpleString>
                        </beanTerminal>
                        <beanNotTerminal>
                                <simpleInt>2</simpleInt>
                                <simpleString>s2</simpleString>
                        </beanNotTerminal>
                        <complexList>
                                <complexInTerminal>
                                        <int1>1</int1>
                                        <string2>s1</string2>
                                </complexInTerminal>
                                <complexInNotTerminal>
                                        <int1>2</int1>
                                        <string2>s2</string2>
                                </complexInNotTerminal>
                        </complexList>
                </veryComplex>
                <complexTable> Nœud complexe contenant des types simples et une liste d'entier
                        <col1>1</col1>
                        <col2>2006-05-17</col2>
                        <col3>fzefzef</col3>
                        <col4>1</col4>          Liste d'entier
                        <col4>2</col4>
                        <col4>3</col4>
                </complexTable>

        </testDataTypes>
</root>
```

Figure 48 – Exemple de documents XML à n niveaux

- Import de données direct à partir d'une base de données, telles que Oracle et MySQL (via l'extracteur de données défini précédemment). Cet import est plus simple à gérer que le premier. En effet, dans le cadre de bases de données relationnelles il peut y avoir au maximun trois niveaux de données dans le document XML :
    - Niveau 0 = racine
    - Niveau 1 = table
    - Niveau 2 = champs d'une occurrence

  Ci-dessous un exemple de document XML à trois niveaux.

```
<Mondial>   Niveau 0
        <city>  Niveau 1
                <Name>Aachen</Name> Niveau 2
                <Country>D</Country>
                <Province>Nordrhein Westfalen</Province>
                <Population>247113</Population>
                <Longitude> </Longitude>
```

```
            <Latitude> </Latitude>
        </city>
        <city> Niveau 1
            <Name>Aalborg</Name> Niveau 2
            <Country>DK</Country>
            <Province>Denmark</Province>
            <Population>113865</Population>
            <Longitude>10</Longitude>
            <Latitude>57</Latitude>
        </city>
        …..
</Mondial>
```

Figure 49 – Exemple de documents XML à 3 niveaux

Nous pouvons résumer les procédures d'import de données à l'aide du schéma de la Figure 50 :

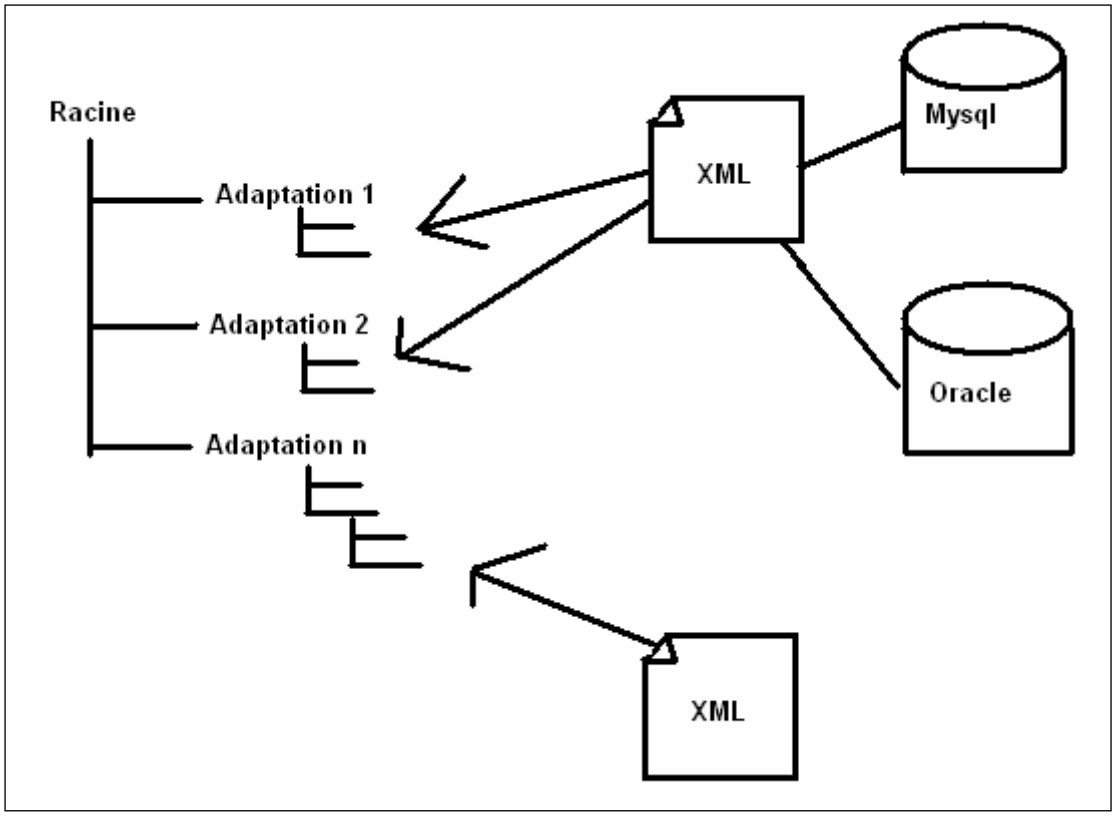


Figure 50 – Schéma d'import de données

Les mécanismes de persistance, d'import et d'export de données reposaient sur la technologie JDOM. Un des problèmes les plus importants était la consommation mémoire lors de la manipulation de documents XML de volume important. Pour remédier à ce problème, JDOM a été remplacé par SAX, accroissant de cette manière le nombre maximal d'occurrences supportées par adaptation (100 000 avec JDOM, 900 000 avec SAX) mais aussi les temps de création, suppression, écrasement et validation d'occurrences.
Le tableau comparatif suivant illustre les améliorations de performances :

	fileSystem SAX	fileSystem JDOM		hsqldb SAX	hsqldb JDOM
	ms	ms		ms	ms
Create 100 occurrences	78	78		31	109
Release 100 occurrences	47	47		46	63
Overwrite 100 occurrences	63	47		47	47
Delete 100 occurrences	47	31		0	15
Create 1000 occurrences	172	203		141	203
Release 1000 occurrences	110	188		141	203
Overwrite 1000 occurrences	125	140		109	156
Delete 1000 occurrences	47	47		16	16
Create 10000 occurrences	968	1500		1375	1516
Release 10000 occurrences	781	1515		1172	1750
Overwrite 10000 occurrences	844	1125		1375	1375
Delete 10000 occurrences	875	891		813	844
Create 100000 occurrences	19813	28281		20796	23859
Release 100000 occurrences	7328	15016		10860	15765
Overwrite 100000 occurrences	17406	23906		19109	22375
Delete 100000 occurrences	132500	139375		122094	123375
loading 200 000 occurrences	35860	36218		9891	12250
commit 200 003 objects	3687	8765		5645	8359

Tableau 1 - Comparatif persistance SAX / JDOM

Ce tableau illustre le gain en temps d'exécution d'opérations effectuées par EBX.Platform avec SAX.
Le tableau suivant met en évident les temps d'import et d'export, avec SAX, de grandes quantités d'occurrences.

	FileSystem	HSQL
20 000 occurrences		
Import	2 s	3 s
Export	1 s	1s
110 000 occurrences		
Import	29 s	29s
Export	2 s	1s
510 000 occurrences		
Import	6m16	6m19
Export	11s	13s

Tableau 2 - Performances import / export sur d'importants volumes de données

Conclusion et perspectives

EBX.Platform présente une solution optimale de Master Data Management. Nous avons vu, tout au long de ce document que nous pouvons solutionner (en théorie et en pratique) certains problèmes liés au Master Data Management, tel que l'intégration de données provenant de sources hétérogènes.

Nous avons vu que tout modèle EBX est un document conforme à la norme XML Schema. Ce principe peut être précisé de la manière suivante :

- Seul un sous-ensemble du langage XML Schema est implémenté par EBX. En effet, certaines caractéristiques de XML Schema sont difficiles à appréhender pour le modélisateur et elles n'apporteraient pas de valeur ajoutée significative à EBX.Platform.
- Les possibilités d'extensions explicitement prévues par XML Schema sont utilisées (par exemple pour les contraintes dynamiques ou les libellés d'énumérations).

Il découle de cette situation les besoins suivants :

- La modélisation et la validation des modèles EBX devraient être mieux outillées. Certes des outils XML Schema existent, mais, d'une part, le modélisateur peut utiliser des caractéristiques de XML Schema qui ne sont pas implémentées par EBX et, d'autre part, il n'est pas guidé en ce qui concerne les extensions EBX (il doit alors passer en mode texte XML verbeux et non assisté). Enfin, l'outil tiers n'effectue qu'une validation très partielle. L'utilisation des profils UML, comme nous l'avons présenté, peut être une solution à ce problème.
- Un certain nombre d'éléments de langage (extensions XML Schema) restent à définir : expression de contraintes sémantiques dans des structures de données complexes ; mapping conceptuel vis à vis des sources de données de façon d'une part à améliorer la qualité des données, d'autre part à exploiter au mieux l'information pour la documentation, et enfin pour assurer l'automatisation des imports et exports de/vers le système d'information existant.
- Enfin, EBX.Platform ne disposant pas d'un métamodèle, il n'est pas possible de disposer du cycle de vie d'un schéma.

La suite de ce travail devrait être, dans un premier temps, la formalisation d'un métamodèle pour le module Master Data Management permettant d'exprimer et de valider des contraintes sémantiques en fonction de profils (par exemple, langage métier).
La définition de ce métamodèle pourrait se subdiviser en deux parties :

- Méthodes de modélisation et expression de contraintes (*expressions de facettes*) : Création d'un formalisme type UML permettant de modéliser un schéma EBX. Cette modélisation s'appuyant sur des règles, le schéma créé est rendu valide.
- Inclusion de l'expression de contraintes selon des profils (contraintes sur les types ou contraintes entre concepts) apportant une sémantique aux concepts représentés, expression des dépendances.

Ce métamodèle prendra en compte :

- les spécificités de la norme ODMG, spécificités que nous avons abordées dans ce document, au formalisme EBX : notion d'héritage *au niveau des modèles* (directement spécifiée dans le schéma), spécialisation, généralisation, dépendance… **[2]**
- les avantages liés au langage OWL dédié à la définition d'ontologie **[10] [11] [12] [13]**

- le formalisme UML **[6]**
- les avantages des graphes conceptuels pour l'expression de relations et donc de contraintes entre concepts. **[14]**

Annexes

A.1 Modèle d’adaptation utilisant les relations d’héritage et de composition

A.2 Métaschéma d’un modèle d’adaptation

A.3 Modèle d’adaptation en XML présentant les types de base d’EBX.Platform

A.1 Modèle d'adaptation utilisant les relations d'héritage et de composition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="root" osd:access="--">
	<xs:complexType>
		<xs:sequence>
		<xs:element name="personnes" type="Personne" minOccurs="0" maxOccurs="unbounded"/>
		<xs:element name="etudiants" type="Etudiant" minOccurs="0" maxOccurs="unbounded"/>
		<xs:element name="enseignants" type="Enseignants" minOccurs="0" maxOccurs="unbounded"/>
		<xs:element name="universites" type="Universite" minOccurs="0" maxOccurs="unbounded"/>
		<xs:element name="departements" type="Departement" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	</xs:element>

	<!--
		Définition du concept Personne
		-->
	<xs:complexType name="Personne">
	<xs:annotation>
		<xs:appinfo>
			<!--
			Définition de la table
			-->
			<osd:table>
				<id>id_personne</id>
				<primaryKeys>/nom /prenom /date_naissance</primaryKeys>
				</osd:table>
				<!--
				Définition de la généralisation
				-->
				<osd:generalisation>
					<osd:conceptPath>/Root/etudiants /Root/enseignants</osd:conceptPath>
				</osd:generalisation>
		</xs:appinfo>
	</xs:annotation>
	<!--
	Attributs
	-->
	<xs:sequence>
		<xs:element name="nom" type="xs:string"/>
		<xs:element name="prenom" type="xs:string"/>
		<xs:element name="date_naissance" type="xs:string"/>
	</xs:sequence>
	</xs:complexType>


	<!--
	Définition du concept Etudiant
	-->
	<xs:complexType name="Etudiant">
	<xs:annotation>
		<xs:appinfo>
		<!--
		Définition de la table
		-->
		<osd:table>
			<id>id_etudiant</id>
			<primaryKeys>/num_etudiant</primaryKeys>
		</osd:table>
		<!--
		Définition de la spécialisation
		-->
		<osd:specialisation>
```

```
                    <osd:conceptPath>/Root/Personne </osd:conceptPath>
            </osd:specialisation>
            </xs:appinfo>
</xs:annotation>
<!--
Attributs
-->
<xs:sequence>
            <xs:element name="num_etudiant" type="xs:string"/>
            <xs:element name="annee_inscription" type="xs:date"/>
</xs:sequence>
</xs:complexType>

<!--
Définition du concept Enseignant
-->
<xs:complexType name="Enseignant">
            <xs:annotation>
                        <xs:appinfo>
                        <!--
                        Définition de la table
                        -->
                        <osd:table>
                                    <id>id_pers</id>
                                    <primaryKeys>/num_enseignant</primaryKeys>
                        </osd:table>
                        <!--
                        Définition de la spécialisation
                        -->
                        <osd:specialization>
                                    <osd:conceptPath>/Root/Personne </osd:conceptPath>
                        </osd:specialization>
                        </xs:appinfo>
            </xs:annotation>
            <!--
            Attributs
            -->
            <xs:sequence>
                        <xs:element name="num_enseignant" type="xs:string"/>
                        <xs:element name="salaire" type="xs:float"/>
                        <xs:element name="email" type="osd:email"/>
                        <xs:element name="num_bureau" type="xs:string"/>
            </xs:sequence>
</xs:complexType>

<!--
Définition du concept Université
-->
<xs:complexType name="Universite">
            <xs:annotation>
                        <xs:appinfo>
                                    <!--
                                    Définition de la table
                                    -->
                        <osd:table>
                                    <id>id_univ</id>
                                    <primaryKeys>/num_universite</primaryKeys>
                        </osd:table>
                        <!--
                        Définition de la composition
                        La composition sera considérée comme une liste de départements
                        -->
                        <osd:composition>
                                    <osd:conceptPath>/Root/Departement </osd:conceptPath>
                        </osd:composition>
                        </xs:appinfo>
            </xs:annotation>
```

```
		<!--
			Attributs
		-->
		<xs:sequence>
			<xs:element name="num_universite" type="xs:string"/>
			<xs:element name="nom_universite" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>

	<!--
		Définition du concept Departement
	-->
	<xs:complexType name="Departement">
	<xs:annotation>
		<xs:appinfo>
		<!--
		Définition de la table
		-->
			<osd:table>
				<id>id_dept</id>
				<primaryKeys>/num_dept</primaryKeys>
			</osd:table>
		</xs:appinfo>
	</xs:annotation>
		<!--
			Attributs
		-->
		<xs:sequence>
			<xs:element name="num_dept" type="xs:string"/>
			<xs:element name="nom_departement" type="xs:string"/>
			<xs:element name="description" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>
</xs:schema>
```

A.2 Métaschéma d'un modèle d'adaptation

- La facette *osd:access* supporte un format à deux caractères permettant de spécifier des droits d'accès sur un noeud. Les valeurs possibles deviennent ainsi : "RW" (lecture et écriture), "CC" (annulation des droits sur les nœuds descendants), "R-" (lecture) et "--" (racine du modèle d'adaptation) :

```
<xs:attribute name="access">
	<xs:simpleType>
		<xs:restriction base="xs:string">
			<xs:enumeration value="--"/>
			<xs:enumeration value="CC"/>
			<xs:enumeration value="R-"/>
			<xs:enumeration value="RW"/>
		</xs:restriction>
	</xs:simpleType>
</xs:attribute>
```

- Définition de tables :

```
<xs:element name="table">
  <xs:complexType>
  <xs:sequence>
    <xs:element name="id" type="xs:string" nillable="true" minOccurs="0"/>
    <xs:element name="primaryKeys" type="xs:string" nillable="false"/>
    <xs:element name="mayCreateDeriving" type="xs:string" default="never" nillable="false" minOccurs="0"/>
    <xs:element name="mayCreateRoot" type="xs:string" default="always" nillable="false" minOccurs="0"/>
    <xs:element name="mayCreateOverwriting" type="xs:string" default="always" nillable="false" minOccurs="0"/>
```

```
        <xs:element name="mayCreateOcculting" type="xs:string" default="always" nillable="false" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

- Définition d'élément permettant de documenter des nœuds :

```
<xs:element name="label" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:attribute name="label" type="xs:string"/>
<xs:attribute name="description" type="xs:string"/>
```

- Définition de contraintes sur des éléments (cf. *facettes étendues* dans la seconde partie de ce document) :

```
<xs:element name="otherFacets">
    <xs:complexType>
    <xs:sequence>
    <xs:element name="minInclusive" nillable="true" minOccurs="0">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="osd:defaultErrorMessage"/>
                    </xs:sequence>
                    <xs:attribute name="path" type="osd:path" use="required"/>
                    <xs:attribute name="index" type="xs:integer" use="optional"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="maxInclusive" nillable="true" minOccurs="0">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="osd:defaultErrorMessage"/>
                    </xs:sequence>
                    <xs:attribute name="path" type="osd:path" use="required"/>
                    <xs:attribute name="index" type="xs:integer" use="optional"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="minExclusive" nillable="true" minOccurs="0">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="osd:defaultErrorMessage"/>
                    </xs:sequence>
                    <xs:attribute name="path" type="osd:path" use="required"/>
                    <xs:attribute name="index" type="xs:integer" use="optional"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="maxEclusive" nillable="true" minOccurs="0">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="osd:defaultErrorMessage"/>
                    </xs:sequence>
                    <xs:attribute name="path" type="osd:path" use="required"/>
                    <xs:attribute name="index" type="xs:integer" use="optional"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="length" nillable="true" minOccurs="0">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="osd:defaultErrorMessage"/>
                    </xs:sequence>
                    <xs:attribute name="path" type="osd:path" use="required"/>
                    <xs:attribute name="index" type="xs:integer" use="optional"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="minLength" nillable="true" minOccurs="0">
```

```
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="osd:defaultErrorMessage"/>
                        </xs:sequence>
                        <xs:attribute name="path" type="osd:path" use="required"/>
                        <xs:attribute name="index" type="xs:integer" use="optional"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="maxLength" nillable="true" minOccurs="0">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="osd:defaultErrorMessage"/>
                        </xs:sequence>
                        <xs:attribute name="path" type="osd:path" use="required"/>
                        <xs:attribute name="index" type="xs:integer" use="optional"/>
                </xs:complexType>
        </xs:element>

<xs:element name="facetOResource" nillable="true" minOccurs="0">
        <xs:complexType>
                <xs:attribute name="modulename" type="xs:string" use="required"/>
                <xs:attribute name="resourceType" use="required">
                <xs:simpleType>
                        <xs:restriction base="xs:string">
                                <xs:enumeration value="ext-images"/>
                                <xs:enumeration value="ext-jscripts"/>
                                <xs:enumeration value="ext-html"/>
                                <xs:enumeration value="ext-stylesheets"/>
                        </xs:restriction>
                </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="relativePath" type="osd:path" use="required"/>
                <xs:attribute name="index" type="xs:integer" use="optional"/>
        </xs:complexType>
</xs:element>
<xs:element name="excludeValue" minOccurs="0">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="osd:defaultErrorMessage"/>
                        </xs:sequence>
                <xs:attribute name="index" type="xs:integer"/>
                <xs:attribute name="value"/>
        </xs:complexType>
</xs:element>
<xs:element name="excludeSegment" minOccurs="0">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="osd:defaultErrorMessage"/>
                </xs:sequence>
                <xs:attribute name="index" type="xs:integer"/>
                <xs:attribute name="minValue" type="xs:integer"/>
                <xs:attribute name="maxValue" type="xs:integer"/>
        </xs:complexType>
</xs:element>

<xs:element name="enumeration" minOccurs="0">
        <xs:complexType>
                <xs:attribute name="path" type="osd:path"/>
        </xs:complexType>
</xs:element>
<xs:element name="nomenclature" minOccurs="0">
        <xs:complexType>
                <xs:attribute name="path" type="osd:path"/>
        </xs:complexType>
</xs:element>

<xs:element name="tableRef" minOccurs="0">
```

```
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="tablePath" type="osd:path" nillable="false"/>
                        <xs:element name="labelPath" type="osd:path" nillable="false" minOccurs="0"/>
                        <xs:element name="displayKey" type="xs:boolean" default="true" nillable="false"
                minOccurs="0"/>
                        <xs:element name="container" type="xs:string" nillable="false" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

<xs:element name="constraint">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="param" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="class" type="xs:string" use="required"/>
        </xs:complexType>
        </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

- Définition de valeurs calculées.

  Par défaut, les valeurs des nœuds d'une adaptation sont lues et persistées dans le référentiel EBX. Néanmoins, il est possible d'alimenter la valeur d'un nœud d'adaptation de manière spécifique. Par exemple une valeur peut provenir d'un système de persistance externe (base de données relationnelle, système central, etc.). Elle peut aussi être le résultat d'un calcul. Ce calcul (ou l'accès à la base) peut même prendre en compte des paramètres de l'adaptation courante :

```
<xs:element name="function">
        <xs:complexType>
                <xs:sequence>
                <xs:element name="param" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="class" type="xs:string" use="required"/>
        </xs:complexType>
</xs:element>
```

- Définition d'un service EBX.
  Un service est une procédure rattachée à un module EBX :

```
<xs:element name="service">
        <xs:complexType>
                <xs:attribute name="ressourcePath" type="osd:path" use="required"/>
                <xs:attribute name="activatedForPaths" type="osd:path" use="optional"/>
        </xs:complexType>
</xs:element>
```

- Définition de messages d'erreur :

```
<xs:element name="defaultErrorMessage" type="xs:string" nillable="false"/>
<xs:element name="mandatoryErrorMessage" type="xs:string" nillable="false"/>
<xs:element name="enumerationErrorMessage" type="xs:string" nillable="false"/>
<!—
Définition de catégories permettant de définir des "filtres" sur des éléments d'un modèle d'adaptation.
-->
<xs:attribute name="category" type="xs:string"/>
```

- Définition de types étendus :

```
<xs:simpleType name="html">
        <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="email">
        <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="path">
        <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="resource">
        <xs:restriction base="xs:anyURI"/>
</xs:simpleType>
<xs:simpleType name="nomenclature">
        <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="currency">
        <xs:restriction base="xs:string">
                <xs:enumeration value="EUR" osd:label="Euro"/>
                <xs:enumeration value="FRF" osd:label="FF"/>
                <xs:enumeration value="USD" osd:label="$"/>
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="locale">
        <xs:restriction base="xs:string">
        <xs:enumeration value="ar" osd:label="Arabic"/>
        <xs:enumeration value="ar_AE" osd:label="Arabic (United Arab Emirates)"/>
        <xs:enumeration value="ar_BH" osd:label="Arabic (Bahrain)"/>
        <xs:enumeration value="ar_DZ" osd:label="Arabic (Algeria)"/>

        Etc…

        </xs:restriction>
</xs:simpleType>
```

A.3 Modèle d'adaptation en XML présentant les types de base d'EBX.Platform

```
<xs:schema
        <xs:element name="root" osd:access="--">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="testDataTypes">
                                        <xs:complexType>
                                                <xs:sequence>
                                                <xs:element name="string" type="xs:string"/>
                                                <xs:element name="boolean" type="xs:boolean"/>
                                                <xs:element name="bigDecimal" type="xs:decimal"/>
                                                <xs:element name="dateTime" type="xs:dateTime"/>
                                                <xs:element name="time" type="xs:time"/>
                                                <xs:element name="date" type="xs:date"/>
                                                <xs:element name="anyuri" type="xs:anyURI"/>
                                                <xs:element name="name" type="xs:Name"/>
                                                <xs:element name="int" type="xs:int"/>
                                                <xs:element name="integer" type="xs:integer"/>
                                                <xs:element name="locale" type="osd:locale"/>
                                                <xs:element name="resource" type="osd:resource">
                                                        <xs:annotation>
                                                                <xs:appinfo>
                                                                        <osd:otherFacets>
                                                                                <osd:FacetOResource
osd:moduleName="wbp" osd:resourceType="ext-images" osd:relativePath="exemple/"/>
                                                                        </osd:otherFacets>
                                                                </xs:appinfo>
                                                        </xs:annotation>
```

```
                                                </xs:element>
                        <xs:element name="email" type="osd:email"/>
                        <xs:element name="html" type="osd:html"/>
                        <xs:element name="bigDecimalList" type="xs:decimal" minOccurs="0" maxOccurs="100"/>
                        <xs:element name="dateTimeList" type="xs:dateTime" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="complexNotTerminal">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="int1" type="xs:int"/>
                                                <xs:element name="string2" type="xs:string"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="complexTerminal" osd:access="RW">
                                <xs:complexType>
                                        <xs:sequence>
                                        <xs:element name="int1" type="xs:int"/>
                                        <xs:element name="string2" type="xs:string"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="complexList" minOccurs="0" maxOccurs="unbounded">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="int1" type="xs:int"/>
                                                <xs:element name="string2" type="xs:string"/>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="simpleTable" minOccurs="0" maxOccurs="unbounded">

                <xs:annotation>
                        <xs:documentation>
                                <osd:label>Very simple table</osd:label>
                                <osd:description>a table with 3 columns</osd:description>
                        </xs:documentation>
                </xs:annotation>
                <xs:complexType>
                        <xs:annotation>
                                <xs:appinfo>
                                <osd:table>
                                        <primaryKeys>/col1</primaryKeys>
                                </osd:table>
                                </xs:appinfo>
                        </xs:annotation>
                        <xs:sequence>
                                <xs:element name="col1" type="xs:integer"/>
                                <xs:element name="col2" type="xs:date"/>
                                <xs:element name="col3" type="xs:string"/>
                        </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:sequence>
                </xs:complexType>
                </xs:element>
                </xs:sequence>
                </xs:complexType>
</xs:element>
</xs:schema>
```

Références bibliographiques

[1] W3C. XML-Schema Part 1: Structures 2nd Ed. 2004 http://www.w3.org/TR/xmlschema-1

Site officiel du consortium W3C

[2] ODMG, « The Object Data Standard : ODMG 3.0 », Morgan Kauffman Publishers, 1999

Cette ouvrage défini les normes du standard ODMG 3.0.

[3] William R.Cook et Carl Rosenberger, « Native Queries for Persistent Objects, a Design White Paper ». Dr. Dobb's Journal (DDJ), February 2006.

Cet article présente la méthode « *Native Queries* » dans les langages de programmation Java et C#. Les points forts et les points faibles de cette méthode sont présentés dans cet article.

[4] William R. Cook, Siddhartha Rai, « Safe Query Objects : Statically Typed Objects as Remotely Executable Queries ». In Proceeding of the 27th Int. Conference on Software engineering (ICSE), pages 97-106, New York, NY, USA, 2005. ACM Press.

Cet article présente la méthode « *Safe Query Objects* » consistant à représenter des requêtes en tant qu'objets typés dans un langage de programmation. Les concepts de mapping entre objets et bases de données sont introduits dans cet article.

[5] Amar Zerdazi, Myriam Lamolle, « Modélisation des schémas XML par adjonction de métaconnaissances sémantiques ». Conférence ASTI, octobre 2005, Clermont Ferrant, France.

Cet article présente une modélisation, sous la forme de schémas XML, de source de données en introduisant des métaconnaissances spécifiques dans le but d'enrichir la structure sémantique de ces schémas.

[6] UML 2.0 « En concentré - Manuel de référence », O'Reilly, 2006.

Cette ouvrage défini les normes du standard UML 2.0.

[7] Mahmoud N., « Analyse/conception par points de vue : le profil VUML », thèse de doctorat d'informatique, INSA Toulouse, 2005.

Cette thèse propose une modélisation orientée point de vue. Nassar Mahmoud introduit par exemple le concept de classe *multivue* ayant pour but de stocker et de restituer l'information selon le profil d'un utilisateur (gestion de droits d'accès, changement dynamique de point de vue, vérification de la cohérence de différents points de vue)

[8] Amar Zerdazi, Myriam Lamolle, « Intégration de sources hétérogènes par matching semi-automatique de schémas XML étendus». Inforsid'06, Hammamet (Tunisie), juin 2006.

Dans la problématique d'intégration de sources de données hétérogènes, cet article propose un algorithme basé sur l'identification sémantique d'entités équivalentes afin de produire une série de mappings entre schémas XML.

[9] Bellatreche L., Boukhalfa K., « An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse Environment », 7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK'05) (3589), edited by Lecture Notes in Computer Science (LNCS), August, Springer-Verlag, 2005.

Le but de cet article est de présenter un algorithme de partitionnement de schémas, issu du domaine de la génétique.

[10] M. Ben Ahmed Mhiri, F. Gargouri, D. Benslimane, « Détermination automatique des relations sémantiques entre les concepts d'une ontologie », Inforsid'06, juin 2006, Hammamet, Tunisie.

Cet article présente une méthode ayant pour but de définir les relations existantes entre différentes ontologies.

[11] Furst F., Trichet F., « Raisonner sur des ontologies lourdes à l'aide de Graphes Conceptuels », Inforsid'06, Hammamet, Tunisie, juin 2006, p. 879-894.

Cet article présente une méthode de représentation graphique d'ontologies basée sur des graphes conceptuels.

[12] S. Staab and A. Maedche, « Axioms are Objects, too - Ontology Engineering beyond the Modeling of Concepts and Relations ».

Cet article présente une modélisation d'axiomes basée sur la représentation de concepts et de relations en tant qu'objets. Dans cette approche les axiomes sont classifés par types et considérés comme étant des objets complexes.

[13] Yannis KalFoglou, Marco Schorlemmer, « Ontology mapping : the state of the art ». The Knowledge Engineering Review 18(1) pp. 1-31. 2003.

Cet article défini la notion de mapping entre ontologies et présente un état de l'art complet sur les solutions existantes de mapping.

[14] Furst F., Leclère M., Trichet F., « TooCoM : a Tool to Operationalize an Ontology with the Conceptual Graphs Model », Proceeding of the Workshop on Evaluation of Ontology-Based Tools (EON'2003) at ISWC'2003, 2003, p. 57-70.

Cet article présent l'application TooCoM, un outil permettant de modéliser des ontoligies en utilisant les propriétés des graphes conceptuels.

[15] Pierra G., Hondjack D, Negue N. N., Bachir M., « Transposition relationnelle d'un modèle objet par prise en compte des contraintes d'intégrité de niveau instance », Inforsid'05, Grenoble, p.455-470.

Cet article présente une méthode de transposition d'un modèle objet vers le domaine relationnel. La méthode présentée repose sur un affinement des contraintes présentes dans les modèles conceptuels. Ces contraintes sont ensuite utilisées pour définir des règles de transposition d'un modèle objet vers un modèle relationnel.

[16] Dimitre Kostadinov – Verónika Peralta – Assia Soukane – Xiaohui XU, « Intégration de données hétérogènes basée sur la qualité ». INFORSID 2005, Grenoble, France. p 471-486.

L'objectif de cet article est de présenter une méthode qui permet de générer, de manière automatique, des requêtes de médiation dans un contexte relationnel et XML, et de les adapter aux besoins de l'utilisateur.

[17] Jean François Baget, Etienne Canaud, Jérôme Euzenat et Mohand Saïd-Hacid, « Les langages du web sémantique ».

Cet article présente un état de l'art des différents langages existants permettant de décrire des ressources situées sur le web.

[18] Widom, J.: « Research Problems in Data Warehousing ». In Proceedings of the 1995 International Conference on Information and Knowledge Management (CIKM), Baltimore, Maryland. 1995.

Cet article présente les avantages d'un entrepôt de données et en présente une architecture type.

[19] S. Abiteboul, S. Cluet, G. Ferran, M.-C. Rousset. « *The Xyleme Project* ». Computer Networks 39, 2002.

Cet article présent le projet *Xylème. Ce projet* est un système d'entrepôt dynamique ayant pour but de stocker et d'intégrer de manière semi automatique toutes les ressources XML du Web. Ce stockage permet à l'utilisateur final d'avoir un accès unique et transparent à toutes les données hétérogènes

[20] J. Rumbaugh, M.Blaha, W. Premerlani, F. Eddy, and V. Lorensen. « Object-Oriented Modeling and Design ». Prentice-Hall, 1991.

Cet ouvrage présente les principes des langages orientés objets et présente un formalisme graphique de représentation des relations entre objets appelé OMT.

[21] G. Booch, « *Object Oriented Design (with applications)* », Benjamin/Cummings Publishing Company, Inc., 1990.

Cet ouvrage présente la méthode de modélisation orienté objet appelée Booch.

[22] I. Jacobson, M. Christerson, P. Jonsson, and G. O¨ vergaard. « Object-Oriented Software Engineering. A Use Case Driven Approach. » ACM Press, 1992.

La méthode de modélisation d'objet OOSE est présentée dans cet ouvrage.

[23] Delobel, C., Reynaud, C., Rousset, M.C., Sirot, J.P., Vodislav, D. : « Semantic integration in Xyleme : A uniform tree-based approach ». Data and Knowledge Engineering 44, (2003). 267-298

Cet article présente l'approche à base d'arbres de l'entrepôt de données défini dans le projet Xylème. Le nbut de cet article est de présenter comment cette approche contribue à faire de Xylème un système efficace pour l'évaluation de requêtes, l'intégration de données et leur maintenance.

[24] Vargas-Solar G., Doucet A. : « Médiation de données : solutions et problèmes ouverts ». Actes des deuxièmes assises nationales du GdRI3, (2002).

Cet article présente des solutions, par une approche virtualisée ou par médiateur, à l'intégration de données provenant de sources hétérogènes.

[25] M. Zloof. « Query By Exemple : a database language ». IBM Systems Journal, 16(4). 1977.

Cet article présente la méthode de requêtage *Qery By Exemple*, créée par M. Zloof. Cette méthode consiste à définir un prototype d'objet que l'on souhaite obtenir à la suite d'une requête.

[26] Amar Zerdazi. « Représentation de schémas de bases de données hétérogènes sous forme de métaschémas XML ». Mémoire de DEA I.S.T. 2003

Ce mémoire présente un système ayant pour fonction d'extraire les schémas de bases de sources de données hétérogènes, sous la forme d'un schéma XML.

[27] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua sagiv, Jeffrey Ullman, Jennifer Widom. « The STIMMIS approach to mediation : Data Models and Languages ». CiteSeer, 1995.

Cet article présente le projet TSIMMIS. Un des objectifs de TSIMMIS est d'intégrer des sources hétérogènes, pouvant être très peu structurées et pouvant évoluer.

[28] Anthony Tomasic, Louiqua Rashid, Patrick Valduriez. « A data model and query processing techniques for scaling access to distributed heterogeneous databases in Disco ». IEEE Transactions on computers, special issues on Distributed Computing Systems, 1997.

Cet article présente un modèle de données ainsi que des méthodes de requêtages utilisées dans le projet Disco dans le cadre de l'intégration de données provenant de sources hétérogènes.

[29] Siméon : « Data Integration with XML : A Solution for Modern Web Applications ». Lecture at Temple University, March (2000).

Cet article présente des solutions, telle que le projer YAT, concernant l'intégration de données hétérogènes via XML.

Glossaire

Attribut XML [Attribute] Les attributs XML permettent de définir les caractéristiques d‘un élément par des couples « nom/valeur ». Par exemple, si l‘on considère que les deuxcaractéristiques (attributs) d‘un livre sont le titre et l‘auteur, un élément Livre peut être écrit en XML sous la forme <Livre titre="paroles" auteur="prevet"/>.

Analyseur XML [XML parser] API analysant et décodant les balises d‘un document XML afin de permettre à l‘application utilisant cet analyseur de traiter facilement des données au format XML.

Balise [Tag] Une balise permet de délimiter des données dans les langages de balisages tels que HTML, XML, …

Bases de données fédérées [federated databases] Une base de données fédérée est une base de données répartie hétérogène constitue de données fédérées, nécessite donc une architecture qui permet la communication entre les différentes sources de données.

Bases de données hétérogènes [heterogeneous databases] Plusieurs bases de données hétérogènes capables d‘interopérer via une vue commune (modèle commun).

CASTOR [Castor] CASTOR est un projet OpenSource de mapping objet/relationnel.
Référence : http://www.castor.exolab.org

Classe [class] On appelle classe la structure interne d‘un objet, c‘est-à-dire les données qu‘il regroupe, les actions qu‘il est capable d‘assurer sur ses données. Une classe est composée de deuxparties : -les attributs, parfois appelés données membres, il s‘agit des données représentant l‘état de l‘objet. -les méthodes, parfois appelées fonctions membres, il s‘agit des opérations applicables auxobjets.

Clé primaire [primary key] Chaque table doit contenir une cléprimaire. En effet, les clés primaires doivent être des champs contenant une valeur unique. Elle joue le rôle d‘identificateur d‘enregistrement (ligne). Les clés influent fortement sur les performances, mais elles sont également nécessaires pour garantir l‘intégralitédes données.

Clé étrangère [foreign key] Les clés étrangères sont des clés prises dans une table différente. Elles permettent de créer les relations entre différentes tables. Donc la clé étrangère référence la clé primaire d‘une autre table afin d‘indiquer leur relation.

Colonne [Column] Les colonnes d‘une base de données correspondent exactement aux attributs du modèle logique. Chaque attribut de ce dernier est représentépar une colonne dans le modèle physique. Un type de données est affectéà chaque colonne créée.

CORBA [Common Object Request Broker Architecture] CORBA est une architecture logicielle, créée en 1992 par différentes sociétés telles que SUN et IBM, pour le développement de composants. Ces composants, qui sont assemblés afin de construire des applications complètes, peuvent être écrits dans des langages de programmation distincts, être exécutés dans des processus séparés, voire être déployés sur des machines distinctes.

DOM [Document Object Model] DOM est une spécification du W3C pour proposer une API qui permet de modéliser, de parcourir et de manipuler un document XML. Le principal rôle de DOM est de fournir une représentation mémoire d‘un document XML sous la forme d‘un arbre d‘objets et d‘en permettre la manipulation (parcours, recherche et mise à jour). A partir de cette représentation (le modèle), DOM propose de parcourir le document mais aussi de pouvoir le modifier. Ce dernier aspect est l‘un des aspects les plus intéressant de DOM.
Référence : http://www.w3.org/DOM/

EBX.Platform [EBX.Plateform] Basé sur Java et XML Schema, EBX.Platform est un logiciel de Master Data Management. EBX.Platform est une solution standard et non-intrusive qui permet aux entreprises d'unifier et de gérer leurs données de référence et leurs paramètres à travers leurs systèmes d'information.

EBJ [Enterprise Java Bean] La technologie EJB est une architecture de composants logiciels côté serveur pour la plateforme de développement J2EE.

Elément XML [Element] Un élément XML est la brique de base d‘un document XML. Un élément est composé : -D‘une balise de début contenant des attributs et leurs valeurs, -De données, -D‘une balise de fin.

Interopérabilité [Interoperability] L‘interopérabilitéimplique la possibilitéde pouvoir demander et recevoir des services entre des « systèmes intéropérables »et pouvoir utiliser leurs fonctions.

IDE [Integrated Development Environment] est un programme regroupant un éditeur de texte, un compilateur, des outils automatiques de fabrication, et souvent un débogueur. Bien que des EDI pour plusieurs langages existent, bien souvent (surtout dans les produits commerciaux) un EDI est dédié à un seul langage de programmation.

JAVA [JAVA] Java est une technologie composée langage de programmation orienté objet et d'un environnement d'exécution. Préalablement nommé Oak, il a été créé par James Gosling et Patrick Naughton chez Sun Microsystems avec le soutien de Bill Joy. Le langage Java a la particularité principale d'être portable sur systèmes d'exploitation tels que Windows, Mac OS ou Linux. C'est la plateforme qui garantit la portabilité applications développées en Java.

J2EE [Java 2 Enterprise Edition] J2EE est une spécification pour le langage de programmation JAVA destinée aux applications d'entreprise. Toute implémentation de cette spécification contient un ensemble d'extension au cadre d'applications Java afin de faciliter la création d'applications réparties.

Métadonnées [metadata] Les métadonnées sont des données structurées, standardisées qui décrivent le contenu de document que souhaite partager des utilisateurs.

Métaschéma [metaschema] Un métaschéma XML est un schéma porteur de méconnaissances (sémantique) adjoint au schéma de la base de données. Dans ce document un métaschéma XML est un fichier XSD résultant de l‘opération de mapping d‘un schéma de base de données.

Namespaces [espaces de noms] Les namespaces permettent de créer des sous ensembles parmi les éléments qui composent un document XML. Tous les éléments appartenant à un namespace sont préfixés par son identifiant suivi du signe deux-points. Ce système permet de lever toute ambiguïtéen fournissant des identifiants uniques. Il correspond à la notion de paquetage des langages orientés objets.

NQ [Native Query] Méthode de requêtage utilisée au sein d'un langage de programmation. Le développeur a donc besoin de connaître uniquement un seul langage. La syntaxe de la requête peut ainsi être contrôlée directement à la compilation et par un IDE.

Objet [object] Un objet est une entitécohérente rassemblant des données et du code travaillant sur ses données. Un objet est caractérisépar des attributs, ce sont des variables stockant des informations d'état de l'objet. Les méthodes caractérisant son comportement (ensemble des actions). L'identité qui permet de le distinguer des autres objets, indépendamment de don état.

ODMG [Object Data Management Group] L'Object Data Management Group, a pour fonction depuis 1993 de fournir des spécifications concernant la persistance d'objets dans des bases de données.

ODBMS [Object DataBase Management System] Bases de données objet respectant les spécifications de l'ODMG.

OQL [Object Query Language] Langage d'interrogation d'objets défini par l'ODMG, proche du SQL. OQL permet de réaliser des requêtes qui ont pour but d'accéder à des données situées dans des référentiels de persistance suivant la norme ODMG.

QBE [Query By exemple] Méthode de requêtage créée par Moshe Zloof pour le compte de la compagnie IBM, en 1977. Cette méthode consiste à définir un prototype d'objet que l'on souhaite obtenir à la suite d'une requête.

Requête [query] On peut définir une requête, par une question posée à un système d'informations conformément à un model.

SAX [Simple Api for XML] L'API SAX propose un ensemble standardisé de mécanismes pour manipuler un document XML. Elle fonctionne sur le principe de l'envoi d'événements. Référence : http://www.saxproject.org/(le site officiel). Voir aussi : DOM.

Schéma de base de données [Schema of database] Organisation ou structure d'une BD, provenant généralement de la modélisation de données. Cette structure est décrite à l'aide d'un vocabulaire contrôléqui nomme des éléments de données et répertorie les contraintes pouvant s'appliquer (type de données, valeurs légales/illégales, etc.). Les relations entre éléments de données constituent également une part importante d'un schéma.

Schéma ODL [Object Description Language] Langage défini par l'ODMG pour spécifier la structure d'un schéma. Il faut noter qu'ODL est uniquement un langage de définition d'objets et non un langage de programmation. ODL possède une syntaxe précise permettant la définition de classes, d'interfaces, de relations, etc… L'ODMG a aussi mis en place un formalisme graphique de représentation d'un schéma.

Schémas XML [XML schemas] La spécification XML Schema est un nouveau mécanisme sur lequel travaille le W 3C afin de définir des vocabulaires XML en lieu et place des DTD. En conséquence, la spécification XML schémas du W 3C se compose de trois parties : -XML schemas Part 0 : Primer (Introduction), -XML schemas Part 1 : Structures (document relatif auxstructures), -XML schemas Part 2 : Datatypes (document concernant les types de données). Voir aussi la partie annexe.

SQL [Structured Query Language] SQL est un langage de requêtes de bases de données qui est devenu un standard de l'industrie en 1986. Ce langage permet de poser des questions complexes à une base de données et de la modifier. De nombreuses bases de données le supportent, par exemples MSSQL Server, DB2, Oracle, PostgreSQL.

Table [table] Les tables représentent l'élément essentiel des bases de données relationnelles. Une base de données relationnelle est constituéd'une ou de plusieurs tables destinées à stocker des informations. Une table est constituée de lignes. Chaque ligne est divisée en champs (colonnes), lesquels possèdent un certain type données.

UML [Unified Modeling Language] Créé par Grady Booch et Jim Rumbaugh, UML est un langage graphique de modélisation fournissant des éléments syntaxiques pour la plupart des systèmes logiciels, ce qu'UML désigne sous le terme « artifact »(en français: artefact). UML est une norme ouverte, maintenue par l'OMG.

Vue [view] Une vue est une table virtuelle qui contient des informations provenant d'autres tables. Une vue n'est rien d'autre que le résultat d'une instruction SELECT présentée comme une table virtuelle par le système de bases de données. Les vues permettent de simplifier des instructions SQL.

W3C [World Wide Web Consortium] Organisme de proposition et de normalisation des technologies, protocoles et langages du W eb. Le site officiel du W 3C est http://www.W 3.org

XSD [XML Schemas Definition] XSD est le langage des XML Schema Definition. XSD est aussi l'extension d'un fichier XML Schema.