Conception de profils UML pour la Gestion de Données de Référence

Myriam Lamolle*, Chan Le Duc*,
Ludovic Menet**

*LIASD - EA4383, IUT de Montreuil, Université Paris8
{m.lamolle, c.leduc}@iut.univ-paris8.fr,
**Orchestra Networks, Paris
ludovic.menet@orchestranetworks.com,

Résumé. Dans cet article, nous proposons un formalisme permettant à des utilisateurs d'être impliqués dans les phases de conception de modèles de Gestion de Données de Référence (dite MDM), tout en faisant abstraction des spécificités techniques de la plateforme cible. Pour cela, des profils sont définis pour étendre la sémantique d'UML, trop générique, à celle du MDM par l'intermédiaire d'un profil XML Schema. L'intégration de ces profils se fait dans le contexte de la plateforme EBX5 qui est un XMLWare.

1 Introduction

De nos jours, la nécessité d'échanger des informations et donc des modèles entre différents systèmes informatiques, différentes applications n'est plus à démontrer. L'approche couramment utilisée pour faciliter cette interopérabilité des systèmes est l'adoption d'une vision unifiée permettant de concevoir des applications en séparant la logique métier de toute plateforme technique. Dans cette optique, UML [1] représente un formalisme abstrait adéquat. Il fournit les fondements pour spécifier, construire, visualiser et décrire les artefacts d'un système logiciel. Cependant, par rapport au formalisme UML standard fourni par l'OMG [2] les développeurs souhaitent souvent rajouter des caractéristiques pour tenir compte de la spécificité de leur domaine d'application. Le domaine de la gestion des données de références, dite MDM [3], en est un exemple. Pour ce faire, UML est doté d'un mécanisme d'extensibilité fondé sur des stéréotypes, des contraintes et des valeurs étiquetées. Un tel mécanisme permet de personnaliser le métamodèle UML, le résultat étant un *profil* UML. La notion de profil a été introduite dans le standard UML 1.3 comme un moyen de structurer des extensions.

Nous présentons une mise en application de ces mécanismes d'extension d'une métamodélisation UML permettant à la fois de présenter des modèles XML Schema (Gao et al., 2011) et des modèles liés au domaine du MDM. En utilisant les profils UML définis, nous avons la capacité de définir d'une manière abstraite de tels modèles. Un diagramme de classes UML demeure dans le domaine de l'abstraction dans la mesure où il représente la sémantique d'un

1. Unified Modeling Language
2. Object Management Group, (http ://www.omg.org/)
3. Master Data Management

domaine. Afin d'obtenir un modèle exploitable techniquement et productif, nous devons être en mesure de générer automatiquement les modèles spécifiques à une plateforme, dans notre cas EBX5 [4] qui est un XMLWare pour le MDM, liés aux modèles exprimés en UML. La génération de modèles est possible à l'aide de règles de transformation. Pour définir des règles de transformation entre deux modèles, il faut préalablement établir des appariements (ou mappings) entre les métamodèles impliqués dans les processus de transformation (Sen et al., 2012). De nombreux travaux sur les problématiques de mapping entre métamodèles ont été menés. Dans (Levendovszky et al., 2002), le mapping est défini comme étant un ensemble de règles de transformation de modèles permettant de traduire des instances d'un métamodèle source M_A en instance d'un métamodèle cible M_B. Pour automatiser les transformations, nous présentons une correspondance partielle entre M_A et M_B en calculant la proportion d'équivalence de M'_A et M'_B par rapport à M_A et M_B obtenant ainsi une interopérabilité par un isomorphisme partiel entre UML et XML Schema dans le contexte du MDM. Par exemple, les notions d'interface et d'opération UML ne sont pas prises en compte dans nos règles de correspondance car elles n'existent pas dans XML Schema.

Nous allons dans un premier temps expliciter la notion de MDM, puis sa représentation dans la plateforme EBX5. La section 3 présente un tour d'horizon des travaux existants concernant les transformations UML/XML Schema. La section 4 spécifie les mappings nécessaires et les profils mis en place pour faciliter l'interopérabilité par un isomorphisme partiel entre ces métamodèles. La section 5 présente l'implémentation elle-même et l'expérimentation faite avec l'éditeur ArgoUML [5] par deux exemples. Une synthèse de l'existant et les évolutions futures envisagées concluent cet article.

2 La gestion des données de référence

La gestion des données de référence ou Master Data Management (MDM) est une approche émergente d'intégration de données fondée sur l'approche matérialisée (Abitboul et al., 2002). Dans cette approche, les données issues de sources hétérogènes sont copiées dans un entrepôt de données (ou référentiel).

Le MDM étant une discipline assez récente, très peu de travaux existent à ce jour (iWays Master Data Center [6], Oracle MDM suite [7], IBM MDM [8], EBX5 MDM [9]) et sont principalement des solutions industrielles. Deux conditions sont nécessaires à une architecture centralisée MDM à savoir (i) il faut disposer d'un outil de MDM générique capable d'accueillir le modèle commun d'information pour toutes les natures de données. Sans ce niveau de généricité, il faudrait accepter des MDM par silos organisés autour des domaines d'information (Client, Produit, Organisation, paramètres fonctionnels, paramètres techniques, etc.) et les conséquences néfastes en terme de duplication des référentiels (ce que l'on cherche à éviter) et de duplication des fonctions de gouvernance (gestion des versions, interface homme-machine d'adminis-

4. http ://www.orchestranetworks.com/product/
5. http ://argouml.tigris.org/
6. http ://www.iwaysoftware.com, 2009
7. http ://www.oracle.com/master-data-management/, 2007
8. http ://www-01.ibm.com/software/data/master-data-management/, 2004
9. http ://www.Orchestranetworks.com, 2000

tration, etc.) ; (ii) il faut une méthode pour la modélisation et la négociation du modèle commun d'information faisant abstraction des formats *propriétaires* des différents systèmes.

Le premier point est assuré par les solutions MDM d'EBX5. Concernant le second point, notre approche pour définir des modèles de données est une solution adéquate.

2.1 Principes du MDM

Une donnée de référence (MD) peut être définie comme étant une donnée nécessaire et devant être non redondante dans un système d'information. A partir de ce constat, le MDM est assimilé à une couche supplémentaire de l'approche matérialisée. Il centralise les données dans un unique référentiel et les propage entre les applications déployées au sein d'un système d'information. Les données sont identifiées de la même manière que dans l'ensemble du système d'information. Il est ainsi possible d'appliquer une stratégie de mutualisation de l'information.

2.2 Solution MDM EBX5

XML est devenu le standard pour échanger de l'information au travers d'Internet. Le W3C a préconisé l'utilisation de XML Schema pour définir la structure des documents XML. XML Schema est un formalisme permettant de décrire la structure d'un document XML de façon beaucoup plus précise qu'une simple DTD [10] (Bosak et al., 1998). Fondée sur le standard XML Schema (Gao et al., 2011), EBX5 simplifie la définition de modèles qui ont pour but 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 et typées et ayant de puissantes propriétés de validation. EBX5 repose sur deux concepts (Menet et al., 2008) à savoir (i) des modèles de données qui sont des documents XML Schema définissant la structure des données de référence, (ii) des jeux de données qui sont des instances XML des modèles de données, qui sont les valeurs réelles au temps T des données de référence.

L'utilisation d'XML Schema permet de préciser que chaque noeud du modèle de données correspond à un type de données existant et conforme au standard du W3C. D'autre part, le formalisme d'XML Schema permet de spécifier des contraintes (énumération, longueur, cardinalités, etc.), des informations relatives au modèle données de référence et à son instanciation (connecteurs d'accès, classe d'instanciation Java, restriction d'accès, etc.) et des informations de présentation (libellé, description, formatage, etc.) pour chaque noeud du schéma. Un jeu de données correspond à une instance du modèle de données de référence. A tout noeud du modèle de données déclaré valorisable correspond un noeud dans le jeu de données. EBX5 est capable de gérer de façon générique un ensemble de fonctionnalités liées aux problématiques du MDM à savoir : (i) les versions de données et leur historisation, (ii) l'héritage des valorisations selon un arbre de contextes, (iii) la gestion des habilitations, (iv) l'accessibilité, l'interrogation, la qualité et la sécurité des données, (v) le workflow intégré.

Le choix d'une architecture XML dans EBX5 permet de définir des modèles de données riches, structurés, fortement typés et d'associer des contraintes métier. Cependant, si l'utilisation d'XML est appropriée à la définition des modèles, elle nécessite une connaissance appro-

10. Document Type Definition

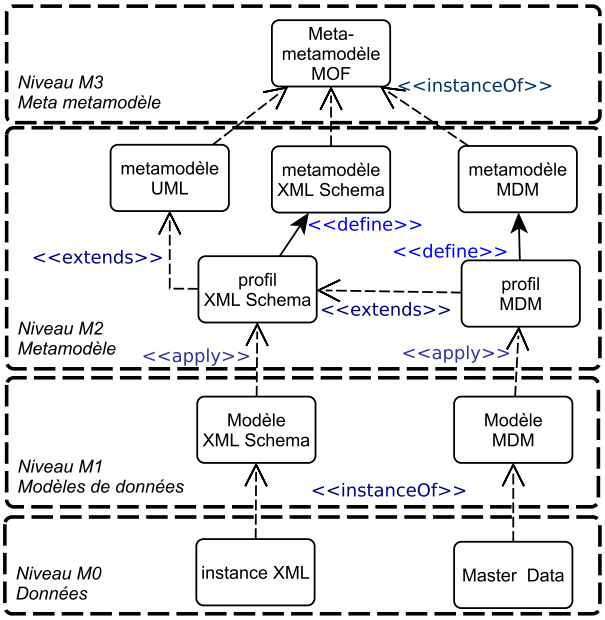

FIG. 1 – *Adaptation de l'architecture "4 couches" au MDM*

fondie de ce langage par les différents acteurs impliqués dans le processus de définition d'un modèle de données. Ceci nous a suggéré l'introduction d'une démarche guidant les concepteurs de façon qu'ils puissent se concentrer uniquement sur la modélisation et l'intégration, et non sur la technologie à utiliser (Bézivin et Gerbé, 2001).

L'approche d'une approche objet et standard pour améliorer la compréhension du modèle et la sémantique associée (sémantique MDM dans notre cas) semble être la meilleure voie. La figure 1 présente la mise en place d'une solution (détaillée dans la section 4) selon l'approche MDA qui a pour objectif principal de faire abstraction de la couche physique (niveau PSM [11], ici EBX5) au profit de la couche fonctionnelle (niveau PIM [12]) selon l'architecture "4 couches" préconisée par l'OMG.

3 État de l'art sur les transformations UML-XML Schema

Dans le cadre de transformations entre UML et XML Schema, nous pouvons citer les travaux de (Booch et al., 1999) qui décrivent une notation graphique dans UML pour concevoir des schémas XML mais elle est contextualisée à un schéma SOX utilisé par CommerceOne. (Carlson, 2001) abordent la transformation de modèles UML en XML Schema et sa

11. Platform Specific Model (de l'OMG)
12. Platform Independant Model

réciproque, obtenant ainsi un mapping bijectif entre ces deux formalismes. Le mapping réalisé utilise comme fondement un profil UML définissant des concepts spécifiques à XML Schema. Ce profil est utilisé afin d'étendre la sémantique d'un modèle UML à la sémantique d'un modèle XML Schema. Cette approche permet un mapping d'une grande partie des concepts introduits par XML Schema, mais ne prend pas en compte certains concepts tels que la notion de groupe (list, union), de contrainte d'identité (key, keyref, unique), de contrainte de généricité (substitutionGroup), etc. De plus, certains concepts importants d'UML tels que l'agrégation, la composition, l'association et la documentation ne sont pas pris en compte lors d'une transformation d'un modèle UML vers un modèle XML Schema. Ce mapping a été mis en application par HyperModel (Carlson, 2006). HyperModel est un plug-in pour l'IDE Eclipse implémentant le mapping bijectif UML/XML. Cet outil est fonctionnellement opérationnel mais souffre de quelques limitations lors d'une transformation d'un modèle UML vers un modèle XML Schema car :

- certains concepts ne sont pas appariés (agrégation, composition, etc.) ;
- des éléments sont appariés plusieurs fois entraînant une redondance d'information et une inconsistance dans le modèle résultat ;
- une perte d'information, notamment concernant les contraintes de cardinalité, se produit sur certains modèles ;
- des modèles XML Schema générés ne sont pas valides au regard de la spécification du W3C.

(Bernauer et al., 2004) outrepassent certaines de ces limitations en étendant si nécessaire certains profils. Les résultats des approches de mapping entre UML et XML Schema sont aussi pris en compte lors de l'élaboration des profils afin de favoriser une meilleure rétro-ingénierie.

(Routledge et al., 2002) abordent le mapping de manière traditionnelle entre UML et XML Schema par l'intermédiaire de l'approche à trois niveaux issus du monde des bases de données à savoir les niveaux conceptuel, logique et physique. Dans le contexte d'un diagramme de classes UML, le niveau conceptuel décrit les objets et leurs relations. Le niveau logique représente les structures de données XML Schema sous la forme d'un profil UML. Le niveau physique représente directement le modèle XML Schema. De la même manière que les travaux de Carlson, certains éléments spécifiques d'UML tels que l'agrégation, la composition et d'autres ne sont pas pris en compte. (Narayanan et Ramaswamy, 2005) considèrent la composition comme une association et proposent un ensemble de spécifications de mappings. D'autres travaux ont été réalisés dans le même contexte par (Conrad et al., 2000), (Wu et Hsieh, 2002) et (Kurtev et al., 2003). Une étude approfondie des différentes approches de transformation est présentée dans (Dominguez et al., 2007). A partir de là, (Domínguez et al., 2011) ont proposé une architecture générique (GEA) qui, dans le cas des transformations UML-XML Schema, permet de mettre à jour automatiquement les changements faits dans le modèle conceptuel UML, de transformer les profils UML vers XML, de propager les changements des classes stéréotypées UML dans les schémas XML et leurs instances.

Notre objectif est de combler une partie des limitations de ces travaux ou de les adapter pour faciliter la définition de modèle de données dans le contexte du MDM. Le but de la transformation est de rendre opérationnel le modèle UML qui a été défini. La section suivante présente les mappings établis entre éléments UML et XML Schema, y compris les notions décrivant des relations de dépendance.

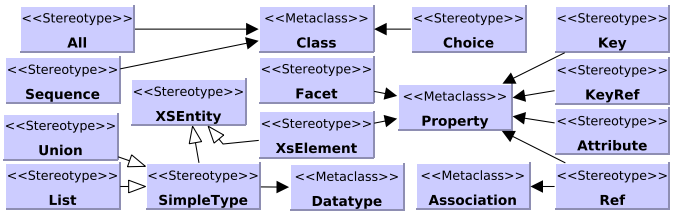

FIG. 2 – *Extrait du profil XML Schema en UML*

4 Spécification des mappings pour le MDM

Afin d'avoir une spécification des mappings UML/XML Schema la plus précise possible, deux profils ont été mis en place dans le métamodèle UML permettant de définir des modèles XML Schema génériques ou appliqués au MDM.

4.1 Profil XML Schema

Le profil XML Schema est constitué d'un ensemble de stéréotypes et de valeurs marquées correspondant à la sémantique de ce langage. Les stéréotypes de la figure 2 montrent un extrait de ce profil XML Schema. Ils sont fondés en partie sur des travaux cités dans la section 3. Ils héritent respectivement de l'élément *Class*, *Datatype*, *Property* et *Association* du métamodèle UML. L'ajout de sémantique se fait via l'extension de ces métaclasses. Cela permet à l'utilisateur d'ajouter de la sémantique au modèle de données défini en UML. Par conséquent, chacun d'eux sera instancié par le constructeur de ces éléments du métamodèle. Dans notre cas, la définition de ces stéréotypes permet d'introduire plus de sémantique, externe à UML, permettant d'adopter une perspective XML Schema lors de la modélisation d'un domaine. Des stéréotypes pour modéliser des bases de données relationnelles en XML Schema ont aussi été créés (par exemple, «*table*»). Cependant, l'utilisation de diagrammes de classes nous impose d'appliquer des restrictions concernant leur définition. En effet, certains concepts UML tels que les opérations, les interfaces ou encore les classes internes ne peuvent être représentés avec XML Schema via son profil UML. Dans le cadre du MDM, ce profil a été étendu par un second profil présenté dans la section suivante.

4.2 Profil MDM

Sachant que les modèles MDM (dits modèles de données de référence dans la plateforme EBX5) sont sauvegardés selon le formalisme XML Schema, il faut définir un profil spécifique pour qu'un utilisateur final puisse créer un modèle MDM. Ce profil MDM est constitué de quatre parties :

- les stéréotypes permettant de définir un modèle de données de référence à partir d'un modèle XML Schema,
- les stéréotypes représentant les propriétés avancées définis par notre solution MDM,

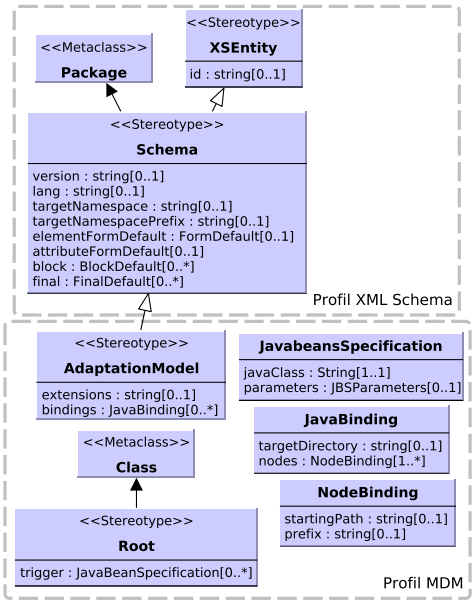

FIG. 3 – *Extrait du profil MDM en UML*

– les stéréotypes représentant des contraintes avancées définies à partir des contraintes XML Schema,
– les stéréotypes représentant des éléments de documentation permettant d'étendre les entités de documentation XML Schema.

Ces stéréotypes vont garantir que le mapping est complet parce que le profil MDM définit toutes les propriétés de la plateforme cible.

Notre modèle de données de référence est un modèle de données enrichi pour les données de références (ou Master Data). Globalement, les objectifs sont de garantir la cohérence de ces données et de faciliter leur gestion. Concrètement, le modèle de données de référence est un document conforme au standard XML Schema. Les principales caractéristiques supportées sont : (i) une riche bibliothèque de types de données, (ii) la possibilité de définir des structures complexes, (iii) la possibilité de définir des listes simples d'éléments, (iv) la spécification de contraintes de validation (facettes XML Schema) telles que les énumérations, unicité, cardinalité, etc. Notre solution utilise également les capacités d'extension d'XML Schema afin de définir des informations utiles telles que les types prédéfinis (locale, ressource, html, etc.), les définitions de tables et de contraintes de clé étrangère, les mapping de données et de Java beans, les contraintes de validation avancées (facettes étendues) (par exemple des énumérations dynamiques), les informations de présentation étendues : libellé, description, messages d'erreur, etc.

La figure 3 présente les stéréotypes permettant de définir un modèle de données de référence à partir d'un modèle XML Schema. Le stéréotype *AdaptationModel* spécifie un modèle pour le MDM. C'est une extension du stéréotype *Schema* dont il hérite. Il a une propriété sup-

plémentaire *bindings* qui permet de spécifier par exemple les types Java à générer à partir du schéma XML. Chaque élément *binding* définit une cible de génération. Il doit être situé à l'emplacement *xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding* (notation XPath) où le préfixe *ebxbnd* fait référence à l'espace de nommage identifié par une URI spécifique. Plusieurs éléments *binding* peuvent être définis s'il y a des cibles de génération distinctes. L'attribut *targetDirectory* de la classe *JavaBinding* définit le répertoire racine de génération des types Java (généralement, il s'agit du répertoire des codes sources d'un projet par exemple). Un chemin relatif est interprété relativement au répertoire courant d'exécution de la machine virtuelle Java (et non par rapport à la localisation du schéma). En définissant des bindings, nous obtenons la capacité d'associer des constantes Java aux chemins d'un modèle MDM représenté par un modèle XML Schema. Pour cela, nous générons une ou plusieurs interfaces à partir d'un noeud du schéma (pouvant être le noeud racine "/"). Les noms d'interface sont décrits dans les balises *javaPathConstants* avec l'attribut *typeName* et le noeud associé est décrit dans la balise *nodes* avec l'attribut *root*. Par exemple, la définition d'un modèle MDM et de bindings sera :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:annotation>
  <xs:appinfo>
   <ebxbnd:binding targetDirectory="chemin_dossier">
    <javaPathConstants typeName="com.org.NomClasseJava">
     <nodes root="/root" prefix=""/>
    </javaPathConstants>
   </ebxbnd:binding>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:schema>
```

Concernant les stéréotypes *Datatype* du profil "XML Schema", le profil MDM les complète par les stéréotypes *AnyURI* en *Resource* et *String* en *Text*, *Email*, *HTML*, *Password*, *Locale*. Enfin, nous définissons un certain nombre de structures avancées dans le modèle MDM :

- *MDMEntity*, élément abstrait pour répondre à des besoins métier de manipulation et de présentation des données ;
- *Domain*, élément complexe possédant des propriétés spécifiques à une données de référence ;
- *MDMSimpleElement*, élément simple possédant des propriétés spécifiques à une données de référence ;
- *Table*, élément complexe qui devra être interprété comme une table au sens SGBD ;
- *PrimaryKey*, élément associé faisant partie de la clé primaire d'une table ;
- *Function*, indique que la valeur de l'élément doit être alimentée de manière spécifique par une classe Java ;
- *AutoIncrement*, indique que la valeur de l'élément est auto-incrémentée.

Des contraintes avancées sont aussi spécifiées par des stéréotypes par extension du stéréotype *DynamicFacet* qui hérite du stéréotype *Facet* du profil pour définir les longueurs, les cardinalités, les énumérations, les clés étrangères, etc.

Cette solution MDM propose un ensemble de fonctionnalités permettant de gérer des données de référence. Cependant, la définition d'un modèle MDM via un diagramme de classes atteint ses limites en terme de visualisation et d'ergonomie lorsque le modèle atteint une taille importante.

D'autre part, dans certains cas, des fonctionnalités spécifiques et non présentes dans notre outil peuvent être requises telles que la mise à jour automatique de systèmes distants ou la définition de pistes d'audit spécialisées. Pour pallier cette limitation, nous avons défini la notion de *service* dans le but d'intégrer dans cette solution MDM des applications Java/JSP. Les services enrichissent l'outil et fournissent ainsi aux utilisateurs des fonctionnalités additionnelles. Le stéréotype *Service* hérite du stéréotype *«Metaclass Class»*. La propriété *resourcePath* spécifie le chemin du service à importer et la propriété *ActivatedForPath* spécifie sur quels éléments du schéma le service est disponible. Par défaut, un service est disponible sur l'instance du modèle MDM. Par exemple, la définition d'un service sera :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="exempleService">
  <xs:annotation>
   <xs:appinfo>
    <osd:service resourcePath="/importXML/importXML.jsp"
                 activatedForPath="/root/table1" />
   </xs:appinfo>
  </xs:annotation>
 </xs: complexType >
 ...
</xs:schema>
```

La mise en place de ces profils est la première étape pour optimiser et standardiser la définition de modèles de données. La couche d'abstraction permet de représenter un modèle de données de référence indépendamment d'un domaine d'application précis grâce à nos profils. L'enrichissement sémantique réalisé à la fois dans XML Schema et dans UML dans le but d'établir des correspondances entre ces deux technologies favorise la spécification des mappings entre UML et XML Schema. Par cette approche, la définition des mappings assure le passage de la solution fonctionnelle (UML) à la solution technologique (XML Schema). Cette approche est applicable de manière générale à toute modélisation de modèle XML Schema et s'applique aussi au domaine spécialisé du MDM. Couplée à des méthodes de transformation, présentées dans la section suivante, l'utilisation de ces profils permet de s'abstraire de toute spécificité technique liée à la définition des modèles XML Schema appliqués au MDM.

5 Implémentation de mappings

5.1 Transformation d'un modèle UML en XML Schema

Il est question dans un premier temps d'être capable à partir d'un modèle UML de générer de manière automatique un modèle XML Schema, ou, autrement dit, de transformer un modèle fondé sur une architecture MOF vers un modèle textuel. Parmi les formalismes existants, nous avons choisi d'utiliser le langage MOFScript (Oldevik, 2011). Ce choix a été motivé par le fait que les langages de transformation tels que QVT (OMG, 2011a) sont essentiellement focalisés sur des transformations de type *modèle à modèle* ; or, nous avons simplement besoin de générer un modèle textuel à partir d'un modèle. Autrement dit, il s'agit de générer du code à partir d'un modèle. Dans cette section, uniquement les spécificités de MOFScript utilisées pour implémenter nos règles de transformation sont décrites.

Règles de transformation MOFScript a pour but de transformer un modèle vers du texte. Les modèles sources sont des instances de métamodèles devant être fondés sur l'architecture du MOF. Il est possible de définir le métamodèle utilisé avec l'instruction *texttransformation*. Cette étude se fonde sur le métamodèle d'UML ; ce qui donne par exemple :

```
texttransformation exempleDeTransformation
  (in uml:"http://www.eclipse.org/uml2/1.0.0/UML"))
```

Les points d'entrée des règles de transformation définissent où commence l'exécution de la transformation définie dans un script. Ce point d'entrée est similaire à la méthode *main()* existante en Java. Il est appliqué dans un contexte donné qui indique quel élément du métamodèle représente le point de départ de la transformation à exécuter. Le point d'entrée sur un modèle UML est défini à l'aide de l'instruction `uml.Model::main()`. Par exemple,

```
uml.Model::main () {
  self.ownedMember->forEach(p:uml.Package) {
    p.transformerPackage() }
}
```

Un point d'entrée peut être appliqué sur plusieurs instances d'une entité d'un métamodèle. Dans ce cas, il sera exécuté pour chaque instance de l'entité spécifiée. La définition d'un point d'entrée sur toutes les instances de l'entité *Class* du métamodèle UML s'écrit par exemple :

```
uml.Class::main () { self.transformerClasse()  }
```

Dans l'exemple ci-dessus, une transformation spécifique est exécutée pour chaque classe d'un modèle UML. Il est à noter que des règles de transformation peuvent être définies sans nécessairement spécifier de contextes particuliers. Dans ce cas, le point d'entrée défini sera exécuté une seule fois. Un point d'entrée sans contexte est défini avec ou sans le mot clé *module*.

Une règle de transformation MOFScript peut être assimilée à une méthode Java dans la mesure où une règle définit un ensemble d'instructions exécutées lorsque la règle est appelée. Une règle peut retourner un objet qui peut être soit un type de données natif MOFScript, soit un type du métamodèle source. L'exemple ci-dessous présente le squelette d'un script permettant de générer un modèle XML Schema à partir d'un modèle UML. L'instruction *file* spécifie le chemin du modèle XML Schema à générer.

```
uml.Package::transformerPackage() {
 file(self.obtenirNomPaquetage() + ".xsd") ;
 // ensemble d'instructions
 self.ownedMember->forEach(c:uml.Class) ;
 c.transformerClasse()
}

uml.Class::transformerClasse() {
 // ensemble d'instructions
 self.traiterStereotypes();
 self.ownedAttribute->forEach(p : uml.Property) {
  p.transformerAttributs() }
 self.transformerAssociations();
 // ensemble d'instructions
}
```

Ce script de transformation itère sur tous les paquetages du modèle UML source. Pour chacun de ces paquetages, un document XML Schema est créé. Le script traite ensuite les classes d'un paquetage, leurs attributs et leurs associations. De la même manière que dans les langages de programmation classiques, il est possible de définir des règles retournant une valeur à l'aide de l'instruction *result*. Cette valeur peut être réutilisée par les règles appelantes.

MOFScript définit un ensemble d'opérations (*hasStereotype(String nomStereotype), getAppliedStereotypes(), hasValue(Stereotype unStereotype, getValue(Stereotype unStereotype*, etc.) applicables sur des modèles UML [13].

La composition (resp. agrégation) est matérialisée dans un document XML par les mécanismes d'extension *<xs:annotation><xs:appinfo>* préconisés par le W3C. Soit une classe *C1* composée de plusieurs éléments de la classe *C2*, la transformation génèrera :

```
1. <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema>
2. <xs:element name="C1">
3.  <xs:complex><xs:sequence>
4.    <xs:element name="C2" minOccurs="0" maxOccurs=unbounded">
5.      <xs:annotation><xs:appinfo>
6.        <osd:composition />
7.      </xs:appinfo></xs:annotation>
      ...
8.    </xs:element>
9.  </xs:sequence></complex>
10. </xs:element>
11. </xs:schema>
```

où les lignes 5, 6, 7 portent la sémantique de la composition (resp. agrégation).

5.2 Transformation d'un modèle XML Schema vers un modèle UML

Une seconde étape consistant à transformer XML Schema étendu au MDM vers UML revêt un aspect plus délicat dans la mesure où nous cherchons à définir un modèle graphique à partir d'un modèle technique en dehors de tout standard. Ceci nécessite donc de pouvoir représenter les modèles XML Schema et UML dans un formalisme commun. XMI (OMG, 2011b) permet de représenter un modèle UML sous la forme d'un document XML permettant ainsi d'établir une corrélation entre un modèle XML Schema et un modèle UML. XSLT (Clark, 1999), quant à lui, est un langage permettant d'appliquer des règles de transformation sur un document XML pour obtenir un autre document XML. Ces règles de transformation sont décrites dans des feuilles de styles XSL. Ce langage de transformation structurelle utilise XPath (Clark et DeRose, 1999) pour sélectionner et filtrer les noeuds d'arbres. XSLT est un langage déclaratif à base de règles spécifiant comment sera le résultat et non comment sont effectuées les transformations sur les noeuds de l'arbre source. Dans ce processus, XSLT est utilisé pour transformer un document XML Schema en XMI. Chaque partie *xsl:template* définit une transformation à faire selon l'appariement trouvé. Une transformation spécifie les balises XMI à créer quand le parseur rencontre un certain type de balise dans le document XML Schema analysé et peut demander au parseur la recherche d'autres appariements applicables (*i.e.*, *apply-templates*). Par exemple, le "template" pour traiter la balise *<xs:schema>* est :

13. Nous choisissons de ne pas présenter en détails les scripts développés dans la mesure où il ne représente qu'un aspect technique des mappings établis.

```
<xsl:template match="xs:schema"><XMI xmi.version="1.2">
 <XMI.header>
  <XMI.documentation><XMI.exporter>Transfo. XML Schema vers XMI
  </XMI.exporter></XMI.documentation>
  <XMI.metamodel xmi.name="UML" xmi.version="1.4"/>
 </XMI.header>
 <XMI.content>
  <UML:Model xmi.id="{generate-id()}"
    name="{substring-after(@targetNamespace,
           'http://psol.com/uml/')}"
    visibility="public" isSpecification="false"
    isRoot="false" isLeaf="false" isAbstract="false">
    <UML:Namespace.ownedElement><xs:apply-templates />
    </UML:Namespace.ownedElement>
 </UML:Model></XMI.content></XMI>
</xsl:template>
```

6 Expérimentation

Pour mettre en application cette approche, un outil de modélisation a été développé permettant de définir des modèles MDM et par extension des modèles XML Schema. L'outil développé est fondé sur l'éditeur ArgoUML [14], logiciel libre de modélisation UML. Pour l'expérimentation dans ArgoUML, un module a été développé incluant les profils UML présentés précédemment ainsi que les fonctionnalités d'import et d'export de modèles XML Schema. La fonctionnalité d'import permet de générer un diagramme de classes UML à partir d'un modèle XML Schema. La fonctionnalité d'export permet de générer le code XML Schema d'un diagramme de classes défini avec ces profils UML.

Le navigateur de profils permet de visualiser les stéréotypes et les types de données définis. L'extension développée permet de générer un modèle XML Schema à partir d'un modèle UML. Il est possible de visualiser directement le schéma XML généré en changeant de perspective graphique/textuelle.

Nous allons illustrer maintenant l'utilisation de ces profils UML par deux exemples. Le premier exemple est consacré à la définition d'un modèle XML Schema générique ; le second exemple présentera la définition d'un modèle de données de référence MDM d'EBX5.

6.1 Exemple 1 : définition d'un modèle de bon de commande

Cet exemple concerne la modélisation d'un bon de commande émis par une personne et géré par une application de facturation. Le bon de commande est constitué d'un élément principal, *commandeType*, et des sous-éléments *adresseLivraison*, *adresseFacturation*, *commentaire* et *produits*. En utilisant le profil UML défini, dédié à la sémantique d'XML, cette structure XML Schema d'un bon de commande peut être modélisée à l'aide d'un diagramme de classes. La figure 4 présente le diagramme de classes associé au code XML Schema présenté par la suite.

14. http ://argouml.tigris.org/

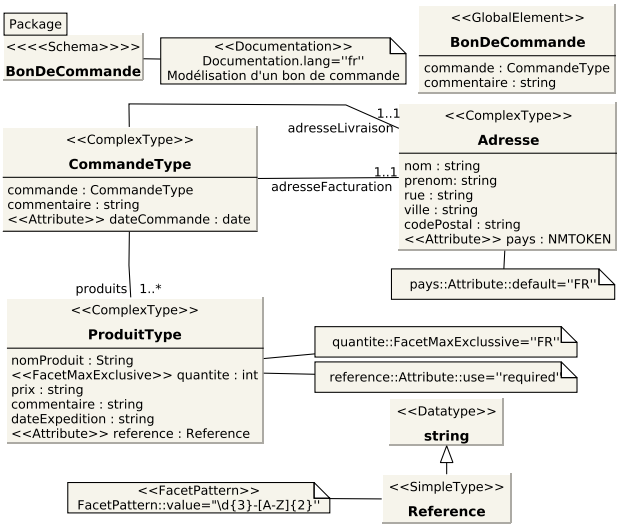

FIG. 4 – *Modélisation UML d'un bon de commande par le profil XML Schema*

Les différents éléments (à l'exception de l'élément *comment*) contiennent à leur tour d'autres sous-éléments, et ainsi de suite, jusqu'à ce qu'un sous-élément comme par exemple *prix* contienne un type simple plutôt qu'un sous-élément. On admet que les éléments qui contiennent des sous-éléments ou portent des attributs sont de type complexe, tandis que les éléments qui contiennent des nombres (ou des chaînes de caractères, des dates, etc.) sans autre sous-élément sont de type simple. Quelques éléments ont des attributs ; les attributs sont toujours de type simple. Dans une instance de document, les types complexes et une partie des types simples qui y sont utilisés sont définis dans le schéma associé. Les autres types simples sont ceux qui sont prédéfinis par XML Schema.

A partir du diagramme de la figure 4 et des règles de transformation implémentées, le modèle XML Schema suivant est généré automatiquement :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:annotation>
  <xs:documentation xml:lang="fr">Bon de commande
  </xs:documentation>
</xs:annotation>
<xs:element name="commande" type="commandeType"/>
<xs:element name="commentaire" type="xs:string"/>
<xs:complexType name="commandeType"><xs:sequence>
  <xs:element name="adresseLivraison" type="Adresse"/>
  <xs:element name="adresseFacturation" type="Adresse"/>
  <xs:element ref="commentaire" minOccurs="0"/>
  <xs:element name="produits" type="ProduitType"
              minOccurs="0" maxOccurs="unbounded">
</xs:sequence>
```

```
<xs:attribute name="dateCommande" type="xs:date"/>
</xs:complexType>
<xs:complexType name="Adresse"><xs:sequence>
   <xs:element name="nom" type="xs:string"/>
   <xs:element name="prenom" type="xs:string"/>
   <xs:element name="rue" type="xs:string"/>
   <xs:element name="ville"  type="xs:string"/>
   <xs:element name="codePostal" type="xs:string"/>
  </xs:sequence>
<xs:attribute name="pays" type="xs:NMTOKEN" default="FR"/>
</xs:complexType>
<xs:complexType name="ProduitsType"><xs:sequence>
  <xs:element name="nomProduit" type="xs:string"/>
  <xs:element name="quantite">
    <xs:simpleType>
      <xs:restriction base="xs:positiveInteger">
        <xs:maxExclusive value="100"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="prix" type="xs:decimal"/>
  <xs:element ref="commentaire" minOccurs="0"/>
  <xs:element name="dateExpedition" type="xs:date" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="reference" type="Reference" use="required"/>
</xs:complexType>
<xs:simpleType name="Reference">
 <xs:restriction base="xs:string">
  <xs:pattern value="\d{3}-[A-Z]{2}"/>
 </xs:restriction>
</xs:simpleType>
</xs:schema>
```

6.2 Exemple 2 : définition d'un modèle de données

Cet exemple concerne plus précisément un modèle de données de référence représentant une base de données relationnelle de publication d'ouvrages. La représentation UML du modèle de données de référence correspondant à cette base de données en utilisant le profil MDM présentée dans la figure 5.

De la même façon que dans l'exemple 1, le code XML Schema suivant est généré automatiquement par ArgoUML grâce à l'implémentation des règles de transformation définies.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="root" osd:access="--">
 <xs:complexType><xs:sequence>
   <xs:element name="Titles" type="Title"
       minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="Publishers" type="Publisher"
       minOccurs="0" maxOccurs="unbounded"/>
```

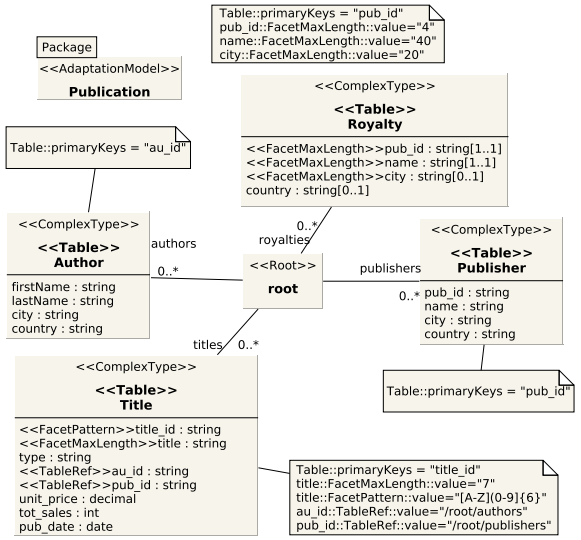

FIG. 5 – *Exemple de modèle de données défini à l'aide du profil MDM*

```
   <xs:element name="Authors" type="Author"
       minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="Royalties" type="Royalty"
       minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence></xs:complexType>
</xs:element>
<xs:complexType name="Publisher">
<xs:annotation>
 <xs:appinfo>
  <osd:table>
   <primaryKeys>/pub_id</primaryKeys>
  </osd:table>
 </xs:appinfo>
</xs:annotation>
<xs:sequence>
 <xs:element name="pub_id">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:maxLength value="4"/>
    <xs:pattern value="[0-9]{4}"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="name">
  <xs:simpleType>
```

```
   <xs:restriction base="xs:string">
    <xs:maxLength value="40"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="city" minOccurs="0">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:maxLength value="20"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:element>
 <xs:element name="country" type="xs:string" minOccurs="0">
</xs:sequence></xs:complexType></xs:schema>
```

7 Conclusion

Dans cet article, nous avons présenté une solution pour faciliter la transformation de modèles entre UML et XML Schema dans le contexte du Master Data Management. Le processus de transformation permet de générer automatiquement un modèle MDM, dans notre contexte XML Schema, à partir d'un modèle UML. La démarche suivie se fonde sur les profils UML présentés et des formalismes standard de transformation tels que MOFScript et XSLT. Ces profils représentent l'ensemble des concepts statiques des plateformes cible à savoir XML Schema et les modèles appliqués au Master Data Management. Ensuite, les règles de transformation mises en place ont été implémentées. Couplés à des méthodes de transformation, l'utilisation de ces profils UML permet de s'abstraire de toute spécificité technique liée à la définition des modèles XML Schema appliqués au MDM. Notre méthodologie s'applique aussi bien à des modèles XML Schema génériques qu'à des modèles XML Schema appliqués au MDM. De ce fait, notre méthodologie est apte à s'appliquer à tous les domaines se fondant sur des modèles XML Schema. Cependant, nous n'avons pas abordé les problématiques de validation incrémentale de modèles afin d'optimiser les processus de validation durant les phases de conception. En effet, la cohérence structurelle et les processus de validation représentent des aspects essentiels dans les phases de définition de modèles de données. Les approches classiques de validation de modèle (Nentwich et al., 2003) consistent à vérifier l'intégralité d'un modèle. Lorsqu'un modèle est modifié, il est nécessaire de valider à nouveau l'ensemble du modèle pour vérifier si la modification apportée n'a pas entraînée une incohérence dans la structure du modèle. Cette approche est convenable dans des situations de modélisation de modèles de taille raisonnable mais nous pouvons entrevoir que ce processus n'est pas envisageable dans des contextes industriels et lors d'un passage à l'échelle, caractéristiques indissociables dans un cadre d'application du Master Data Management. La principale cause de ce problème est que les informations issues des contrôles effectués lors de chaque modification unitaire ne sont pas exploitées lors des processus ultérieurs de validation globale. Une étude sur la mise en place d'approche incrémentale de validation de modèles est une suite logique à notre approche pour optimiser une architecture de modélisation.

Références

Abitboul, S., S. Cluet, G. Ferran, et M.-C. Rousset (2002). The xyleme project. *Computer Networks 39*(3), 225–238.

Bernauer, M., G. Kappel, et G. Kramler (2004). G.k. : Representing xml schema in uml - a uml profile for xml schema. Technical report, business Informatics Group, Vienna University of Technology, Austria.

Bézivin, J. et O. Gerbé (2001). Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE'01, Washington, DC, USA, pp. 273–. IEEE Computer Society.

Booch, G., M. Christerson, M. Fuchs, et J. Koistinen (1999). Uml for xml schema mapping specification. *Rational Software Corp. and CommerceOne Inc.*.

Bosak, J., T. Bray, D. Connolly, E. Maler, G. Nicol, C. M. Sperberg-McQueen, L. Wood, et J. Clark (1998). Dtd specification, revision 1.2. Standard document, OMG.

Carlson, D. (2006). Semantic models for xml schema with uml tooling. In *Proceedings of the 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Athens, USA.

Carlson, D. A. (2001). *Modeling XML Applications with UML : Practical e-Business Applications*. Addison Wesley Professional.

Clark, J. (1999). Xml path language (xpath) version 1.0. W3c recommendation, W3C.

Clark, J. et S. DeRose (1999). Xsl transformations (xslt). W3c recommendation, W3C.

Conrad, R., D. Scheffner, et J. Freytag (2000). Xml conceptual modeling using uml. In *Proceedings of the 19th International Conference on Conceptual Modeling (ER'2000)*, pp. 291–307. Lecture Notes in Computer Science, Volume 1920/2000.

Dominguez, E., J. Lloret, B. Pérez, A. Rodriguez, A. L. Rubio, et M. A. Zapata (2007). A survey of uml models to xml schema transformations. In *Proceedings of the Web Information Systems Engineering Conference (WISE'07)*, pp. 184–195. Lecture Notes in Computer Science, Volume 4831, Springer.

Domínguez, E., J. Lloret, B. Pérez, Á. Rodríguez, A. L. Rubio, et M. A. Zapata (2011). Evolution of xml schemas and documents from stereotyped uml class models : A traceable approach. *Information & Software Technology 53*(1), 34–50.

Gao, S., C. M. Sperberg-McQueen, et H. S. Thompson (2011). W3c xml schema definition language (xsd) 1.1 part 1 : Structures. W3c candidate recommendation, W3C.

Kurtev, I., K. Berg, et A. M. (2003). Uml to xml-schema transformation : a case study in managing alternative model transformations in mda. In *Forum on specification and Design Languages*.

Levendovszky, T., G. Karsai, M. Maroti, A. Ledeczi, et H. Charaf (2002). Model reuse with metamodel-based transformations. In *Proceedings of the 7th International Conference on Software Reuse : Methods, Techniques, and Tools*, pp. 166–178.

Menet, L., M. Lamolle, et A. Zerdazi (2008). Managing master data with xml schema and uml. In *Proceedings of International Workshop on Advanced Information Systems for Enterprises (IWAISE'08)*, pp. 53–59. IEEE Computer Society.

Narayanan, K. et S. Ramaswamy (2005). Specifications for mapping uml models to xml schemas. In *Proceedings of the 4th Workshop in Software Model Engineering (WiSME'05)*, Montego Bay, Jamaica.

Nentwich, C., W. Emmerich, et F. A. (2003). Flexible consistency checking. *ACM Transactions on Software Engineering and Methodology 12*(01), 28–63.

Oldevik, J. (2011). Mofscript user guide version 0.9 (mofscript 1.4.0). W3c recommendation, W3C.

OMG (2011a). Meta object facility (mof) 2.0 query/view/transformation specification. Standard document formal/2011-01-01, OMG.

OMG (2011b). Xmi specification, v2.4.1. Standard document, OMG.

Routledge, N., L. Bird, et A. Goodchild (2002). Uml and xml schema. In *Proceedings of the 13th Australasian Database Conference*, Melbourne, Australie, pp. 157–166.

Sen, S., N. Moha, V. Mahé, O. Barais, B. Baudry, et J.-M. Jézéquel (2012). Reusable model transformations. *Software and Systems Modeling 11*, 111–125.

Wu, I. et S. Hsieh (2002). An uml-xml-rdb model mapping solution for facilitating information standardization and sharing in construction industry. In *Proceedings of the National Institute of Standards and Technology*, Gaithersburg, USA.

Summary

In this paper, we propose a formalism (based on standards), which allows to people to be involved in Master Data Management design with no knowledge about technical specificities of the target platform. Profiles are defined to extend the UML semantic to the MDM semantic based on XML Schema profile. The integration of these profiles is made in the context of EBX5 platform, which is a XMLWare.