Managing Master Data with XML Schema and UML

Ludovic Menet, Myriam Lamolle, Amar Zerdazi

Managing Master Data with XML Schema and UML

Ludovic Menet
Orchestra Networks
75 boulevard Haussman
75008 Paris. France
ludovic.menet@orchestranetworks.com

Myriam Lamolle, Amar Zerdazi
IUT de Montreuil. Université de Paris 8.
140 rue de la nouvelle France
93100 Montreuil. France
{m.lamolle, a.zerdazi}@iut.univ-paris8.fr

Abstract

To manage parameters, the majority of Information Systems is concerned by heterogeneity in both data and solutions. Consequently, the management of this data becomes complex, inefficient, insecure and expensive. The need to use a structured formalism to handle complex data appears. We suggest a data integration solution based on a XML architecture. This architecture embeds a Master Data Management in the Information System. The unification of Master Data is primarily done by the definition of models. These models are XML Schema documents describing complex data structures. We propose to enrich the structure and the semantics of these models by defining a metamodel. In the metamodel, we introduce semantic object relations for defining links between concepts. The resulting metamodel is used to define an UML profile and to optimize operations such as models validation, data factorization and trees representation. Moreover, UML profile is exploited to make easier the definition of models.

1. Introduction

In the frame of the interoperability of heterogeneous data sources, two main data integration approaches exist: the virtual approach (or mediator) [1] and the materialized approach (or data warehouse) [2]. We suggest an implementation of the second approach by an XML [3] architecture called EBX.Platform. This architecture allows companies to unify the management of their strategic data without any changes in their databases or their existing applications. This unification is conducted in three ways: **(i)** definition of the main data model through the XML Schema language, **(ii)** persistence in a common repository, specific to the product, in a remote or integrated database, **(iii)** availability of a generic user-friendly web tool interface for data consulting, for updating and for synchronizing the repository with the Information System of the companies.

One of the major benefits of EBX.Platform for companies is that the repository manages the inheritance of instances. The abilities of data factorization brought by inheritance and by EBX allow data duplication and related problems (costs and risks) to be avoided. To implement the mechanism of inheritance, a first conceptual model has been realized. This paper presents our XML solution of data integration and the improvement of the conceptual model by the definition of an object metamodel. The resulting metamodel is used for defining an UML profile which enables us to describe a formal approach of design of Master Data models.

2. EBX.Platform

The company Orchestra Networks proposes Master Data Management software called EBX.Platform. Based on Java and XML Schema, EBX.Platform is a standard and non-intrusive solution that helps companies unify and manage their reference business data and parameters across their Information System.

2.1. Master Data Management (MDM)

The Master Data Management is a way to unify, manage and integrate references data across the Information System of the company. These data can be of several kinds:

- Products, services, offers, prices.
- Customers, providers.
- Lawful data, financial data.
- Organizations, structures, persons.

Currently, the majority of Information Systems is concerned by heterogeneity in both data and solutions. In this framework, there are three kinds of heterogeneity:

- Heterogeneity of storage systems (databases, directories, files…).
- Heterogeneity of formats of data (files owner, XML documents, tables…).
- Heterogeneity of solution to manage the different types of data.

Consequently, the management of the data becomes complex, insecure, inefficient and expensive. Moreover, using different applications to manage this diversity involves some redundancy in both data and tools.

An Information System without Master Data Management (MDM) presents some issues such as:

- No unified vision of the references data.
- Duplicated data in several systems.
- No coherence between companies and subsidiary ones.
- No single tools for users.

EBX.Platform is a MDM solution based on powerful concepts to solve these problems.

2.2. EBX.Platform's concepts

EBX.Platform is based on two concepts: **(i) an adaptation model** which is a data model for a set of Master Data. It is an XML Schema [3] document and **(ii) an adaptation** which is an XML instance of the adaptation model which contains Master Data Values. Using XML Schema allows each node of the data model corresponds to an existing data type according to the W3C standard [3] to be specified. EBX.Platform supports the main XML Schema datatypes, as well as multi-occurrence complex types. Indeed, the XML Schema formalism allows us to define constraints (enumeration, length, lower and higher limit, etc.), information about adaptation and its instances (access connector, Java factory class, access restriction, etc.) and layout information (label, description, formatting…) to be specified for each node of the schema. For each node of the adaptation model, declared possible instances, corresponds a node in the adaptation. If an adaptation model has several adaptations, we consider that an adaptation tree is handled. The figure 1 presents an adaptation model and its instances as an adaptation tree.

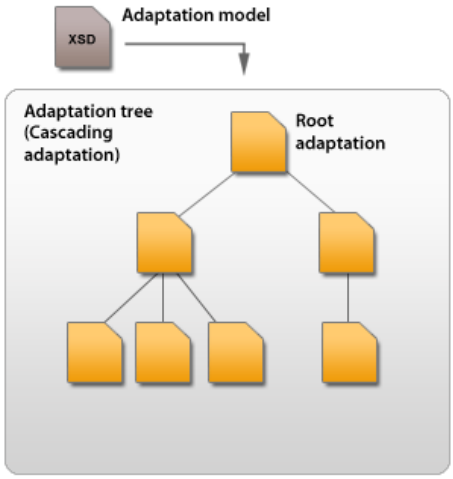

Figure 1. An adaptation model and its instances.

In an adaptation, each node has the following properties: **(i) An adaptation value**; if this value is not defined in the current adaptation then it is inherited from its ancestor (parent adaptation), recursively. If no ancestor defines a value, then the value is inherited, by default, from the data model. **(ii) An access right** for descendants; the adaptation node can be either hidden (to descendants), in read only (for the descendants), or in read/write (for the descendants).

These powerful concepts are deployed on a specific architecture made of several components. The figure 2 presents the architecture of EBX.Platform.

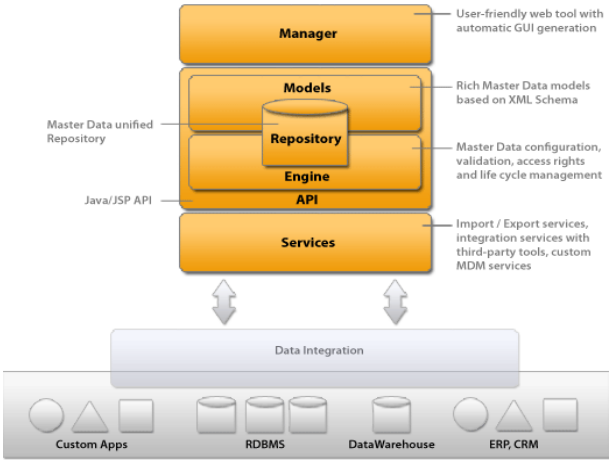

Figure 2. Architecture of EBX.Platform.

The architecture presented in the figure 2 is based on three important components:

- EBX.Platform Engine is based on a technology that allows to manage multiple instances of Master Data in a core repository. EBX.Platform Engine main features are data validation, data configuration, life cycle management and access rights management.
- EBX.Platform, with EBX.Manager, provides both business and technical users with a Web-based tool for Master Data Management. EBX.Manager dynamically generates a rich User Interface from Master Data models without any programming. The figure 3 shows the graphical interface generated from an adaptation model.

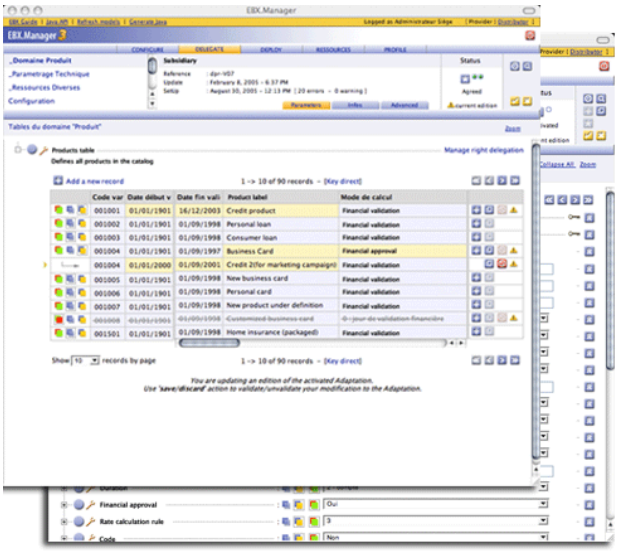

Figure 3. EBX.Manager web-based tool.

• EBX.Platform services allows to integrate Master Data with Information Systems. It provides import/export features and integration with third party tools, such as EAI, ETL, ESB, directories. Custom MDM services can be developed using a standard Java API. Indeed, using our Java API it is possible to integrate new features in EBX.Platform. For example, services can be used to perform some reporting, data historization, management of processes etc... In the figure 4, we have illustrated the use of services defining a workflow engine.

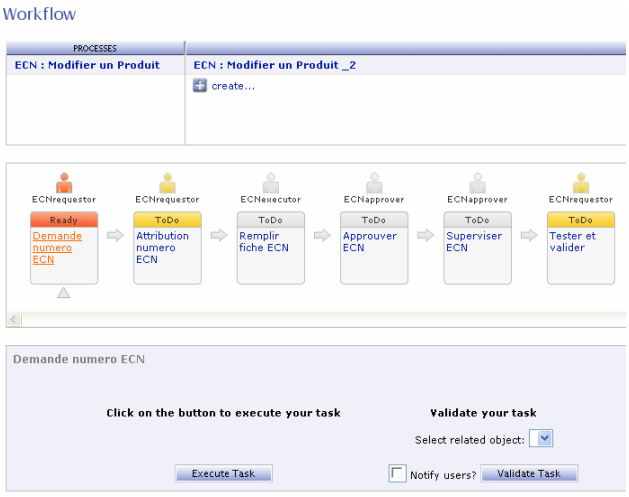

Figure 4. Example of a workflow service.

In this custom service, it is possible to define tasks for users. The mechanism of workflow enables to define ordered tasks to be performed by users. Each task of the process has to be fully realized before the next one. For example, this service can be used for the management of projects where tasks are assigned to teams in a precise order.

Our solution presents a way to unify master data from several data sources. The existing works on data integration [6] [7] [8] [13] [14] are focused on conceptual models without semantic relations between concepts; in the extension of our works, we propose to enrich the metamodel of the adaptation models adding some specifics objects concepts.

3. Meta-modelling "object"

We introduce object features which are added to the conceptual model and we propose a metaschema of an adaptation model in order to consolidate both the conceptual model and the existing data validation. Our first goal is to add object metadata to the adaptation model. We can use the following notions in terms of relations between objects such as generalization, specialization and dependence (aggregation or composition). To illustrate these concepts, let's take an example frequently used in some UML [5] academic cases. This example defines five concepts, *Person*, *Teacher*, *Student*, *University* and *Department*. These concepts are semantically linked. More precisely the concept *Person* is the generalization of *Student* and *Teacher*, and the concept *University* is a composition of *Departments.* These semantic links have strong impacts in data factorization and optimization. Let us consider in our example the notion of dependence (more precisely the composition) between *University* and *Department* concepts. The composition implies that there cannot be instances of the *Department* concept without instances of the *University* concept. An optimization can be made for the instance deletion process; the deletion of an instance of the *University* concept implies that all dependent instances (*departments*) will be removed. However, in the case of an aggregation between these concepts, the aggregated instances (*departments*) will not be deleted if they are used by other concepts.

Generalization and *specialization* relations are used to factorize data in our system. In the generalization case, common attributes are gathered in a general concept. For example, attributes such as *first name* and *last name* are common to the concepts *Student* and *Teacher.* These two attributes are migrated to the concept *Person* to factorize data, avoiding duplication of their definition in the concepts *Student* and *Teacher.* In the same way as [12], we propose some metadata to be included in the XML schema to implement these notions. As W3C suggests it, the XML Schema extensions that we add are defined in the « *appInfo* » element (labelled *concept_object_name* at the third line of the figure 5).

```
…<xs:annotation>
      <xs:appinfo>
            <osd:concept_object_name/>
      </xs:appinfo>
</xs:annotation>…
```

Figure 5. XML Schema extension representing an object relation.

We propose to apply our extensions on the previous example. The figure 6 represents an UML diagram defining the relations of our example.

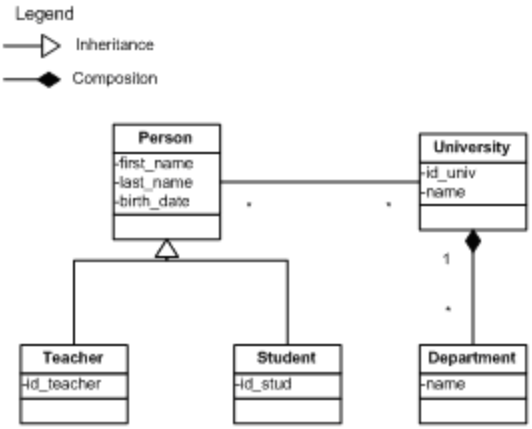

Figure 6. UML class diagram.

In this diagram the *University* concept has a composition relationship with the *Department* concept. The composition is represented in an adaptation model as the following:

```
…
<xs:complexType name=”University”>
    <xs:sequence>
        <xs:element name=”dept” type=”Department”>
            <xs:annotation>
                <xs:appinfo>
                    <osd:composition/>
                </xs:appinfo>
            </xs:annotation>
        </xs :element>
    …
    <xs:sequence>
</xs:complexType> …
```

Figure 7. Definition of a composition in an adaptation model.

The figure 8 presents the relation of specialization between the *Teacher* concept and *the Person* concept.

```
…
<xs:complexType name=”Teacher”>
    <xs:sequence>
        <xs:element name=”relation” type=”Person”>
            <xs:annotation>
                <xs:appinfo>
                    <osd:specialization/>
                </xs:appinfo>
            </xs:annotation>
        </xs:element>
    …
    <xs:sequence>
</xs:complexType> …
```

Figure 8. Definition of a specialization in an adaptation model.

By defining XML Schema extensions in the elements *annotation* and *appInfo*, it is not necessary to provide a schema allowing the structure of these extensions to be de-fined. One of the resulting issues is that the validation of these extensions is fully delegated to the validation engine of EBX.Platform, and not to the XML Schema engine. We have defined a metaschema of an adaptation model describing the structure of the concepts provided by EBX.Platform to avoid this issue.

The existing works [6] [7] [8] based on data warehouses define several way of integration of data sources. In our solution this integration is done across the definition of a global data model. This one can be realized automatically or manually using some particular technologies implying a wide knowledge about them. We propose to use formalism, such as UML, for defining a data model. Moreover, the adding of objects features, that we have realized, allows us to use the semantic functionalities of UML for the definition of models.

4. Definition of an UML profile

The definition of an adaptation model is made through the XML Schema technology. The use of XML is adapted to the needs of EBX.Platform which implies a wide knowledge of this language. There are many XML Schema tools but on the one hand the user can use the XML Schema features which are not implemented by EBX and on the other hand he is not guided about the extensions of EBX. As a result, formal-ism must be implemented to make this modeling easier. In addition to its modeling abilities, UML allows profiles [4] [5] to be defined. A profile specializes the UML formalism for an application field or a particular technology. Many profiles have been developed for several goals, for example a profile for expressing all the semantic of CORBA [9], a profile defining the features of EJB, or a profile for the persistence of semantics of our metamodel.

The adequate process for the definition of a profile is as the following:

− Definition of the domain of the profile, i.e. the metamodel defining the concepts and the relations between these ones.

− Technical definition of the profile establishing the matching between UML concepts and the ones defined in the profile.

− Definition of an example simple and concrete of the profile.

4.1. Definition of the domain of the profile

The first step has been realized previously by defining the metaschema of an adaptation model. This metaschema is used to describe the technical definition of our profile.

4.2. Technical definition of the profile

The table 1 presents the technical definition of our UML profile.

Stereotype	Applied to	Description
AdaptationModel	Class	Defines an adaptation model.
Root	Class	Defines the root of an adaptation model.
Sequence	Class	Defines a XML Schema complex element of type <xs:sequence>.
SimpleType	Class	Defines an XML Schema redefined type.
Table	Class	Defines a table element.

Table 1. Extract of the technical definition of the UML profile representing EBX.Platform metamodel

The technical definition is used to realize the UML profile of the adaptation models.

4.3. Concrete definition of the UML profile

The figure 9 presents a piece of the UML profile defining the relations between the different concepts introduced by EBX.Platform.

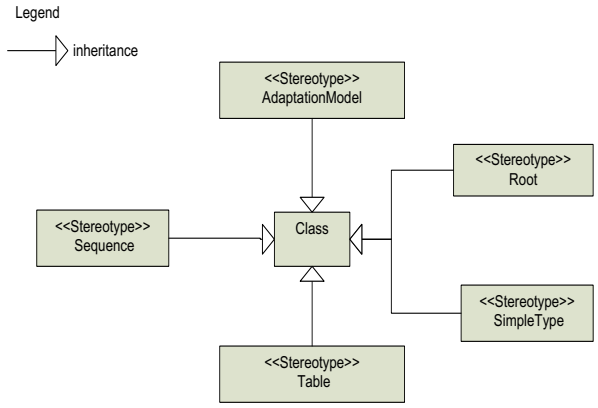

Figure 9. Piece of UML profile representing EBX.Platform metamodel.

Using UML extension mechanism enables us to extend the UML formalism to our semantic. This extension is performed using *stereotypes* and *marked values*. Stereotypes are employed to define a new type of element from an existing one. We can see on the figure 9 that the stereotypes (labelled <<Stereotypes>>) inherit from the *Class* element of the UML metamodel. As a consequence, the stereotypes will be instantiated from the metamodel constructor in the same way that the *Class* element. Tagged values specify keyword-value pairs of model elements to set properties for existing elements or for stereotypes. The definition of these stereotypes allows to introduce more semantic, out of the concepts defined in UML, that will let us to define an adaptation model with an UML diagram.

4.4. Application of the UML profile

This section illustrates a piece of an UML model defined with our profile and its equivalent in XML Schema. The figure 10 represents the UML modeling of an adaptation model defining a simplified network train. This model is composed of concepts and relations between these ones. It is question in our example to highlight the objects relations that we have previously introduced, and some adaptation model specificities such as redefined types. Thus we have defined, in our example, the *Train* concept being composed of an *engine, wheels* (notion of composition), being able to have *wagons* (notion of aggregation) and having some properties such as a *type* and a *trademark*. We associate a driver of type *Person* to a train. The *Person* concept defines properties such as, first name, last name and a birth date. We have also defined the property *email* representing the use of a redefined type. The *Email* class uses the stereotype *<<SimpleType>>* allowing to indicate that it is a redefined type within the meaning of XML Schema. The properties of this redefined type are defined in an UML

annotation specifying values for *SimpleType::base*, *SimpleType::pattern* and *SimpleType::whitespac*e. The root of the schema is materialized by the stereotype <<*Root*>>, applied to the *NetworkTrain* class. The stereotype <<*AdaptationModel*>>, applied to the AdaptationModelRoot class, indicates the definition of an adaptation model. The classes defined in this model, excepted for the class *AdaptationModelRoot*, are also stereotyped <<*Sequence*>>. This stereotype specifies that the corresponding class represents a complex element, within the meaning of XML Schema, of the type <xs:sequence>.

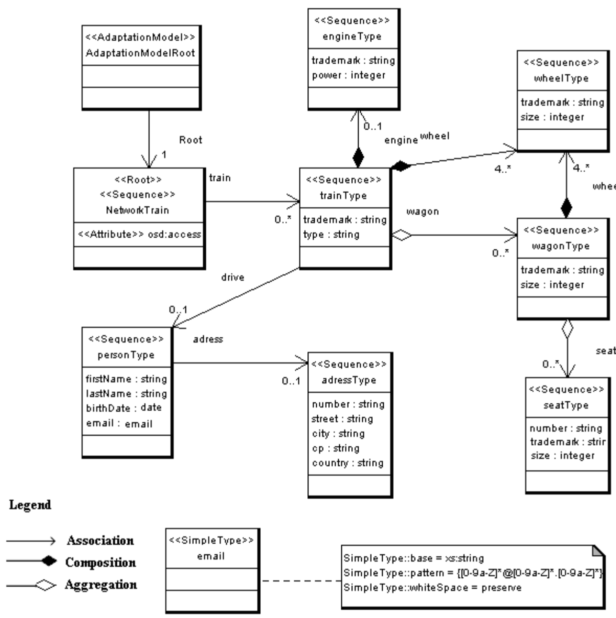

Figure 10. Definition of an adaptation model with UML.

The corresponding adaptation model defining in XML Schema is shown if the figure 11.

```
<!-- Start of  the adaptation model -->
<xs:schema xmlns:fmt="urn:ebx-schemas:format_1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Root of the adaptation model  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<xs:element name="NetworkTrain" osd:access="--">
  <xs:complexType>
    <xs:sequence>
    <xs:element name="train" type="trainType"
minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : trainType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--trainType -->
<xs:complexType name="trainType">
  <xs:sequence>
    <xs:element name="trademark" type="xs:string"/>
    <xs:element name="type" type="xs:string"/>
    <xs:element name="wagon" type="wagonType"
maxOccurs="unbounded">
          <xs:annotation>
                    <xs:appinfo>
                              <osd:aggregation/>
                    </xs:appinfo>
          </xs:annotation>
    </xs:element>
    <xs:element name="engine" type="engineType">
          <xs:annotation>
                    <xs:appinfo>
                              <osd:composition/>
                    </xs:appinfo>
          </xs:annotation>
    </xs:element>
    <xs:element name="wheel" type="wheelType"
minOccurs="4" maxOccurs="unbounded">
          <xs:annotation>
                    <xs:appinfo>
                              <osd:composition/>
                    </xs:appinfo>
          </xs:annotation>
    </xs:element>
    <xs:element name="drive" type="personType"/>
  </xs:sequence>
</xs:complexType>
<!--End train -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : wagonType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- wagonType -->
<xs:complexType name="wagonType">
  <xs:sequence>
    <xs:element name="trademark" type="xs:string"/>
    <xs:element name="size" type="xs:integer"/>
    <xs:element name="wheel" type="wheelType"
minOccurs="4" maxOccurs="unbounded">
          <xs:annotation>
                    <xs:appinfo>
                              <osd:composition/>
                    </xs:appinfo>
          </xs:annotation>
    </xs:element>
    <xs:element name="seats" type="seatType"
maxOccurs="unbounded">
          <xs:annotation>
                    <xs:appinfo>
                              <osd:aggregation/>
```

```
                    </xs:appinfo>
            </xs:annotation>
      </xs:element>
   </xs:sequence>
</xs:complexType>
<!-- End wagonType -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : engineType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- EngineType -->
<xs:complexType name="engineType">
   <xs:sequence>
        <xs:element name="trademark" type="xs:string"/>
        <xs:element name="power" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>
<!—End engineType -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : wheelType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- wheelType -->
<xs:complexType name="wheelType">
   <xs:sequence>
        <xs:element name="trademark" type="xs:string"/>
        <xs:element name="size" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>
<!—end wheelType -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : personType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- personType -->
<xs:complexType name="personType">
   <xs:sequence>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="lastName" type="xs:string"/>
        <xs:element name="birthDate" type="email"/>
        <xs:element name="email" type="xs:date"/>
        <xs:element name="address" type="addressType"/>
   </xs:sequence>
</xs:complexType>
<!—End personType -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : addressType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- adresse -->
<xs:complexType name="addressType">
   <xs:sequence>
      <xs:element name="number" type="xs:string"/>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="cp" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<!-- adresse -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : seatType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- seatType -->
<xs:complexType name="seatType">
   <xs:sequence>
      <xs:element name="numero" type="xs:string"/>
      <xs:element name="marge" type="xs:string"/>
      <xs:element name="taille" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<!-- end seatType -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Class : emailType  -->
<!--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!--Email Type -->
<xs:simpleType name="email">
<xs:restriction base="xs:string">
<xs:whiteSpace value="preserve"/>
<xs:pattern value="{[0-9a-Z]*@[0-9a-Z]*.[0-9a-Z]*}"/>
</xs:restriction>
</xs:simpleType>
<!-- End of email type -->
<!-- End of the adaptation model -->
</xs:schema>
```

Figure 10. Definition of an adaptation model with XML Schema.

By defining a metamodel, we improve the process of data validation which enables us to use the XML schema validation engine in an automatic way. Moreover, the definition of a UML profile allows us to ensure that the designer uses the semantic strictly defined by EBX.Platform, avoiding some XML Schema specificities which are not managed. We associate a generator of XML Schema code to this UML profile which makes the definition of an adaptation model easier and more secure.

5. Conclusions and perspectives

We propose a generic solution of data integration based on the XML technology. Indeed, our solution is able to integrate data from several kinds of sources such as databases or XML documents. We have seen that every adaptation model is an XML Schema standard document. Some XML Schema features are difficult to handle for the designer of models. Graphical XML Schema tools exist but they are not restricted to our semantic and they cannot be aware about the extensions brought by EBX. So, we have proposed a metamodel to validate adaptation models in a transparent way by using the XML Schema validation engine. We have also presented how the use of UML

profiles makes easier the definition of an adaptation model.

The follow-up to our work will be the enrichment of our metamodel for the Master Data Management module allowing semantic constraints to be expressed and validated according to profiles (for example business language). We will develop two ways. The first one is the modelling methods and constraint expression (expression of facets). Formalism design as UML will allow EBX schema modelling. The created schema is validated by this modelling based on rules. The second one is the integration of constraint expression according to profiles (constraints on types or between concepts). This notion supports a semantic to the represented concepts and expressions of dependency.

We will take into account the ODMG [10] standard features (ODMG, 1999) to EBX formalism (inheritance notion about the models directly specified in the schema, such as specialization, generalization, dependence, etc.), the UML formalism, the ad-vantages of OWL language dedicated to the ontology definition [11], and the advantages of conceptual graphs for the expression of relations and of constraints between concepts. The features of OWL and of conceptual graphs will be used to perform inferences on data allowing some optimizations.

6. Acknowledgements

We would like to thank Orchestra Networks for there support to our research.

7. References

[1] Lenzerini M. (2002), Data Integration: A Theoretical Perspective, 21st ACM SIGMOD International Conference on Management of Data / Principles of Database Systems (PODS'02), p. 233-246.

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

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

[4] Mahmoud N. (2003) VUML: a Viewpoint oriented UML Extension, 18th IEEE International Conference on Automated Software Engineering (ASE'03), pp. 373-376.

[5] Pilone D., Pitman N. (2006), UML 2.0 in a Nutshell, O'Reilly (Eds).

[6] Baril X. and Bellahsène Z. (2003). XML Data Management: Native XML and XML-Enabled Database Systems. Chapter Designing and Managing an XML Warehouse, pp.455–474. Addison Wesley Professional.

[7] Gofarelli, M., Rizzi, S., Vrdoljak, B. (2001). Data Warehouse Design from XML Sources, In Proc. The 4th ACM Intl Workshop on Data Warehousing and OLAP (DOLAP01), pp. 4047, Atlanta, 2001.

[8] Design of XML Document Warehouses, In Proc. Data Warehousing and Knowledge Dis-covery, 6th International Conference, DaWaK 2004, pp. 114, Zaragoza, Spain, 2004.

[9]OMG Document. "CORBA specifications". http://www.omg.org/technology/documents/formal/profile_corba.htm.

[10] ODMG (1999) The Object Data Standard: ODMG 3.0, Morgan Kauffman Publishers.

[11] Kalfoglou Y. Schorlemmer M. (2003) Ontology mapping: the state of the art. The Knowledge Engineering Review 18(1) pp. 1-31.

[12] ZERDAZI A. LAMOLLE M (2005) Modélisation des schémas XML par adjonction de métaconnaissances sémantiques, 2ème rencontre des Sciences et Technologie de l'Information, ASTI, Clermont-Ferrand.

[13] S. Abiteboul, S. Cluet, G. Ferran, M.-C. Rousset. « The Xyleme Project ». Computer Net-works 39, 2002.

[14] 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.