# Mapping Paradigm for Document Transformation

Arnaud Blouin, Olivier Beaudoux

# Mapping Paradigm for Document Transformation

Arnaud Blouin and Olivier Beaudoux
ESEO - GRI
Angers,
France
{arnaud.blouin, obeaudoux}@eseo.fr

## ABSTRACT

Since the advent of XML, the ability to transform documents using transformation languages such as XSLT has become an important challenge. However, writing a transformation script (e.g. an XSLT stylesheet) is still an expert task. This paper proposes a simpler way to transform documents by defining a relation between two schemas expressed through our mapping language. And then by using a transformation process that applies the mapping instances of the schemas. Thus, a user only needs to focus on the mapping without having any knowledge about how a transformation language and its processor work. This paper outlines our mapping approach and language, and illustrates them with an example.

## Categories and Subject Descriptors

I.7 [**Document and text Processing**]: Document Preparation—*Markup languages*

## General Terms

Design, languages

## Keywords

XML, document transformation, mapping, XSLT

## 1. INTRODUCTION

XML [9] is now a standard for storing or organizing data. In order to make these XML data interoperable, other standards like XSLT have been defined; XSLT [10] is a programming language specifically designed for XML transformation being, thereby, a medium of communication between applications. XSLT is now a widely used transformation language in spite of some limitations: we can hardly know if two XSLT stylesheets give the same result [7] or modify incrementally a presentation [8, 1]. Moreover, the process of mapping and the process of transformation are not clearly separated making the development of an XSLT program (usually called an XSLT stylesheet) more difficult than it should be.

Mapping brings interoperability between heterogeneous data and applications by establishing correspondences between two schemas. Whereas a transformation process concerns the transformation of two instances of these schemas [4], as depicted in figure 1.

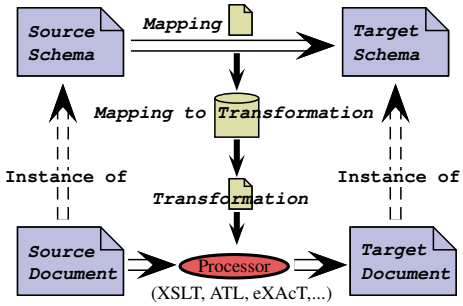


**Figure 1: From Mapping to Transformation**

In a transformation program such as an XSLT stylesheet, the separation between these two processes is not apparent. Our work aims to propose a mapping language that allows a user to write a mapping between two schemas without needing the level of programming skills needed for XSLT stylesheets since a mapping is a specification disjoined from the (often complex) transformation process. Research has already been carried out to avoid the direct use of transformation languages such as XSLT. Pietriga et al. present in [5] a visual approach to XML transformation. This point of view is very attractive since it reduces user's cognitive load. However, it has the drawback to create complex graphics for complex transformations. In the mapping domain, Clio [6] is a tool that allows a user to graphically define a mapping between relational schemas; its mapping language has the main drawback to not allow operations between associations of the two concerned schemas.

The first part of this paper introduces our proposed approach and is following by a simple example illustrating it.

## 2. PROPOSED APPROACH

The mapping concept is already used in the database domain to facilitate the integration and the management of databases [2] [6]. However, there is no simple schema-mapping language allowing the definition of mappings between UML class diagrams. Our approach aims to define a mapping language which would be easily integrated with UML editors. By specifying only the process of mapping between two schemas, we are independent of the transformation processor that could be an XSLT, ATL [3], or eXAcT processor. It allows an XSLT, ATL or eXAcT non-expert to easily establish a mapping between two schemas, that will produce a transformation between an instance of each schema.

A **schema** is a set of classes and associations between classes. A class is defined by its unique name within the schema and includes a set of attributes. An association *Bs* from a class *A* to a class *B*

represents the set of the B's instances involved in the relation; we note *|Bs|* the multiplicity of *Bs*. A schema can be defined in different formats such as XML-Schema, or XMI. Given two schemas $E$ and $F$, a **mapping** is an application $f$ from $E$ to $F$. A mapping can also be defined as a set $M = \{f_1, f_2, \ldots, f_n\}$ where each $f_i$ is a sub-mapping defining a part of the correspondence between the schemas $E$ and $F$. Consequently, a **sub-mapping** is an application $f'$ from $E' \subset E$ to $F' \subset F$. In our framework, our mapping language describes a mapping from one schema to another defined by a set a sub-mappings.

Syntactically, our mapping language is composed of a set of sub-mappings contained in a main mapping that defines the schemas to use. Each sub-mapping defines relations between the implicated components *via* instructions.

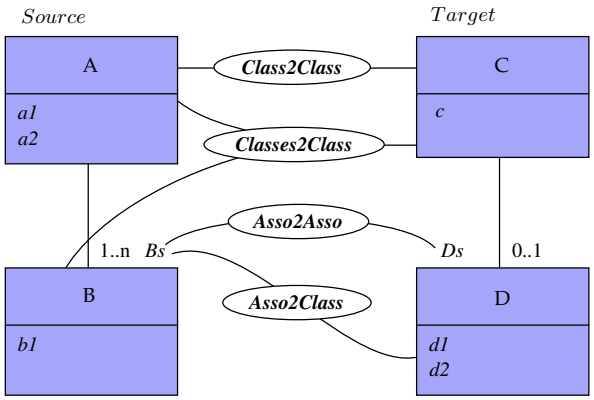


**Figure 2: Different kinds of mappings**

Sub-mappings can be classified in categories. Four of them are represented in figure 2. The two most important are the **association to association** *(Asso2Asso)* sub-mapping and the **class to class** (*Class2Class*) sub-mapping. The goals of an *Asso2Asso* sub-mapping are twofold: it defines both the multiplicity of the output association and the position of each object. For example, given two associations *Es* and *Fs* with $|Es| = 0..n$ and $|Fs| = 0..n$ , a sub-mapping from $Es$ to $Fs$ could be as follows:

```
1: Es -> Fs
2: {
3:    |Es| -> |Fs|
4:    Es@i -> Fs@i
5: }
```

The first line specifies that the sub-mapping concerns the relation from $Es$ to $Fs$. Line 3 defines the multiplicity of the relation $Fs$ ($|Fs|$) as being linked to the multiplicity of $Es$. The next line establishes the order of each object related to *Fs*: for each object related to $Es$ at the position $i \in [0, n]$, an object exists within $Fs$ at the same position $i$. More complex operations can be carried out during this step; for example we can define the order of the objects related to *Fs* by inverting these related to *Es*:

```
invert(Es)@i -> Fs@i
```

*invert(Es)* returns the list of the objects related to *Es* in the reverse order.

A *class2class* sub-mapping defines the relation between the attributes of the two given classes. For example, given two classes *G* and *H*, where *g1* and *g2* are two attributes of *G* and *h1* and *h2* two of *H*. A possible sub-mapping from *G* to *H* could be:

```
1: G -> H
2: {
3:    min(g1,g2) -> h1
4:    max(g1,g2) -> h2
5: }
```

*min(g1,g2) -> h1* means that the minimum between *g1* and *g2* is linked to *h1*.

Mixes of these two kinds of sub-mappings can be used to create more complex sub-mappings; for example we can define a *n-classes to class* or an *association to class*. Complex features are notably conditions that can be used into sub-mapping instructions. The following code sample defines the syntax of conditions.

```
1: Is -> J, K
2: {
3:   |Is|=0:
4:      0    -> J.value
5:   |Is|>0:
6:      |Is| -> K.value
4: }
```

Given an association *Is* and two classes *J* and *K*, and a mapping from *Is* to *J* and *K*. *J* and *K* are linked to Is respectively when the multiplicity of *Is* is, equal to 0, and greater than 0.

Our mapping language is not specified in XML. XML is a standard to store data but we believe it is not a good programming language notably because of its verbosity. The syntax and the grammar of a programming language must be defined according to its target domain (here the mapping domain) in order to be efficient and easy to use. However, we have planned to specify an XML version of our mapping language in order to be more easily transformed as is the case with the *Relax NG* and *Relax NG Compact* languages. For example, the "*mapping to transformation*" step described in figure 1, may be an XSLT stylesheet that transforms the XML version of a mapping into the document that will be used by the transformation processor.

## 3. EXAMPLE

Figure 3 presents a simple case of mapping from a library to a table; a library is defined by its name and contains documents described by their authors, title and year. A table contains lines where each line corresponds to a document and a header that defines the title of each column. Each line and the header have the same number of cells.

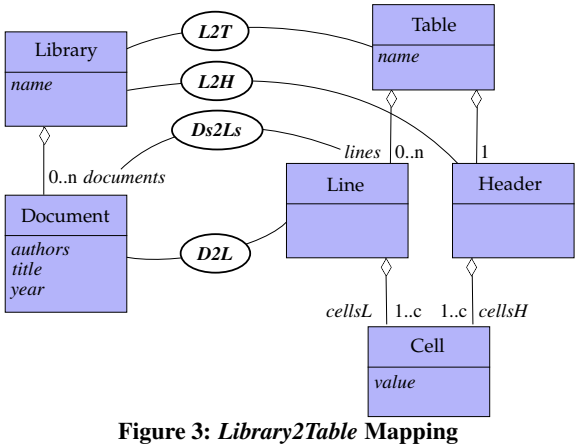


**Figure 3: *Library2Table* Mapping**

The following code corresponds to a possible mapping of figure 3. The main mapping *Library2Table* defines that the mapping is

from the schema *library.xsd* to the schema *table.xsd*. This mapping is composed of four sub-mappings; the sub-mapping *L2T* simply links the name of the library to the name of the table. By "*links*" we means for each *Library* a *Table* must exist. The sub-mapping *L2H* links the library to the header of the table: the multiplicity of the association *cellsHeader* takes the value *3* and the content of each cell is defined. We can notice that the *Library* of the sub-mapping *L2H* is never used in the instructions. A sub-mapping does not necessary use the source classes and associations in its instructions since their main utility is to be linked to the target classes and associations of the sub-mapping. Line 10, *Header.cellH* means that we can access to the associations related to a class from this class. The sub-mapping *Ds2Ls* is a *Asso2Asso* sub-mapping; as explained in the previous section, this kind of mapping defines the multiplicity of the association *lines* and defines the order of each object related to *lines*. The last sub-mapping *D2L* is a *Class2Class* sub-mapping that establishes a connection between a *Document* and a *Line*: the multiplicity of the association *cellsLine* takes the value *3* and each of these three cells is linked to an attribute of *Document*, i.e. *authors*, *title* and *year*.

```
1: Library2Table : library.xsd -> table.xsd
2: {
3:   L2T : Library -> Table
4:   {
5:     name -> name
6:   }
7:   L2H : Library -> Header
8:   {
9:     3 -> |Header.cellsHeader|
10:    "Authors" -> Header.cellsH@1.value
11:    "Title"   -> Header.cellsH@2.value
12:    "Year"    -> Header.cellsH@3.value
13:  }
14:  Ds2Ls : documents -> lines
15:  {
16:    |documents| -> |lines|
17:    documents@i -> lines@i
18:  }
19:  D2L : Document -> Line
20:  {
21:    3 -> |Line.cellsLine|
22:    Document.authors -> Line.cellsL@1.value
23:    Document.title   -> Line.cellsL@2.value
24:    Document.year    -> Line.cellsL@3.value
25:  }
26: }
```

Thanks to the distinct separation of each sub-mapping, we think that our mapping language has the advantage to be more readable than an XSLT or an eXAcT program. Moreover, it could be graphically depicted within UML class diagrams and seems easier to implement and to understand for a beginner than an XSLT stylesheet. But above all, our mapping process can be used behind different kinds of transformation processes such as XSLT, ATL or eXAcT, which is mainly due to the *declarative* approach of the language. Since a mapping operates at the schema level, transformed documents are always valid, which is not necessarily the case with transformation process.

# 4. CONCLUSION

In this paper, we propose the use of mapping in the context of document transformation. This approach facilitates the creation of a transformation by establishing the mapping between the concerned schemas and by being technologically independent from a transformation language and its processor. The next step of our work will be the implementation of our mapping language, the connection with a transformation process with the support of incremental transformation and user interaction. Tests on more realistic and complex cases will be done too.

# 5. REFERENCES


[1] O. Beaudoux. XML active transformation (eXAcT): transforming documents within interactive systems. In *DocEng '05: ACM symposium on Document engineering*, pages 146–148. ACM Press, 2005.

[2] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A vision for management of complex models. *SIGMOD Rec.*, 29(4):55–63, 2000.

[3] F. Jouault and I. Kurtev. Transforming models with ATL. In *Satellite Events at the MoDELS 2005 Conference*, pages 128–138. Springer, 2006.

[4] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy. Representing and reasoning about mappings between domain models. In *Eighteenth national conference on Artificial intelligence*, pages 80–86. American Association for Artificial Intelligence, 2002.

[5] E. Pietriga, J.-Y. Vion-Dury, and V. Quint. VXT: a visual approach to XML transformations. In *DocEng '01: Proc. of the 2001 ACM Symposium on Document engineering*, pages 1–10, 2001.

[6] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating web data. In *Proceedings of VLDB 2002*, pages 598–609, 2002.

[7] A. Trombetta and D. Montesi. Equivalences and optimizations in an expressive XSLT fragment. In *IDEAS '04: Proc. of the International Database Engineering and Applications Symposium*, pages 171–180. IEEE Computer Society, 2004.

[8] L. Villard and N. Layaïda. An incremental XSLT transformation processor for XML document manipulation. In *WWW '02: Proc. of the 11th international conference on World Wide Web*, pages 474–485, New York, NY, USA, 2002. ACM Press.

[9] W3C. Extensible markup language 1.1 specification. Technical report, W3C, 2006.

[10] W3C. XSL transformations (XSLT) version 2.0 recommendation. Technical report, W3C, 2007.