Model Transformation and Weaving in the AMMA Platform[1]

Marcos Didonet Del Fabro, Frédéric Jouault

ATLAS Group, INRIA and LINA, University of Nantes - 2, rue de la Houssinière, BP 92208 - 44322 - Nantes cedex 3, France
{marcos.didonet-del-fabro, frederic.jouault}@univ-nantes.fr

Abstract. Model transformation creates a set of target models from a set of source models. This operation is central in current MDE approaches. However, we highlight in this paper some of its limitations. Model weaving is another operation on models, which specifies typed links between model elements. Model weaving can thus be used to specify abstract model transformations as sets of links between source and target metamodel types. Therefore, we propose to combine model transformation and model weaving in order to overcome some of current model transformation limitations. This paper presents how we experimented this with ATLAS Transformation Language (ATL) and AMW (ATLAS Model Weaver) on an example.

1 Introduction

In current MDE (Model Driven Engineering) approaches, such as "Software Factories" by Microsoft [12] or MDA™ (Model Driven Architecture) by OMG [17], model transformation is one of the central operations. It consists in automatically generating a set of target models from a set of source models, each conforming to an explicit metamodel. Each transformation programs is also itself a model conforming to a metamodel, which corresponds to the transformation language in which it is written. The OMG issued the QVT (RFP) [19] to request transformation language proposals. Several answers were officially given, such as [21], [3] and [13]. There is, at the time this paper is being written, only one merged proposal remaining, from QVT-Merge group [20]. Other model transformation languages, such as ATLAS Transformation Language (ATL) [6] [13], have evolved in parallel to this process.

Model weaving [9] [10] is another kind of operation on models. Models are weaved by establishing typed links between their elements. Similarly to transformation programs, which are models, links themselves form a model conforming to a weaving metamodel. Link types are consequently defined in this metamodel. There is one main difference between transformation and weaving. On one hand, model transformation metamodels have fixed semantics that can be implemented in transformation engines. On the other end, weaving metamodels have

[1] Work partially supported by a grant from Microsoft Research, Cambridge, UK, and ModelWare IST European project 511731.

user-defined semantics. ATLAS Model Weaver (AMW) is a tool we developed to experiment with model weaving.

ATL and AMW tools are developed in the context of the ATLAS Model Management Architecture platform (AMMA) [4] [7]. They are both available as open-source Eclipse plugins on GMT [11].

Model transformation can be used to solve some problems by specifying automatic translations from one representation to another. Creating a transformation program is, however, not automatic and has a certain cost. We identified, on an example, three limitations of using only model transformation. They are related to directionality, patterns reuse and changes propagation. This shows that, although model transformation is a key feature of model engineering approaches, it is not enough to address all encountered issues.

In this paper, we propose to use model weaving to improve efficiency in the creation and maintenance of model transformation programs. Weaving links are first established between two metamodels. Transformation programs are then automatically generated to transform models conforming to one of them into models conforming to the other, and vice versa. This generation is itself implemented as model transformations.

This approach overcomes the three limitations previously identified. Additionally, it provides an alternative way of specifying model transformation using model weaving.

This paper is organized as follows. Section 2 presents a model transformation example from relational database tuples into an XML document using ATL. Section 3 highlights some limitations of this first approach. Section 4 describes how model weaving can be applied to our example. Section 5 presents how we can combine model transformation and model weaving in the AMMA platform. Section 6 presents some related works. Section 7 gives conclusions.

2 Transforming Relational Tuples into an XML Document

Let us consider a simple information system for a library. It stores and publishes a list of books with their titles, authors and ISBN (International Standard Book Number). Additionally, a subject is attached to each book. A relational database is used for storage while publication is done in XML. We want to translate the data from the relational database (i.e. a set of tuples) into its equivalent XML representation (i.e. an XML document). We choose to perform this task using model transformation in ATL.

The schemas are presented in Fig. 1: a relational schema *R1* and an XML schema *X1*. Both represent the same information but distinct data structures are used, which is common on data exchange scenarios. For instance, whereas subjects have a *Name* in *R1,* they have a *Descr* (for description) in *X1*. Such dissimilarity may be solved by renaming one of these elements. There is however a more complex structural difference. A relational book refers to a subject using an identifier (i.e. a foreign key), whereas an XML book contains a child element *Subjects*. Although an identifier reference mechanism could be used in *X1 as well*, a nested table is not possible in *R1*.

A transformation between these two representations therefore needs to deal with this issue.

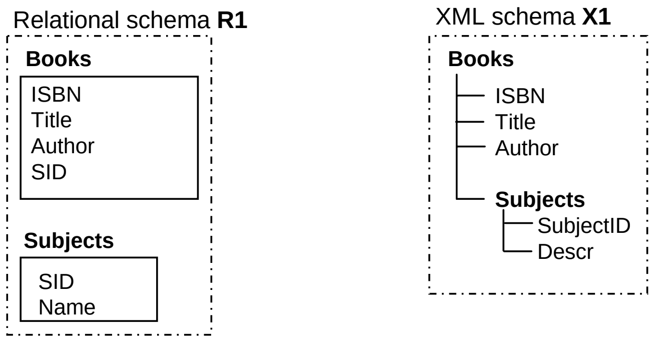

Fig. 1. Relational and XML schemas of a library

We introduce ATL before looking at a concrete solution. We describe only a subset of ATL (i.e. declarative, with single source and target models, etc.), which is necessary to solve our data exchange problem. More complete descriptions can be found in [6] and [13]. An ATL transformation program consists of transformation rules. Each rule contains a source pattern to be matched in the source model and a target pattern to be created in the target model for each match. The source pattern has a type *Ts* coming from the source metamodel and a guard *G*, in the form of an OCL (Object Constraints Language) [22] boolean expression. A match corresponds to a source element, which type is *Ts* and which satisfies *G*. The target pattern has a type *Tt* coming from the target metamodel and contains a set of bindings. Each of them specifies how to initialize one property of the target element from an OCL expression. For each match, one target element of type *Tt* is created in the target model. Each target element is then initialized by applying the corresponding bindings. The abstract syntax of ATL is specified by a metamodel as shown in [6]. Transformation programs are thus models that can be source or target of other transformations.

We now detail the ATL transformation program *RDBMS2XML* implementing a solution to our problem. A listing is given in Fig. 2. The first line contains the program name. The second line declares source and target models. In our case the source is a *RDBMS* (Relational DataBase Management System) model and the target is an *XML* model. Two rules follow. A first one, named *Books*, specifies that an *XML!Book* must be created for each *RDBMS!BookRcd*. Note that each type name is prefixed by its metamodel name to avoid collisions. A second rule, named *Subjects*, specifies that each *RDBMS!SubjectRcd* is transformed into an *XML!Subject* if there exist a book referencing it. Simple target properties, such as *ISBN* and *Descr*, are directly initialized from their corresponding source properties. The *subjects* reference requires a more complex expression: we need to find a *RDBMS!SubjectRcd* having an *SID* equal to the book’s.

```
module RDBMS2XML;
create Target : XML from Source : RDBMS;
rule Books {
    from
        db : RDBMS!BookRcd
    to
        xml : XML!Book (
            Title <- db.Title, Author <- db.Author,
            ISBN <- db.ISBN,
            subjects <- RDBMS!SubjectRcd.
                allInstances()->select (e | e.SID = db.SID)
        )
}
```

```
rule Subjects {
    from
        db : RDBMS!SubjectRcd (RDBMS!BookRCD.
            allInstances()->exists(e | e.SID = db.SID))
    to
        xml : XML!Subject (
            SubjectID <- db.SID,
            Descr <- db.Name
        )
}
```

Fig. 2. RDBMS to XML transformation in ATL

3 Limitations of Direct Transformations

The ATL program presented in Fig. 2 automates the transformation from a relational to an XML representation of a library. This can be executed on any set of tuples conforming to *R1*. There are however limits to this approach. The following paragraphs describe three of them.

Directionality. An ATL transformation can only be executed in the specific direction for which it is written. Source and target models as well as source and target types are clearly identified. This is also visible in bindings. For instance, the relationship between *Subjects.Name* in *R1* and *Subjects.Descr* in *X1* is specified by an assignment of the first to the second in rule *Subjects*. A new program must be developed to implement the inverse transformation. In cases where the mappings between metamodels are more complex, this approach may appear inefficient.

Patterns reuse. The number of different coding patterns used in a given transformation depends on the number of corresponding structures in source and target metamodels. Thus, even if we had more tables and elements in *R1* and *X1*, most of additional rules would only have either simple copy bindings or complex identifier-based to containment-based bindings. New expressions would however have to be written each time. This is a typical tedious and error-prone procedure. Moreover, a different application with a similar architecture (database storage plus XML publication), for instance a phone book or a museum inventory, requires writing a completely new transformation. Since most involved patterns remain the same, this additional work seems inefficient.

Changes propagation. A transformation is tied to a pair of metamodels. Most modifications occurring in any one of them must be reflected into the program. In some cases, multiple corrections must be made. Despite its simplicity, the listing given in Fig. 2 already makes two references to each *RDBMS* type.

Writing a transformation generally solves one problem. But solutions must often be provided to a set of similar problems. In these cases, another approach can be used, as shown in the following sections.

4 Introducing Model Weaving

In order to provide a description of model weaving, we use the example presented in section 2. We need to capture the links between both schemas with all relevant semantic information.

These links are represented in the mapping *R1_X1*, as illustrated in Fig. 3. As already stated, both schemas are similar, thus most elements from *R1* have similar elements in *X1*. The equality between these elements is represented by *Equals* links in the mapping. In addition to this, we must also take into account the structure of both schemas: the foreign key constraints and the nested elements are respectively represented by *FK* and *Nested* links.

Some issues have to be considered concerning the set of links between elements of both schemas:

- The links capture the semantic meaning of relationships between schemas elements and their structure using link types.
- The set of links cannot be automatically generated because it is often based on design decisions or heuristics.
- It should be possible to use this set of links as an input to automatic tools.
- It should be possible to save this set of links as a whole, in order to use it later in various contexts.

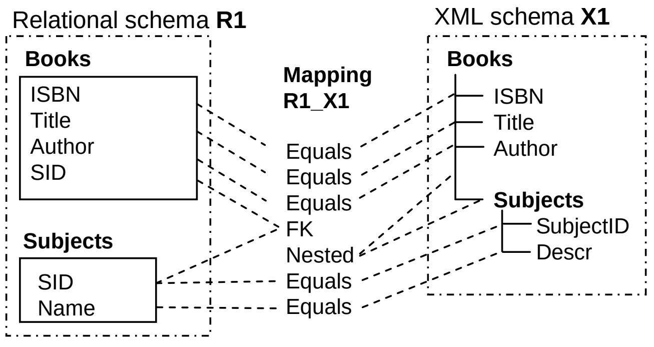

Fig. 3. Links between a relational and an XML schema

We specify a model weaving operation that produces a precise weaving model *WM* representing the mapping between two metamodels. Like all models, *WM* conforms to a weaving metamodel (*WMM*). The produced weaving model describes the relationships between the metamodels *LeftMM* and *RightMM (R1* and *X1* in our example). Fig. 4 illustrates the conformance relations of *LeftMM*, *RightMM* and *WM*.

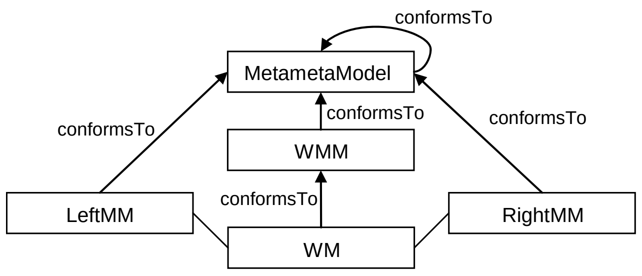

Fig. 4. Weaving conformance relations

In our example, there is weaving model representing relations between a relational and an XML schema. However, metamodels or models with different structures or semantics could be woven. Possible applications are similar to the framework to map between ontologies presented in [16] or business integration presented in [18]. Different semantics could also be captured, such as concatenation [15] or ordering of elements. From that, we conclude that each application domain has different needs that must be considered by the developers. It is not feasible to create a complete weaving metamodel capturing any existing linking semantics: the metamodel designer should be an expert in every linking application.

Thus, to be able to use a weaving metamodel in many different domains it should be designed over an extensible basis. Therefore, we created a small weaving metamodel defining the abstract notions of weaving links between model elements. This core metamodel can then be extended by domain-specific types of links. The design of such a core metamodel is a delicate compromise between expression power and simplicity. Metamodel extension is a complex operation that is out of the scope of this paper. The basic idea, in our case, is that every domain-specific link type extends the core link type.

5 Combining Weaving and Transformation

In our data exchange example the information captured by the weaving model is used as a specification to create transformation programs. We use the Model Weaver tool (AMW) to create a weaving model representing the mappings represented in Fig. 3. The weaving model is then used to generate the transformation given in Fig. 2: from RDBMS to XML. However, the weaving links do not suppose any direction. They can consequently also be used to generate a transformation in the other direction: from XML to RDBMS. We therefore solve the **directionality** problem.

Since the weaving model is used, in this scenario, to automate transformation production, we do not handwrite these two transformations. Each different weaving link is associated with a transformation coding pattern that is automatically applied. For this application, we define three link types in the weaving metamodel: *equality*, *foreign key* and *nested elements*. We create one ATL transformation that implements the corresponding coding patterns. It takes as input the weaving model, the RDBMS and XML schemas and automatically produces a transformation responsible for actually translating the data. This way the patterns are coded once and may be applied to any combination of weaving models and schemas following these patterns. In our example we may generate transformations of any schemas with *equality*, *foreign key* and *nested elements* semantics. **Patterns reuse** can thus be achieved without resorting to manual coding.

In the general case there are many different requirements and a great number of coding patterns. The extensibility of the weaving metamodel enables adding new links types, with new semantics and new corresponding coding patterns as required. For instance, we may need *concatenation* between two elements (e.g. between a street address and a zip code) to obtain a single element. A new weaving link type is created and it can be associated to a string concatenation coding pattern.

As for the **changes propagation** issue, we may observe that weaving links only refer to metamodel types. Provided the reference mechanism supports changes in woven metamodels (e.g. using unique identifiers rather than element names), no changes need to be done to the weaving model. Even if this was not the case, linking semantics is often less verbose than actual code.

6 Related Work

Although ATL only focuses on unidirectional transformations, other approaches directly offer bidirectionality. In the model transformation languages classification presented in [8], [5] is mentioned as such a solution. The approaches from [21], [2] and [3] may also be interpreted as bidirectional, though they require fixing the direction of executable transformations. In current approaches, the semantic of high level specifications is tightly coupled with the transformation engine like in [21]. Model weaving can be used to represent transformation specification as models and is not tightly coupled with any transformation language. This means that AMW and ATL can be used separately or in conjunction with other similar solutions. Changes in the weaving semantics do not require modifications in the core transformation engine. Only the transformations implementing this semantics need to be changed.

In actual transformation languages, reusability mechanisms are based for instance on rule inheritance (e.g. in ATL [13] and in [3]), rule composition (e.g. in [21]), derivation (e.g. in [13]). The extensibility of the weaving metamodel allows providing a vocabulary closer to the application domain. This enables encapsulating sets of model transformation coding patterns and possibly reusing them in different contexts.

7 Conclusion

We have presented a data exchange scenario between relational and XML schemas. By analyzing it, we identified three limitations of model transformation approaches related to: directionality, patterns reuse and changes propagation. Model weaving was then introduced and used on the example to specify a more abstract transformation.

We have shown that, by combining model transformation and model weaving, the identified limitations could be overcome. We do not know of any other solution providing such integration.

Separating specification and actual implementation has several advantages. Thus, a bidirectional mapping can be specified by a single weaving model instead of a pair of transformation programs. Coding patterns implementing semantics of weaving link types can be reused for different left and right metamodels. The use of weaving models coupled with a transformation engine is a promising compromise since it adds variability and automation to the transformation process.

The approach presented in this paper has also been applied on a more complex scenario: bridging the UML profiles approach and the Domain Specific Language (DSL) approach [1].

However this approach may still be improved. For instance, algorithms and heuristics may be developed so that the process of creating a weaving model may become semi-automatic rather than manual.

References

1. Abouzahra, A., Bézivin J., Didonet Del Fabro, M., and Jouault, F. A Practical Approach to Bridging Domain Specific Languages with UML profiles; To appear in: Workshop of Best Practices for Model Driven Software Development, OOPSLA 2005, San Diego, USA
2. Akehurst D. H., Kent S. A Relational Approach to Defining Transformations in a Metamodel. In J.-M. Jézéquel, H. Hussmann, S. Cook (Eds.): UML 2002 - The Unified Modeling Language 5th International Conference, Germany, 30/09 – 10/04, 2002. Proceedings, LNCS 2460, 243-258
3. Alcatel, Softeam, Thales, TNI-Valiosys and Codagen Corporation, Revised Submission for RFP-QVT, OMG Document, August 2003
4. The AMMA Platform. http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/ (2005)
5. Braun P., Marschall F. The Bi-directional Object-Oriented Transformation Language. Technical Report, Technische Universität München, May 2003
6. Bézivin, J., Dupé, G., Jouault, F., Pitette, G., and Rougui, J. E. First experiments with the ATL model transformation language: Transforming XSLT into XQuery, 2nd OOPSLA Workshop on Generative Techniques in the context of MDA, Anaheim, CA, USA (2003)
7. Bézivin, J., Jouault, F., Rosenthal, P. and Valduriez, P., Modeling in the Large and Modeling in the Small, Lecture Notes in Computer Science, Volume 3599, Aug 2005, Pages 33 – 46
8. Czarnecki K., Helsen S. Classification of Model Transformation Approaches. In online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim, October, 2003
9. Didonet Del Fabro M, Bézivin J, Jouault F, Breton E, Gueltas G (2005) AMW: a generic model weaver. In: Proc. of 1ère Journée sur l'Ingénierie Dirigée par les Modèles, Paris, France. pp 105-114
10. Didonet Del Fabro, M., Bézivin, J., Jouault, F., and Valduriez, P., Applying Generic Model Management to Data Mapping, to appear in the proceedings of the Journées Bases de Données Avancées (BDA05), 2005
11. Generative Model Transformer (GMT), 2005 Reference site: http://www.eclipse.org/gmt/
12. GreenField J., Short K. with Cook S., Kent S., (foreword by Crupi J.) – Software Factories, Assembling Applications with Patterns, Models, Frameworks and Tools, Wiley Publishing, 2004

13. Interactive Objects and Project Technology, Revised Submission for RFP-QVT, OMG Document, August 2003
14. Jouault, F., and Kurtev, I., Transforming Models with ATL, to appear in proceedings of Model Transformations in Prac-tice Workshop, October 3rd 2005, part of the MoDELS 2005 Conference
15. Pottinger, Bernstein, P.A. Merging models Based on Given Correspondences, Proc. 29th VLDB Conference, Berlin, Germany, 2003
16. Maedche A., Motik .B, Silva N.,Volz R. MAFRA - A Mapping Framework for Distributed Ontologies; Proc. of the 13th EKAW, Madrid, Spain, 2002
17. Model Driven Architecture, by Richard Soley and the OMG Staff Strategy Group
18. Omelayenko B. RDFT: A Mapping Meta-Ontology for Business Integration, In: Proceedings of the Workshop on Knowledge Transformation for the Semantic Web (KTSW 2002) at the 15-th European Conference on Artificial Intelligence, 23 July, Lyon, France, 2002, p. 76-83
19. OMG, MOF 2.0 Query/Views/Transformations RFP, OMG document ad/2002-04-10 (2002)
20. OMG, Revised Submission for MOF 2.0 Query/View/Transformations RFP (ad/2002-04-10), OMG Document ad/2005-07-01 (2005)
21. QVT-Partners, Revised Submission for MOF 2.0 Query / Views / Transformations RFP, OMG Document ad/2003-08-08, August 2003, http://www.qvtp.org
22. UML OCL 2.0 Specification, ptc/03-10-04, http://www.omg.org/docs/ptc/03-10-14.pdf