Applying Generic Model Management to Data Mapping[1]

Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, Patrick Valduriez

ATLAS Group, INRIA and LINA University of Nantes - 2, rue de la Houssinière, BP 92208 - 44322 - Nantes cedex 3, France

{marcos.didonet-del-fabro, jean.bezivin, frederic.jouault}@univ-nantes.fr, patrick.valduriez@inria.fr

Abstract. *Mapping between heterogeneous data is a central problem in many data-intensive applications. In particular, using one mapping language causes serious limitations and makes mapping management difficult. In this paper, we propose a solution that can better control the trade-off between genericity, expressiveness and efficiency of mappings. Our solution considers mappings as models and exploits specific mapping engines. We define model weaving as a generic way to establish element correspondences. Weaving models may then be used by a model transformation language to translate source model(s) into target model(s). To validate our solution, we implemented a mapping prototype, called AMW, and used it for experimenting with significant application scenarios.*

Key words: model weaving, metamodels, data mapping, metamodel composition.

1. Introduction

Mapping heterogeneous data from one representation to another is a central problem in many data-intensive applications. Examples can be found in different contexts such as schema integration in distributed databases [Özsu 1999], data transformation for data warehousing [Cui 2003], data integration in mediator systems [Lenzerini 2002], data migration from legacy systems [Bisbal 1999], ontology merging [Ehrig 2004], etc.

A typical data mapping specifies how data from one source representation (e.g. a relational schema) can be translated to a target representation (e.g. an XML schema). Although data mappings have been studied independently in different contexts, there are two main issues involved. The first one is to discover the correspondences between data elements that are semantically related in the source and target representations. This is called schema matching in schema integration [Batini 1986]; many techniques have been proposed to (partially) automate this task, e.g. using neural networks [Li 2000]. After the correspondences have been established, the second issue is to produce operational mappings that can be executed to perform the translation. Operational mappings are typically declarative, e.g. view definitions or SQL-like queries. Creating and managing data mappings can be very complex and time-consuming if done manually. Recent work in schema integration has concentrated on the efficient management of data mappings. For instance, Clio [Miller 2001] provides techniques for the automatic generation of operational mappings from value correspondences obtained from the user or a machine learning technique. ToMAS [Velegrakis 2003] also provides techniques for consistency management while schemas evolve.

However, there is a trade-off between genericity and efficiency of mapping management. By supporting one representation language, e.g. relational or XML, mapping management can deal efficiently with correspondences using a fixed mapping language, e.g. SQL or XQuery. This approach can be extended to deal with multiple representation languages using

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

wrappers or extractors [Garcia-Molina 1995]. The development of such wrappers or extractors that also involve data mappings can be difficult and time-consuming.

To support arbitrary mappings in different languages with different semantics, generic model management has recently gained much interest [Bernstein 2003]. A model is a formal description of a design artifact such as a relational schema, an XML schema, a UML model or an ontology. By considering mappings between models also as models, much expressiveness, flexibility and genericity can be gained. Rondo [Melnik 2003] is the first complete prototype of a generic model management tool. It uses a high-level algebraic language to manipulate models and mappings between models. As a result, Rondo can define data mappings between heterogeneous representations, and use the operators for complex model manipulation, including merging of models [Melnik 2003]. It uses its own mapping language called morphisms. However, there are many cases where a specific mapping language and associated engine has better expressiveness and can be more efficient, e.g. XSLT for mapping XML schemas.

In this paper, we propose a solution that can better control the trade-off between genericity, expressiveness and efficiency of mapping representation. In other words, our objective is to support generic data mapping (as in Rondo but with a different approach) while exploiting specific mapping languages and engines, such as Clio, ToMAS or an XSLT engine. We thus consider data mapping languages as extensible Domain Specific Languages (DSL) [GreenField 2004].

To be able to achieve this trade-off, we use model weaving, which consists of establishing correspondences with semantic meaning between model elements. A weaving model is a special kind of model used to save these correspondences. Since a weaving is considered to be a model, it conforms to a definition language that specifies the possible formal structures. This language is expressed in terms of models as well. The created weaving model may be later used by a model transformation language to translate source model(s) into target model(s).

Capitalizing on previous work on schema integration and model management, the main requirements for generic data mapping management can be summarized as follows:

1. be able to perform mappings between complex models, which implies to reason about correspondences between these complex models;
2. be able to produce new mappings from existing ones, e.g. to adapt mappings after models change or to allow incremental specification of mappings;
3. be able to generate operational mappings in different languages with their own mapping engine.

Thus, we can state the problem as follows: given two models, produce a weaving model that represents all relevant mapping correspondences. New weaving elements may be modified, added or removed. The weaving model is used to translate source model(s) into target model(s) or to generate a different mapping representation. The specification of weaving models must be extensible to be used in different mapping scenarios.

In developing our solution, we define a weaving model specification representing the basic concepts in a mapping system, e.g., the links between elements and a way to represent these links from different models. Considering weaving specifications also as models enables to add extensions capable of expressing complex semantics, such as foreign keys constraints or ontology mappings. We define a generic operation to create weaving models. A weaving model can also be modified to follow evolution of woven models. It can be transformed in two different ways: first, using a declarative model transformation language such as ATL [ATL 2005] to transform source models into target models and second, by translating into a different mapping representation (such as morphims or XSLT) to be executed in the corresponding transformation engine. Thus, we have one weaving model, and several weaving executions.

To summarize, the paper has several contributions. First, we define a generic model management operation, called model weaving, to represent mappings between complex models (requirement 1). Second, we propose the supporting technology that enables the reuse and evolution of mapping definitions (requirement 2) and the generation of operational mappings in different languages (requirement 3). Third, we validate our approach using the ATLAS Model Weaver (AMW) prototype on application scenarios.

All the results presented in this paper are supported by open source prototypes currently running under the Eclipse Modeling Framework (EMF) [DelFabro 2005].

This paper is organized as follows. Section 2 presents an example that motivates the need for an adaptive mapping platform. Section 3 describes formally model weaving and related concepts. Section 4 shows how data mapping is represented as a weaving. In Section 5 we present a validation example. Section 6 discusses related work. Section 7 concludes.

2. Motivating Examples

We illustrate the general data mapping problem we address with two related data exchange scenarios. We also discuss the difficulty to create at the same time a generic solution capable to handle dedicated mappings and to integrate existing specialized solutions.

Scenario 1: Libraries usually exchange data to have a standard catalogue format, both for standardization and interoperability purposes. Let us consider two data sources as show in Figure 1. One library has its own relational schema as defined by *Relational schema R1*. But it also agrees to use an XML format as defined by *XML schema X1*. Schema *R1* has two tables: *Books (ISBN [International Standard Book Number], Title, Author, SID)* and *Subjects (SID, Description)*, with the foreign key *SID* on *Books* referencing the subjects of a book. Schema *X1* has the same basic structure except for the foreign key in books since this correspondence is represented by the nested structure between *Books* and *Subjects*.

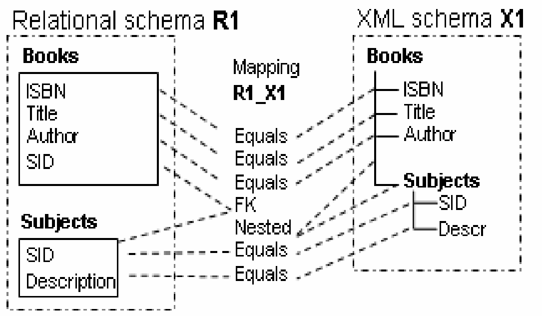

Figure 1. Relational to XML mapping

The translation from *R1* into *X1* is represented by the mapping *R1_X1*. It has three mapping structures: *Equals* that is an inter-schema correspondence that indicates equalities such *as R1.Books.ISBN = X1.Books.ISBN, R1.Books.Title = X1.Books.Title*, and so on; *FK* is an intra-schema correspondence indicating the foreign key constraint between *R1.Books.SID* and *R1.Subjects.SID*; *Nested* is another intra-schema correspondence representing the nesting relationship between *X1.Books* and *X1.Books.Subjects*. These intra-schema correspondences guarantee the generation of a valid output model.

Scenario 2: Continuing the library exchange problem, let us consider a second library that will use the *XML schema X1* to integrate with its own data as indicated in Figure 2 by the *Ontology O1*. The ontology represents periodic data, e.g., newspapers and magazines. It has the attributes *ISSN [International Standard Serial Number], Title (Subtitle), Publisher, Subjects (ID, Description)* and *Author*.

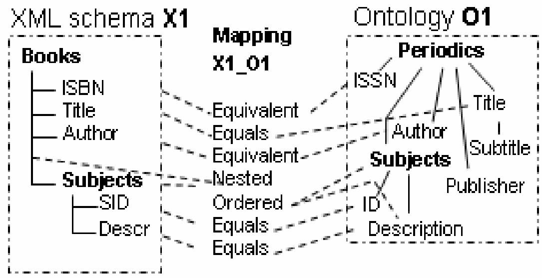

Figure 2. XML to ontology mapping

The mapping structures are represented by *Mapping X1_O1*. It has *Equals* and *Nested* with the same semantics as in Scenario 1, and adds two new mappings: *Equivalent* and *Ordered*. Equivalent shows that *ISBN* is equivalent but not equal to *ISSN*, because they both indicate the object identification, though with different meaning. The same is valid for *Author* because the book author represents a person while a magazine's author is an entity. *Ordered* indicates that the periodics' subjects must be ordered by *Description*.

Analyzing these two scenarios, we observe that all inter- and intra- model relationships need a structure to represent correspondences between elements, independently of mapping semantics. This motivates the creation of a common mapping core. In both scenarios, new additional structures are specified following different requirements, such as *FK* and *Nested* in Scenario 1, *Equivalent*, *Nested* and *Ordered* on Scenario 2. This shows the necessity to allow incremental specification of mappings, having an extensible core with dedicated subsets. This also shows the importance to have an expressive representation allowing to reason about correspondences between complex models like the *Ordered* relationship in Scenario 2. The mappings are represented in the same language. However, in both cases one may generate operational mappings in a different language with a more dedicated transformation engine (relational to XML and XML to ontology).

3. Model Weaving

In this section, we define model weaving, which is a generic operation that establishes correspondences with semantic meaning between complex model elements.

There is no accepted formal definition of the main elements behind model weaving, such as model, metamodel, model transformation, etc. The MOF specification [OMG 2002] describes a model-based four-layer architecture, but with no formal definitions. In [Melnik 2004], there is a formal definition of models but not adapted to model weaving. In [Popa 2002] and in many other model platforms over relational databases, terms such as (relational) data model, relational schemas, nested schemas, database states, model instances are defined, but there is no standard taxonomy. In [Bernstein 2003], there is an informal starting point. Thus, we develop our own definitions.

3.1. Models

A system may be computationally represented by a model. A system is a group of interacting, interrelated, or interdependent elements that form a complex whole, for example a library system, an internet bid system, a car rental system, or a university application system.

Definition 3.1 (Model). A model is a directed graph *G = (V, A)*. The set of vertices *V* denotes model elements. A model element from *V* has an identifier and a value. The identifiers may be implemented as URIs and the element value may be of any data type, such as integer, strings, classes. The set of labeled edges *A* denotes associations between model elements.

Let us illustrate Definition 3.1 with a model representing a book, with a single property title. We have a model *M1 = (V, A); V = {r1 (""), h1 ("has"), t1 ("Data mapping")}; A = {(r1, h1), (h1, t1)}*. The element identifiers are illustrated by *r1*, *h1, t1*. The element values are inside parenthesis, which in this case are character literals. The model element *r1* represents the book record, *t1* the title and *h1* a containment. We have two associations between them. However the possible structures of the model are not explicitly defined. They are defined in a *metamodel.*

***Definition 3.2 (Metamodel)*.** A metamodel is a special kind of model that specifies the structure of a model. A model conforms to a metamodel. Given a model *M = (V, A)* and a metamodel *MM = (V', A'),* for every model element *e* ∈ *V*, there is an outgoing edge to an element *me* ∈ *V',* labeled as a *Meta* edge. We denote it by *Meta (e, me).*

Consider the model *M1* presented above. It conforms to metamodel *MM1 = (V', A')*; *V' = {record ("*record"), *title ("title"), hasA ("hasA")}; A'= {(record, hasA), (hasA, title)}. MM1* defines the concepts of record, title and containment. The associations indicate source and target elements. We have the following *Meta* edges from *M1* to *MM1*: *Meta (r1, record)*, *Meta (t1, title)*, *Meta (h1, hasA)*. *MM1* acts like a typing system. A metamodel conforms to a *metametamodel*.

Definition 3.3 (Metametamodel). A metametamodel is a metamodel defining the base structure for all metamodels and models within a specific context. A metametamodel conforms to itself.

Consider the metamodel *MMM1 = (V'', A'')*. *V''= {entity ("entity"), link ("link")}*. *A''= {(entity, link), (link, entity)}*. *V''* has two elements, indicating an entity and a link. It has two associations, one going from an element to a link and another from a link to an element. *MM1* conforms to *MMM1*, which means we have the meta edges: *Meta (title, entity), Meta (record, entity)* and *Meta (hasA, link)*. All the other metamodels and models from this context are constructed in terms of *links* and *entities*.

We may have the same system, for instance a library system, represented by models in different implementation contexts, such as XML documents (XML trees); relational databases (relational model); or MDA™ (a special kind of graph). Thus each context has a unique metametamodel.

3.2. Model Transformations

Model transformation is an operation that takes as input a set of models and produces another set of models as output.

Definition 3.4 (Model transformation). A model transformation *T* is an operation that given a set of input models *(M1,…, Mn)*, evaluates them and returns a set of output models *(OM1,…, OMn)*. A transformation may be denoted by a model *Mt*, called a transformation model. A transformation model has the following properties:

1. it conforms to a transformation metamodel;
2. the transformation body is created taking as values the input or output metamodels
3. source and target models are distinct;
4. in the transformation execution, the input elements are matched based on the input metamodels;
5. the output elements are created from the evaluation of the matched elements.

Let us illustrate model transformation on Scenario 1 in Section 2 where we need to translate relational database records into XML documents. The transformation takes as parameters the relational records, conforming to the library relational schema. In its body it is specified how the table *Subject* and its columns are translated into the corresponding nested node and its attributes. It produces as output an XML document. In the second scenario we have a transformation that specifies how XML nodes and attributes are translated into the ontologies and its attributes. The translation between these data sources is specified in a transformation language, such as an ATL or XSLT.

3.3. Model Weaving

Model weaving is a generic operation that establishes fine-grained *correspondences* between model elements. It receives as parameter a set of models and produces a *weaving model.*

A *correspondence* defines associations between elements from different models. Given two models *M1 = (V, A)* and *M2= (V', A')* and model elements *e1* ∈ *V'* and *e2* ∈ *V''*; the edge *(e1, e2)* (which is not a *Meta* edge) is said to be a correspondence.

However its syntactic nature does not allow defining complex structures to relate two or more models. We use a weaving model to capture more complex models relationships.

Definition 3.5 (Weaving model). A *weaving model* represents correspondences in terms of its model elements. Let *M1 = (V, A)* and *M2 = (V', A')* be distinct models. Given elements *e1* ∈ *V* and *e2* ∈ *V',* the correspondence *(e1, e2)* is denoted by the triple *(e1, Mw, e2)*, where *Mw = (Vw, Aw)* is a weaving model. The structure of a weaving model is defined in a weaving metamodel.

Definition 3.6 (Model weaving). *Model weaving* is a generic operation that takes as input a set of models *(M1, ..., Mn)*, a weaving metamodel *MMw* and returns a weaving model *Mw.*

A weaving operation has the following properties:

1. it may define hooks to enable gradual refinement of a weaving model;
2. it may be defined in terms of model transformations.

After the operation execution, the models *(M1, ..., Mn)* are woven models. Note that metamodels and metametamodels may also be woven.

The model weaving operation is implemented by the operator *Weave* (see Figure 3). We describe it for the case of weaving two complex models, *M1* and *M2*, but it can be extended to weave several models. The *Weave* algorithm first creates a weaving model conforming to *MMw.* Then for every element e_i from *M1,* and every me_j from *MMw,* it searches for matching elements in *M1* or *M2*. It returns the correspondences found. The search is executed in the *SearchCorresp* function. This function is not generic, it must be modified to handle any different structure defined by each me_j. The returned correspondences are used to create the element *mnew*, which associates them according to the structure of me_j. We call it a *weaving link* element.

A variant of the algorithm has a weaving model *Mw* as an extra parameter. The signature is modified to *Weave* (*M1*, *M2*, *MMw, Mw*). This way, weaving elements may be incrementally added into an existing weaving model. In Figure 3, the code lines to be added to the basic algorithm are in bold font.

In Scenario 1, the mapping *R1_X1* has two sets of correspondences, one set with the relational schema elements from *R1* and another with the XML schema elements from *X1*, illustrated by the dashed lines. To be able to create links between these two models, we must create a weaving link element. This is illustrated by *Equals*, *FK* and *Nested*. The same is valid for the second scenario, where mapping *X1_O1* defines correspondences with an XML schema and with an ontology. It has equality, equivalence, nested and ordered semantics. The mapping is represented by a weaving model created by a weaving operation.

```
Weave (M1, M2, MMw, Mw)
   If Mw is null
      Mw = createWeavingModel(MMw);
 for all ei in M1 do
 begin
   for all mej in MMw do
   begin
     corresp = SearchCorresp (mej, ei, M1, M2);
     for all ek in Mw
       if not exists ek with corresp
       begin
         mnew = Create(mej, corresp);
         Mw = add ( mnew );
       end
   end
 end
return Mw;
```

Figure 3. Algorithm to weave 2 models

The weaving metamodel is not a fixed metamodel. It might be extended to form dedicated weaving metamodels. This is done using the composition operation.

Definition 3.7 (Composition operation). The composition operation takes as input a weaving metamodel *MMw*, a metamodel *MMe* and a weaving model *Mwc*. It returns a new weaving metamodel *MMwn*, which is the composition of *MMw* and *MMe*.
The operation is defined as *MMwn = ComposeMM (MMw, MMe, Mwc)*. The composition semantics between *MMw* and *MMe* are specified in the weaving model *Mwc*.

```
abstract class WElement{
   attribute name : String;
   attribute description : String;
   reference model : WModel; }
abstract class WModel extends WElement{
   reference ownedEl[*] container: WElement;
   reference wModel[1-*] container: WModelRef; }
abstract class WRef extends WElement{
   attribute ref : String; }
abstract class WModelRef extends WRef{
   reference ownedElementRef[0-*]
            container : WElementRef;}
abstract class WElementRef extends WRef{
   reference modelRef : WModelRef; }
abstract class WLink extends WElement{
   reference end[1-*] container : WLinkEnd; }
abstract class WLinkEnd extends WElement{
   reference link : WLink oppositeOf end;
   reference element : WElementRef; }
```

Figure 4. Abstract weaving metamodel

The operation reads *Mwc* and executes the specified semantics. It has as principal requirement the creation of at least one new element *new_e* in the resulting weaving metamodel. This element put into relation one element *mme* $\in$ *MMe* and one element *mmw* $\in$ *MMw*, for instance by the means of references, containments or inheritance. It prevents from creating a mal-formed weaving metamodel with two sets of elements without any association between them. It is a metametamodel-specific operation.

4. Data Mapping

We propose to use model weaving as the base for a solution to various data mapping problems. The first step to achieve this is to define weavings capable of reasoning about complex mappings. Then the weavings metamodels should be adapted as application requirements evolve. We call this *correspondence discovery.* Weaving models are further used as a guidance to generate operational mappings in different transformation languages. This is called *operational mapping production*.

4.1. Correspondence Discovery

We specify a minimal weaving metamodel used as a basis for a mapping platform. It may be further composed with another metamodels to create dedicated weaving metamodels. The metamodel represents the concepts of weaving links. We use as metametamodel Ecore [EMF 2005]. The weaving metamodel is thus specified in a textual language to represent metamodels in Ecore called KM3 [Bézivin 2004]. We provide an excerpt of our weaving metamodel in Figure 4.

WElement is the base element from which all other elements inherit. *WModel* represents the root element that contains every model element. We have the notion of link extremity (*WLinkEnd*). It makes reference to a *WElementRef*. This element captures the necessary information to make reference to the elements of the woven model, providing a flexible identification mechanism. The element *WLink* references multiple extremities, representing a weaving link. *WModel*'s contains also *WModelRef's*, which is equivalent with the reference of *WLinkEnd* and *WElementRef*, but for models as a whole.

The weaving metamodels must adapt to follow evolution in the woven models and in the mapping requirements. New data mapping specifications, e.g. weaving metamodels, are incrementally composed with existing ones, being able to express other complex relationships. Each extension may be separately saved and further reused (by composition) with other weaving metamodels, according to different mapping requirements.

Consider we have a weaving metamodel *MMw*. It contains the elements representing the abstract metamodel described in Figure 4. We have another metamodel *MMdb* with elements representing foreign keys (fk_*e*) and generation of automatic values (*av_e*). We compose *MMw* with *MMdb,* adding an inheritance association between *fk_e* and *WLinkEnd,* and between *av_e* and *WLinkEnd*. The elements *fk_e* and *av*_e become capable of representing element correspondences, and add semantic meaning to them. The new weaving metamodel may be composed in turn with a new extension that contains one element defining ordering of elements.

However the existing metamodel must not change in a way it interferes with existing weaving models. For example we may have an element *e1* that is woven with an element *e2* by the means of an equality element *e* (its structured is defined in a *mequal* element in the metamodel). The *mequal* element should not be excluded or modified from the metamodel; otherwise the current weaving model becomes invalid. In this case it is necessary to recreate the model.

4.2. Operational Mapping Production

A weaving model is not an executable entity: the translation between data sources are executed by model transformations that use the weaving as specification. However weaving models should not be dedicated to one transformation engine. There are many performing engines and languages that could be used in specific cases. A weaving model may also be translated into another mapping language that will be used in its own mapping platform.

It is not desirable to directly create a transformation from source model(s) into target model(s); otherwise one should write by hand a new transformation for every weaving model. We define algorithms based on the weaving metamodel and model elements. They automatically produce different transformation models, which body takes as values the woven metamodel elements. We may produce transformations in different transformation languages or mappings, such as ATL, SQL queries, XSLT or morphisms. They are further serialized into the appropriated representation. The serialized form takes as input the woven models, to actually perform the data translation between the data sources in the dedicated transformation engine.

Thus, we may obtain different transformations as output based on the same weaving. This is possible because despite having different syntax, expressive power and capacity of calculation, the structure of existing transformation languages follows similar standards. This enables the creation of weavings targeted for transformations in general. We describe such standards below:

- *input* and *output models* and their *metamodels*: are the source and target models, e.g. an XML document, an ontology, a relational table. The metamodels may be explicitly specified or implicitly implemented in one ad-hoc engine;

- *rules*: are self contained commands containing all the necessary constructs to translate source elements into target elements , e.g. an SQL view, an XSLT stylesheet or an ATL rule;

- *input elements*: define which elements from the input model are transformed. Input patterns usually relate elements formed by sub-elements or attributes, e.g. ATL input patterns, XSLT matched templates or SQL select from clauses;

- *output elements*: define the target elements, strictly related with the input elements, e.g. ATL out patterns, XSLT elements or SQL create view clauses;

- *selection expressions*: define filters in the input patterns to recuperate only a set of elements, e.g. ATL filters, XPath expressions or SQL where clause;

- *equivalence expressions*: define the correspondences between the attributes of a given input element and the attributes of the output elements, e.g. ATL bindings, XSLT value-of or SQL relation from the select to the view clause. The weaving elements indicating correspondences and their semantics should be translated as equivalence expressions;

- *calculation expressions*: return a new value after executing calculations over input element to be used in an equivalence expression, e.g. OCL expressions [UML 2004], XPath or SQL functions.

5. Validation

In this section, we present a validation based on our ATLAS Model Weaver (AMW) prototype which we use to experiment with the scenarios defined in Section 2. The prototype is available in the Eclipse GMT project [GMT 2005].

5.1 Model Weaver Prototype

AMW is a component-based platform with separated components to handle each weaving requirement. The platform is based on the Eclipse [Eclipse 2005] contribution mechanism: components are defined in separated plugins. The plugins are further interconnected to create the model weaver workbench. Components for user interface, matching algorithms and serialization of models may be plugged as necessary. We extend an existing architecture for model manipulation (Eclipse EMF [EMF 2005]). This extended component coordinates the weaving actions. We use the EMF dynamic API to obtain a standard weaving editor. The editor adapts its interface according to the weaving metamodel. Metamodel extensions are plugged as KM3 files. Each KM3 file may have an associated user interface to help in the matching task. As representation metametamodel we use *Ecore*, which is the Eclipse EMF metametamodel similar to the OMG Meta Object Facility [OMG 2002]. The ATL transformation engine is plugged as the standard transformation platform.

5.2 Experiments

To demonstrate support for data mapping requirements, we start from the minimal weaving metamodel as a basis. We incrementally refine it with extensions adapted for the application scenarios in Section 2. The created weaving should be used as a specification to automatically generate transformations for different engines.

We first defined a concrete version of the abstract weaving metamodel, and created a weaving model to represent mappings *R1_X1* and *X1_O1*, first without specific semantics. We were able to define similar structures as morphisms and value correspondences. They could be used in their respective mapping environments, thus showing the feasibility of integrating different mapping solutions in a common core.

We incrementally adapted the existing weaving metamodel (represented by *MMw*), e.g., mapping specification, composing it with the new extensions until having a weaving metamodel with all necessary semantics. We have thus dedicated mapping specifications with variable expressive power: we represented from simple element links such as *Equals;* then *Nested* and *FK* constraints; *Equivalent*; until complex ones as *Ordered*, as shown in Figure 5. This brings an advantage over all purpose and complex mapping languages because they are usually designed focusing a specific environment and do not adapt well.

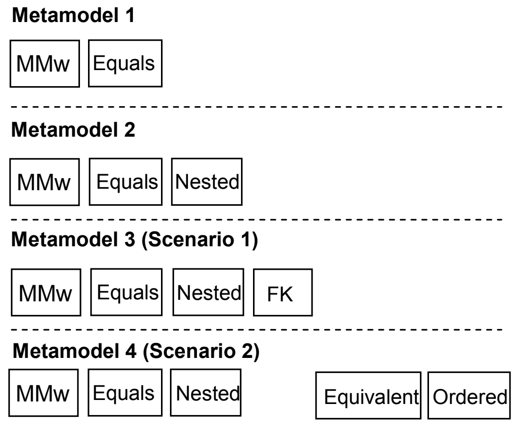

Figure 5. Composed weaving metamodels

The weaving model was created and modified in parallel with each new metamodel composition, e.g., as soon as we created a new metamodel, we changed the associated model, which did not invalidate the existing elements.

To be able to weave models created in different contexts, the relational schema was imported into our tool. The same applies to the XML schema and to the ontologies. We used simplified versions of the models and metamodels, capable of representing only the desired structures.

The weaving model is used as specification for producing operational mappings in two different languages, ATL and XSLT. We also generated morphisms, obtaining a different mapping representation.

We choose ATL because it enabled us to apply sound model management concepts; XSLT with XPath because it is the standard transformation language for XML documents, with several engines available; and morphisms to obtain a different mapping representation, even if it is not capable of expressing all the desired semantics.

We produced an ATL model, a XSLT model and a model representation of morphisms. They were serialized in their text format. The generated ATL and XSLT were actually used to transform the source models into the target models. We show in Figure 6 an excerpt of the generated operational mappings from *Scenario 1* with the rules to handle nested and foreign key semantics.

```
XSLT rule
<xsl:template match="bookRcds">
   <xsl:element name="books">
      <xsl:attribute name="ISBN">
         <xsl:value-of select="@ISBN"/>
      </xsl:attribute>
      <xsl:variable name="sid" select="@SID"/>
      <xsl:apply-templates select="/descendant-or-
         self::subjectRcd[@SID=$sid]">
      </xsl:apply-templates>
   </xsl:element>

</xsl:template>
<xsl:template match="subjectRcd">
   <xsl:element name="subjects">
      <xsl:attribute name="SubjectID">
         <xsl:value-of select="@SID"/>
      </xsl:attribute>
   </xsl:element>
</xsl:template>
ATL rule
rule Books {
   from
      db : RDBMS!BookRcd
   to
      xml : XML!Book (
         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,
      )
}
```

Figure 6. Generated XSLT and ATL

In Figure 7 we see the AMW user interface for *Scenario 1*. In the left we have the source relational database schema, in the right the target XML schema, and in the middle the weaving model created conforming to the *Metamodel 3*.

6. Related Work

Data mapping has been extensively studied in the literature. There are several solutions focusing on specific application domains, or on specific mapping problems. Clio [Miller 2001 and Popa 2002] concentrates on mapping schema-based structures such as XML and relational databases, generating SQL queries or XSLT transformations based on value correspondences. Our model representation of mappings enables mapping models with different kinds of structure, and we may generate transformations for a variety of execution engines.

In [Omelayenko 2002] a rich mapping meta-ontology is defined to map between XML DTDs and RDF schemas concentrating on business integration. We have rich mapping representations as well, however our extensible weaving metamodel may be applied to a wider family of problems. MAFRA [Maedche 2002] is a framework for aligning ontologies. It introduces the notion of semantic bridges for mapping between ontologies and it creates one "semantic bridge ontology" with these mapping constructs. It has a similar approach as the mapping ontology for business integration, focusing on ontologies in general, not fitting for specific mappings requirements for other contexts, such databases and XML documents.

Rondo [Melnik 2003] is the most general solution. It implements generic model management operators such as *Match*, *Merge*, *Extract*, as well as the necessary semantics to generate well-formed models. It solves many mappings problems; however the syntactic representation is not capable of expressing complex model constructs. These operators produce mappings based on fixed semantics. In our solution we have variation on mapping structure, which allows obtaining domain specific mapping languages. This variable specification makes difficult the implementation of the proposed operators in a generic way, since the semantic is not known in advance. Model management operations over reified mappings are proposed in [Bernstein 2003]. Our approach is similar in the utilization of mappings however we propose the execution of mappings in terms of model transformations, and we provide extensible mapping definitions.

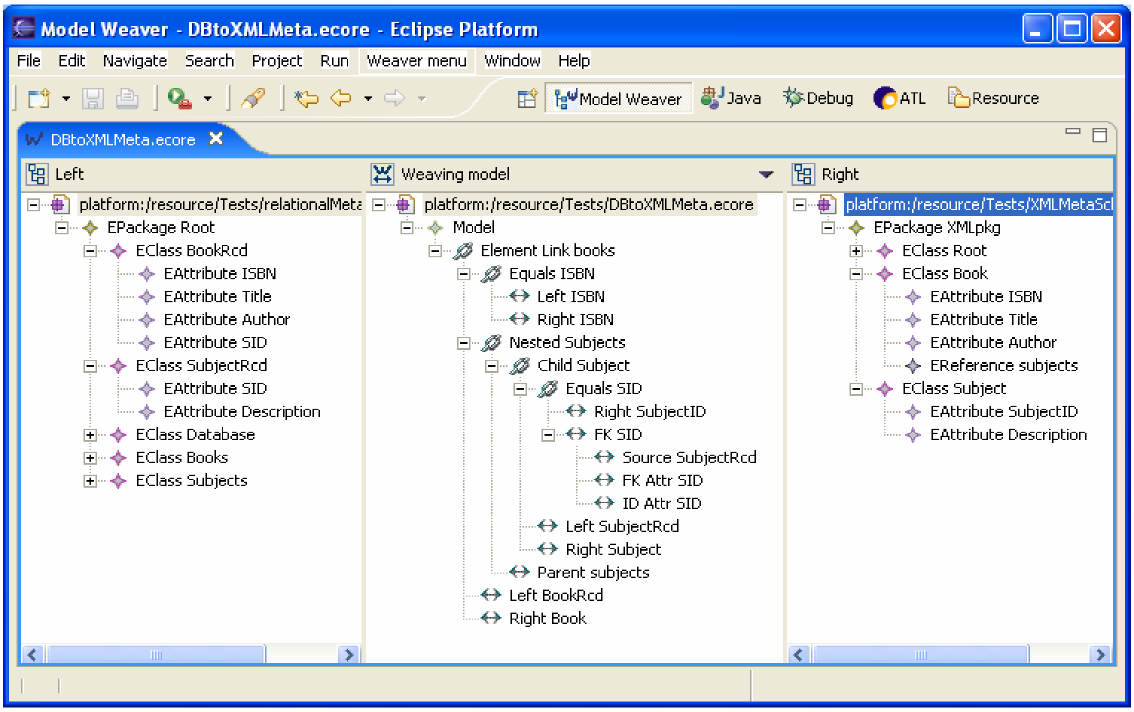

Figure 7. Weaving in the AMW prototype

7. Conclusion

In this paper, we proposed a solution that applies generic model management to data mapping in order to better control the trade-off between genericity, expressiveness and efficiency of mappings. Our solution is based on model weaving, a new way of establishing fine-grained correspondences between model elements. Since a weaving is considered to be a model, it conforms to a metamodel that specifies the possible formal structures. The created weaving model may be later used by a model transformation language to translate source model(s) into target model(s).

The main contributions of this paper may be summarized as follows. First, we defined model weaving, a generic model management operation to create mappings between complex models. The model weaving operator weaves several correspondences into a weaving element, which may represent complex semantics. We defined a minimal weaving metamodel to obtain a generic representation.

Second, we proposed the supporting technology that enables the evolution and reuse of mapping definitions and the generation of operational mappings in different languages. The composition of weaving metamodels enables to add constructs with greater expressiveness, having dedicated mapping languages. This enables to handle fine-grained problems that are not addressed by overall mapping architectures. The compositions may be done incrementally to follow the evolution of mapping requirements. We may reuse each metamodel extension composing it with different weaving metamodels. We separated mapping specification and definition from operational mapping production. We summarized the common features of existing transformation languages to be used as a guidance to define weaving metamodels. It allowed us to generate different mappings representations.

Third, we validated our approach using the ATLAS Model Weaver (AMW) prototype on application scenarios. The weaving metamodel was incrementally composed with other metamodels. We reused two metamodels extensions, having as result metamodels with specialized semantics for each scenario. We produced operational mappings in ATL and XSLT, and morphisms. Experimentation with this model weaving in the AMMA platform has shown that many different proposals may be unified by our model-based approach. Coupling a weaving facility (like AMW) with a transformation facility (such as ATL) gave us good efficiency and flexibility.

As future work we plan to use model weaving in application scenarios not yet explored, such as merging of models. We also plan to study semi-automatic matching of weavings to be used inside the model weaving operation. For the time being we envision using standard Eclipse plug-ins to solve this problem, by plugging different matching algorithms to help in the weaving model creation.

8. Acknowledgments

The prototype was built with the help of Erwan Breton and Guillaume Gueltas as part of a collaboration with the Sodifrance Company.

References

ATL 2005, ATLAS Transformation Language. Reference site: www.sciences.univ-nantes.fr/lina/atl/ (February 2005) or http://www.eclipse.org/gmt

Batini, C., Lenzerini, M., and Navathe, S. B. A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys 18, 4, 323–364, 1986.

Bernstein, P.A. Applying Model Management to Classical Meta Data Problems, Proc. CIDR 2003, pp. 209-220

Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P. The AMMA platform support for modeling in the large and modeling in the small. LINA research report (04.09), November 2004

Bisbal J., Lawless D., Wu B., Grimson, J. Legacy Information Systems: Issues and Directions. IEEE Software, September/October 1999, pp. 103-111, Vol. 16, Issue 5. 1999.

Cui, Y., and Widom, J. Lineage Tracing for General Data Warehouse Transformations. VLDB Journal, 12(1):41-58, May 2003.

Didonet Del Fabro M., Bézivin J., Jouault F., Breton E., Gueltas G. AMW: a generic model weaver In: Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles, Jun/Jul 2005

Eclipse 2005, Eclipse Project. Reference site: http://www.eclipse.org

Ehrig M. York Sure: Ontology Mapping - An Integrated Approach. ESWS 2004: 76-91

EMF 2005 Eclipse Modeling Framework (EMF), 2005 Reference site: http://www.eclipse.org/emf

Garcia-Molina H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In Proc. of the AAAI Symposium on Information Gathering, Stanford, California, March 1995.

GMT 2005 Generative Model Transformer (GMT), 2005 Reference site: http://www.eclipse.org/gmt

GreenField J., Short K., Cook S., Kent S. (foreword by Crupio J.) - Software Factories, Assembling Applications with Pattenrs, Models, Framewords and Tools, Wiley Publishing, 2004

Lenzerini M. Data integration: a theoretical perspective, Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, June 03-05, 2002, Madison, Wisconsin

Li W, Clifton C. SemInt: a tool for identifying attributes correspondences in heterogeneous databases using neural networks. Data & Knowledge Engineering 33(1):49–84, 2000

Maedche A., Motik .B, Silva N.,Volz R. MAFRA - A Mapping Framework for Distributed Ontologies; Proc. of the 13th EKAW, Madrid, Spain, 2002.

Melnik, S., E. Rahm, Bernstein P. A. Rondo: A Programming Platform for Generic Model Management, Proc. SIGMOD 2003, pp. 193-204

Melnik, S Generic Model Management: Concepts and Algorithms, Ph.D. Dissertation, University of Leipzig, Springer LNCS 2967, 2004

Miller, R. J., Hernandez, M. A., Haas, L. M., Yan, L.-L., Ho, C. T. H., Fagin, R., and Popa, L. The Clio Project: Managing Heterogeneity. SIGMOD Record 30, 1, 78–83, March 2001.

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

OMG 2002 MOF: Meta Object Facility (MOF) Specification. OMG Document AD/02-04-03, April 2002.

Özsu T., Valduriez P. Principles of Distributed Database Systems. 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 666 pages, 1999.

Popa L. ,Velegrakis Y., Hernandez M., Miller R. J., Fagin R. Translating Web Data, in 28th International Conference for Very Large Databases (VLDB 2002), August 2002.

UML 2004 UML OCL 2.0 Specification, ptc/03-10-04, http://www.omg.org/docs/ptc/03-10-14.pdf

Velegrakis Y., Miller R. J., Popa L. Adapting Mappings in Frequently Changing Environments, Int. Conf of Very Large Databases (VLDB), Sep 2003.