A Model-driven Approach to Interoperability in B2B Data Exchange

Dumitru Roman, Brice Morin, Sixuan Wang, Arne J. Berre

SINTEF, Norway

Forskningsveien 1, Oslo, Norway
{firstname.lastname}@sintef.no

ABSTRACT: *With the B2B data exchange becoming ubiquitous nowadays, automating as much as possible the exchange of data between collaborative enterprise systems is a key requirement for ensuring agile interoperability and scalability in B2B collaborations. Semantic differences and inconsistencies between conceptual models of the exchanged B2B data hinder agility, and ultimately the interoperability in B2B collaborations. In this paper we introduce a model-driven technique and prototype that support humans in reconciling the differences between the data models of the parties involved in a data exchange, and enable a high degree of automation in the end-to-end data exchange process. Our approach is based on the use of OMG Model-Driven Architecture (MDA) for abstracting platform-specific schemas and instances to platform-independent metamodels and models, specification of transformations at the platform-independent level, and generation of executable mappings for run-time data exchange. This paper presents the MDA-based data exchange framework we have developed, and focuses on the mapping metamodel and the generation of executable mappings from platform-independent transformations. Benefits of the proposed framework include the possibility of the mappings creator to focus on the semantic, object-oriented model behind the different platform-specific schemas and specify the mappings at a more abstract, semantic level, with both specification and execution of data mappings (i.e. design- and run-time mapping) provided in a single, unifying framework.*

KEY WORDS: *MDI, MDA, data exchange, mapping, transformation*

1. Introduction

Improving the level of automation of data exchange between B2B systems is widely regarded as a key enabler for agile interoperability and scalability in B2B collaborations (Bussler, 2003). Techniques and tools have been proposed to improve automation in data exchange in special for concrete representation formats such as XML, however, generic approaches that can easily handle different data formats are currently missing, hindering the agility and scalability of B2B data exchange, and in general the interoperability of B2B systems and applications.

Rather than focusing on a specific data representation format, we introduce a generic technique and tool for design- and run-time support of B2B data exchange, based on the OMG Model Driven Architecture[1] approach. The use of model-driven approaches for data mapping/exchange hasn't been yet widely investigated in the community. With this paper we aim at providing a solution to the end-to-end data exchange problem based on the use of OMG MDA framework which we use for high-level, abstract specification of schemas and mappings between them, as well as for run-time execution of mappings.

Figure 1 provides an overview of the elements involved in a typical B2B data exchange between two companies X and Y. Company X, the initiator of the exchange, wants to send the Source Instance (e.g. an invoice document) to Company Y. The Source Instance document is compliant with a schema, Source Schema (e.g. an invoice schema), made available by Company X such that the receiver of its instance document, Company Y, can understand the structure and meaning of such a document. Company Y processes data (i.e. Target Instance) according to its own schema Target Schema. In the typical case that the target schema differs from the source schema, then company X is faced with the problem of having to process the Source Instance document which it does not understand.

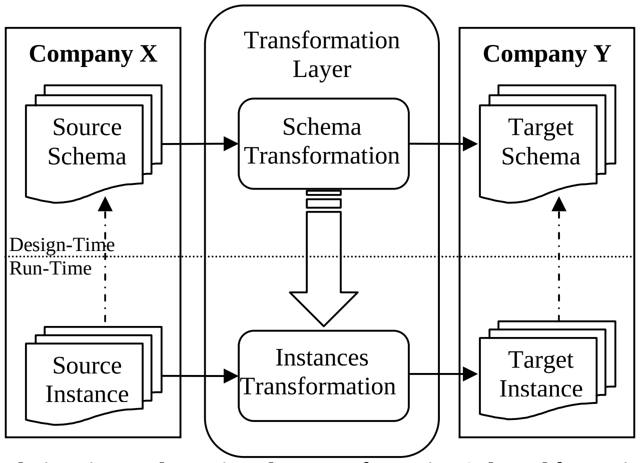


Figure 1. Generic design-time and run-time data transformation (adapted from Liao et al, 2010)

[1] http://www.omg.org/mda/

The core challenge in such as scenario is to generate the Target Instance document from the Source Instance document, given the Source Schema and the Target Schema. A Transformation Layer is usually designed to address this challenge by providing means to map the Source Schema to the Target Schema at design time, and by providing an engine that implements the schema mappings at run time when the Target Instance needs to be generated from Source Instance. Since the transformation cannot be fully automated, the core question is how to design the transformation layer in such a way that the human intervention in the specification and execution of mappings is kept at a minimum.

The OMG Model Driven Architecture (MDA) approach appears attractive for addressing the problem of data exchange, in particular for its support in modeling schemas and transformations at a technology-independent level and its capabilities for automatic generation of executable run-time transformations. The OMG MDA approach, in essence, facilitates the mappings creator to focus on the semantic, object-oriented model behind the different schemas and specify the mappings at a more abstract, semantic level, and potentially enables both specification and execution of data mappings (i.e. design- and run-time mapping) in a single, unifying framework. Rather than having to deal with technicalities of specific data formats and their transformations at lower levels, this paper uses the MDA approach to provide a generic a solution to the end-to-end data exchange problem.

The remaining of this paper is organized as follows. Section 2 gives an overview of our proposed generic MDA data exchange framework. Section 3 describes the transformation metamodel – a core element of our framework – and its use in the specification of transformations. Section 4 focuses on the run-time aspects of the data exchange framework, and provides details on the generation of executable mapping rules. Section 5 gives concludes this paper, together with some relevant related work and potential extensions.

2. Generic MDA Data Exchange Framework

Figure 2 below gives an overview of our proposed data exchange framework. The shadowed box in the center of the figure (to which we refer to as *MDI Transform Engine*), is the core part of the framework, where all the specifications and transformations take place at the platform-independent level. The elements outside the shadowed box (source and target schemas, their transformation, the source instance, and the generated target instance) represent platform-specific models and transformations.

In our approach, source and target schemas (e.g. XSDs, data-bases schemas, etc) are abstracted as platform- and technology-independent models. In our framework (and current implementation), they are abstracted as ECore[2] metamodels (depicted as Source MM and Target MM in the figure), which are very closed to UML class diagrams. Both ECore and UML provide advanced object-oriented mechanisms to data modeling (inheritance, references, compositions, etc). In a similar way, the source instances are abstracted to platform-independent models (depicted by Source M in the figure), which are processed by our run-time executable transformations to generate platform-independent target models

[2] http://www.eclipse.org/modeling/emf/?project=emf

(depicted by Target M in the figure), which are then serialized in platform-specific Target Instances.

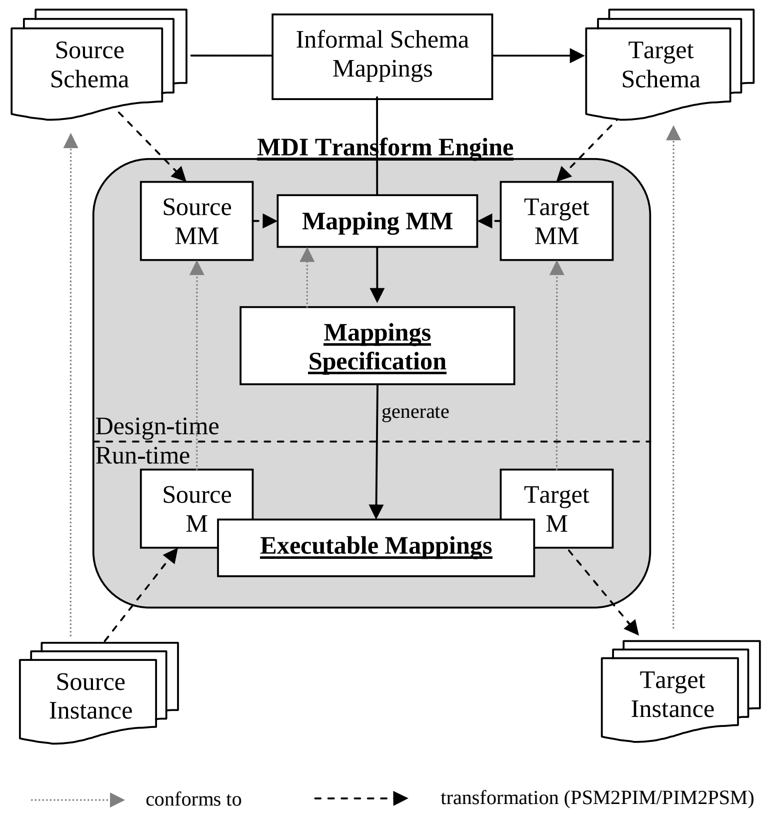


Figure 2. Model-driven Data Exchange Approach – Overview

At the heart of the transformation is a mapping metamodel (depicted by Mapping MM in the figure), which enables us to specify the schema mappings (Mappings Specifications in the figure) in an intuitive way at the platform-independent level, based on the informal schema mappings provided by the user. Furthermore, our framework provides support for automatic generation of executable mappings that are used at run-time to execute the transformation of the Source Model (source data instances) to the Target Model (target data instances). This way, the use needs to focus on the transformations at the platform-independent level, with the actual data exchange being fully automated.

Another way of looking at the framework is through its desing- and run-time elements. The following process takes place when using the framework for data exchange.

At design-time:

1. The platform-specific Source and Target Schemas are abstracted into platform-independent source and target metamodels through a given transformation (PSM2PIM) specific to the concrete technologies used at the platform-specific level.
2. By using the mapping metamodel, the mappings between the source and target metamodels are specified, based on the informal mappings provided by the user.
3. Executable mappings are generated from the mappings specified in the previous step, and will be used during the run-time data exchange.

At run-time:

4. The platform-specific Source Instance is abstracted into a source model through a given transformation (PIM2PSM) specific to the concrete technologies used at the platform-specific level.
5. The executable mapping rules from step 2 are executed for the source model and a target model corresponding to the source model is generated.
6. The target model is serialized into a platform-specific instance target through a given transformation (PIM2PSM) specific to the concrete technologies used at the platform-specific level.

In the following two sections we will focus on the techniques behind the MDI Transform Engine by introducing the mapping metamodel, specifying mappings, and generating executable transformations.

3. Platform-Independent Specification of Transformations

Platform-independent specification of transformations between a source and target metamodel relies on the existence of a mapping metamodel. In this section we introduce our proposed metamodel and exemplifies its use in a simple yet realistic use case.

The Mapping MM, depicted in Figure 3, is inspired by graph-based approaches (Grønmo et al, 2009) and Aspect-Oriented Modeling approaches (Morin et al, 2010). The basic idea is to describe mappings between model fragments (the Left-Hand-Side and the Right-Hand-Side), at the instance level yet in a platform-independent manner. Working at the instance level allows us to easily express constraints, filters or patterns on the mappings. The mapping framework comes with a set of default Mapping Operations, which implements default mapping behaviors. In addition to classic operators (e.g. copy, split, merge specific attributes) we also provide composite adaptations which allow for example to copy all the similar attributes (same name, same type). The metamodel is easily extensible and users can customize it as needed:

- By defining generic adaptations that can be applied to any input and output metamodels. In this case, these extensions can directly be integrated into the generic framework, so that any sub-sequent derivation of the framework will benefit from these adaptations.

- By defining domain-specific adaptation that takes into account the specific semantics of the given input and output metamodels. In this case, these extensions will only be integrates into the specific framework.

Since the LHS can match multiple times, we complement mappings with instantiation strategies we introduced in (Morin et al, 2010). This allows designer to control the way the elements of the RHS should be instantiated: every time the LHS matches, once, etc. By default, all the elements of the RHS are instantiated every time the LHS matches. The *global instantiation* strategy allows designer to specify that some elements of the RHS are global *i.e.*, only one instances of these elements will be created, even if the LHS matches several times. The *scoped instantiation* allows designers to relate the instantiation of some RHS elements to some LHS elements. Typically, this strategy allows handling overlapping mappings: if a RHS element has already been created and associated to a LHS element in a previous mapping, then it will be reused and not duplicated.

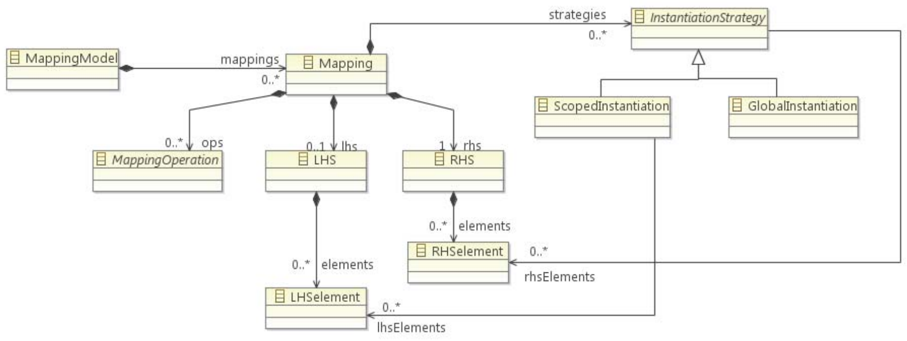


Figure 3. Mapping Metamodel

Once the mapping framework has been automatically contextualized, designers can use it to define mappings. We adopt an interactive approach that guides the designer when defining mappings. The idea is to automatically infer some mappings that the user can validate, discard, or modify. We currently use simple string-based heuristic to identify potential mappings. However, since our approach is based on platform-independent models, it could easily benefits from the theory of model typing (Steel and Jézéquel, 2007), or from an adaptation of the theory of bi-simulation (Nejati et al, 2007), adapted to class diagrams. The idea would be to automatically identify fragments that match in both metamodels, based on advanced heuristics.

To illustrate the use of the metamodel for specification of transformations consider the following example where invoices are send from a source system to a target system. By a given transformation, the source invoice schema is abstracted in the ECore metamodel depicted in Figure 4. Invoices can be either simple or composite invoices. Simple invoices contain some statements (with the price), while composite contains other invoices (simple or composite). Each invoice has a date and a status, modeled as a string.

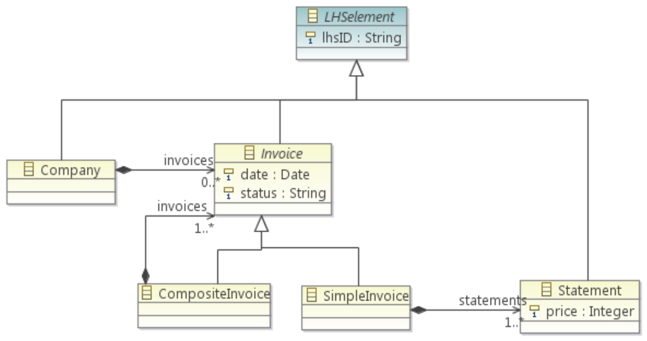


Figure 4. Source MM linked to the Mapping MM (via LHSelement)

By a given transformation, the target invoice schema is abstracted to the ECore metamodel depicted in Figure 5. Here, the invoices have a flat structure and also contain some statements. Note that the date attribute is hold by the statements and not by the invoice. The state of the invoice is reified by the State abstract class, which has two concrete sub-classes: Payed or Waiting.

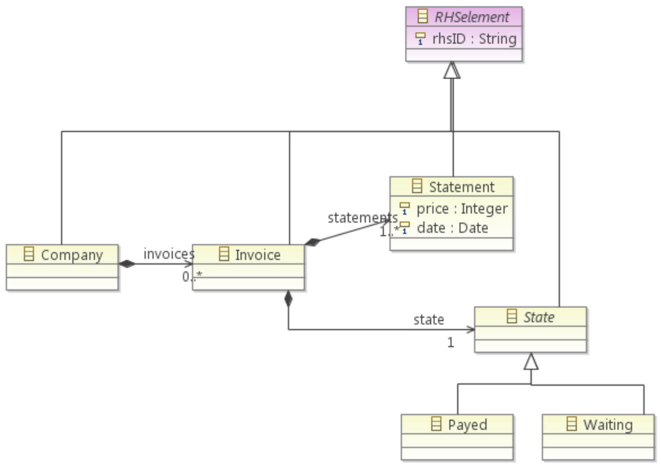


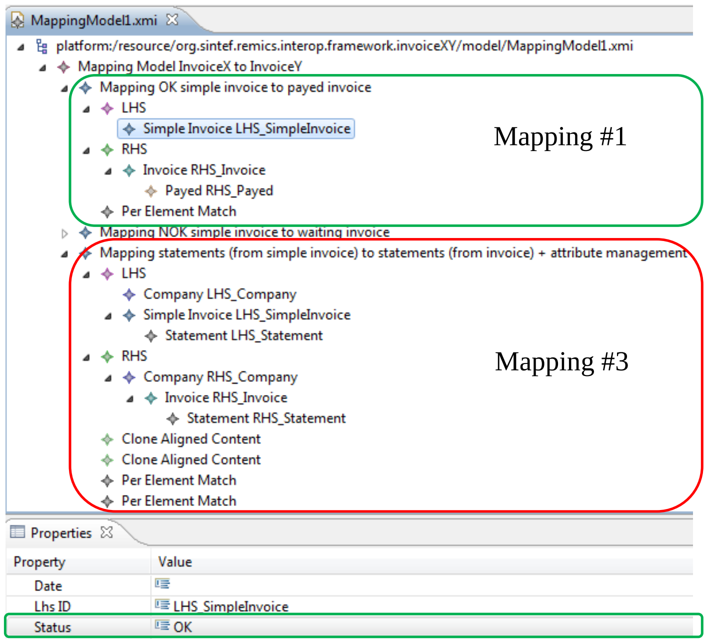
Figure 5. Target MM linked to the Mapping MM (via RHSelement)

The *informal mappings* between the source and the target schemas are as follows:

1. All the source invoices with an “OK” status are used to generate target invoices with *Payed* state.
2. All the source invoices with an “NOK” status are used to generate target invoices with *Waiting* state.
3. All the source simple invoices and their source statements are used to generate target invoices associated with target statements as follows: the price of the target statements should be copied from the corresponding source statements

and the date of the target statements should be copied from the corresponding source invoices.

These informal mappings are captured in our framework by instantiating the mapping metamodel, as illustrated in Figure 6.



```
platform:/resource/org.sintef.remics.interop.framework.invoiceXY/model/MappingModel1.xmi
  Mapping Model InvoiceX to InvoiceY
    Mapping OK simple invoice to payed invoice
      LHS
        Simple Invoice LHS_SimpleInvoice
      RHS
        Invoice RHS_Invoice
          Payed RHS_Payed
      Per Element Match
    Mapping NOK simple invoice to waiting invoice
    Mapping statements (from simple invoice) to statements (from invoice) + attribute management
      LHS
        Company LHS_Company
        Simple Invoice LHS_SimpleInvoice
          Statement LHS_Statement
      RHS
        Company RHS_Company
          Invoice RHS_Invoice
            Statement RHS_Statement
      Clone Aligned Content
      Clone Aligned Content
      Per Element Match
      Per Element Match
```

Property	Value
Date	
Lhs ID	LHS_SimpleInvoice
Status	OK

Figure 6. Mappings specification

In the first mapping the left-hand side is a simple invoice with its attribute status equals to "OK." This will basically match any simple invoice from the source data and filter these data according to the value of the status attribute. The second mapping is similar. The third mapping uses two "Clone aligned content" adaptations, which will actually copy all the aligned attributes (same name, compatible types) of the source into the target. The engine currently provides a limited support (based on the name of the classes) to infer these adaptations. Since the mappings are overlapping, the use of "Per Element Match" strategies makes it possible to reuse some elements that have been created by a previous mapping, in order to avoid inconsistent duplication of data.

4. Generation of executable mappings and run-time data exchange

For the platform-independent mappings specifications (such as those in Figure 6) to be usable at run-time for data exchange, executable transformations need to be generated. Our framework has been designed to automatically compile executable code (Java and Drools

Expert[3]) from the specifications of mappings, using a 2-pass visitor implemented in Kermeta (Muller et al, 2005). The first pass declares model-elements and manages their attributes, while the second pass is responsible for managing (potentially cyclic) references among elements. This generated code directly manipulates graph of objects (Source M and Target M) in memory. We use the persistency API of the source and target systems to load and save data from/to different sources e.g, EMF models saved into XMI, XML files, data bases, etc.

The following script illustrates the result of the first mapping described in the previous section. The *when* clause corresponds to the LHS. We use Drools Expert to automatically identify the pattern defined in the LHS of the mappings. Drools implements and extends the Rete algorithm with object-oriented optimizations. We can generate Drools code from any arbitrary pattern. The more precise the pattern is, the more reduced the set of matched places will be. This could be useful to migrate a precise sub-set of the input data. The first line of this clause specifies that the rule is looking for any simple invoice, which have a status equals to "OK." The second line of this clause is actually not used for this mapping. It re-declares the simple invoice variable and links it to the former declaration (*this* ==...). In more complex mappings including references among elements (such as the 3rd mapping), this second declaration is responsible for handling references among elements. Since Drools (similarly to Java) does not allow referring to variables which are declared after a given statement, this two-staged declaration of variables makes it possible to handle (potentially cyclic) references.

```
rule "OK_simple_invoice_to_payed_invoice"
when
    LHS__SimpleInvoiceDecl: SimpleInvoice(status == "OK")
    LHS__SimpleInvoice: SimpleInvoice(this == LHS__SimpleInvoiceDecl)
then
    Invoice RHS__Invoice = null;
    Payed RHS__Payed = null;

    //Code dealing with the instantiation of the elements,
    //depending on the strategy. See Morin et al. MODELS 2010.

    RHS__Invoice.setState(RHS__Payed);
end
```

The *then* clause of the script corresponds to the RHS. All the elements of the RHS are first declared and set to null. Then that are properly instantiated according to their associated strategy, as described in (Morin et al, 2010). Finally, the mapping operations are compiled into *set* primitives that properly manage the attribute and references of the RHS elements.

This script is finally executed on source data such as the one illustrated on the left side of Figure 7.[4] By applying the script, we obtain the data depicted on the right of Figure 7. In particular, the composite structure of the have been flattened, and the string-based status have been transformed into explicit states. The overall execution takes about 5 seconds for mapping 2000 statements (serialized in around 5000 lines of XMI) on an Intel Core i7-620M@2.67GHz, 8Gb RAM, Win7 64 bits.

[3] http://www.jboss.org/drools/drools-expert.html

[4] The relationships between instances of *SimpleInvoice* and instances of *Statement* have been omitted for clarity purpose. The simple invoices contain the statements located in the same column.

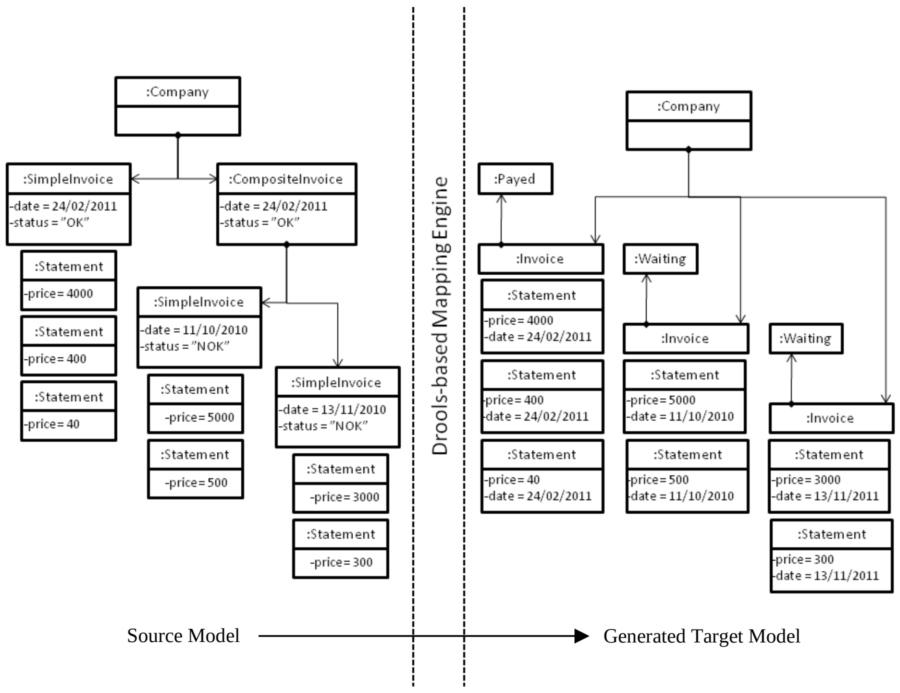


Figure 7 Runtime Data Exchange

5. Related Work, Conclusions, and Outlook

Although the problem of mapping between data structures has been extensively studied for many years now, with schema mapping being a well established research field (Bernstein and Melnik, 2007; Smith et al, 2009), the use of OMG MDA for data mapping/exchange hasn't been yet widely investigated in the community. With this paper we provided a preliminary solution to the data exchange problem based on the use of OMG MDA as a mechanism for platform-independent specification of schemas and mappings between them, and generation of executable mappings to be used at run-time for data exchange. Our approach allows the mappings creator to focus on the platform-independent models behind the platform-specific schemas and specify the mappings at a more abstract, semantic level, rather than having to deal with technicalities of platform-specific schemas. With the use of automatic generation of executable mappings, the proposed framework allows both specification and execution of data mappings (i.e. design- and run-time mapping) in a single, unifying framework.

Our approach is inspired by the SmartAdapters aspect model weaver (Morin et al, 2010). While Aspect-Oriented Modeling approaches (Whittle et al, 2009; Reddy et al 2006) are usually concerned with endogeneous weaving, our approach is focused on the interoperability of exogeneous (yet, with some similarities) data. An aspect-model weaver basically takes two models conforming to the same metamodel and produces a third model, also conforming to the same metamodel. Our model-driven mapping engine takes a model conforming to a given metamodel and produces a model conforming to another metamodel. Despite this difference, we were able to reuse the SmartAdapters tool-chain in our framework (especially the Drools+Java code generator) with minor modifications.

Our approach to Model-Driven Interoperability is based on the definition of mappings at the instance level. More precisely, we define mappings between patterns (fragments of model). This allows designers to easily filter data based on patterns, or on the values of attributes in order to identify different sub-sets in the source data that should be migrated according to different strategies. However, one of the main drawbacks of working at the instance level is that dedicated tools (such as advanced graphical editors) should be implemented or adapted for each source/target metamodels. Different from our approach, ModMap (Clavreul et al, 2010) proposes to define mappings at the metamodel level. Mappings are defined in a graphical way by defining links between classes defined in ECore diagrams. These tools and the mappings have been used to generate some AspectJ code in order to align APIs of legacy systems (written in Java) and enable their interoperability. It could however be possible to generate other type of code e.g., the code we generate to enable the interoperability of data. While this tool is very useful and easy to use to define mappings between classes, it is very cumbersome to define mappings among patterns (classes and references).

The work presented in this paper can be considered in the wider context of MDE model transformation techniques and languages (Czarnecki and Helsen, 2003; Mens and Van Gorp, 2006) such as ATL Transformation Languages (ATL).[5] ATL also relies on graph-based transformations, similarly to our approach. However, our approach provides high-level constructs (such as the strategies) and mapping operators to control and ease the definition, which should be coded by the designers in ATL to achieve the same result. Another difference is that ATL, as well as most of the model transformation languages focuses on the manipulation of EMF-based models. Our approach intends to manipulate data coming from different sources such as XML files or databases.

The recent work in (Liao et al, 2010) provides a solution (FloraMap) to the data exchange problem with a focus on XML data exchange. It proposes the uses a logical rule language (Frame Logic) for formalization of schemas, instances, and transformations. Whereas it addresses the same problem and goes in the same direction of using a platform-independent approach for specification and execution of mapping, the approach presented in this paper strives to be more general through the use of well established OMG MDA technologies.

To summarize, in this paper we provided a preliminary report on the development of the data exchange framework. The preliminary experimental results indicate that the approach presented here is doable in practice. Nevertheless, there are various directions that can be considered to further enhance the framework presented in this paper:

[5] http://www.eclipse.org/atl/

1. Extensions for handling end-to-end n-m data exchanges, where multiple sources and multiple targets can exchange data.
2. (Semi-)Automated generation of platform-independent mapping rules, where techniques from e.g. ontology alignment techniques could be reused.
3. Address the issue of incompleteness of mappings by a coverage analysis which will ensure that all mandatory attributes are mapped.
4. Investigate possible synergies between instance-level mappings and metamodel-level mappings in order to mitigate respective drawbacks and combine respective advantages.
5. Systematic validation – whereas we performed some initial experimental results for the scalability of our framework, our approach need to be analyzed in a more systematic way (e.g. analyze the complexity of the specification of mapping rules, etc).

Acknowledgements. This work is partly funded by the EU projects “A Semantic Service-oriented Private Adaptation Layer Enabling the Next Generation, Interoperable and Easy-to-Integrate Software Products of European Software SMEs (EMPOWER)” and “Reuse and Migration of legacy applications to Interoperable Cloud Services (REMICS).”

References

Bernstein P A, and Melnik S, *Model management 2.0: manipulating richer mappings*. In Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data (Beijing, China, June 11 - 14, 2007).

Bussler C, *B2B Integration*. 2003, Springer, ISBN 3540434879.

Clavreul M, Barais O, and Jézéquel J-M. *Integrating legacy systems with MDE* In ICSE'10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering and ICSE Workshops, volume 2, pages 69--78, Cape Town, South Africa, May 2010.

Czarnecki K and Helsen S, *Classification of Model Transformation Approaches*. In: Proceedings of the OOPSLA'03 Workshop on the Generative Techniques in the Context Of Model-Driven Architecture, Anaheim, California, USA.

Grønmo R, Krogdahl S, and Møller-Pedersen B, *A Collection Operator for Graph Transformation*. In ICMT’09: 2nd International Conference on Theory and Practice of Model Transformations, pages 67–82, Berlin, Heidelberg, 2009. Springer-Verlag.

Liao Y, Roman D, and Berre A-J, *Model-driven Rule-based Mediation in XML Data Exchange*. In: Proceedings of the First International Workshop on Model-Driven Interoperability. MDI 2010, Oslo, Norway, ACM (2010) 89–97.

Mens T and Van Gorp P, *A Taxonomy of Model Transformation*, Electronic Notes in Theoretical Computer Science, Volume 152, 27 March 2006, Pages 125-142.

Morin B, Klein J, Kienzle J, and Jézéquel J-M, *Flexible model element introduction policies for aspect-oriented modeling*. In Proceedings of ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2010), Oslo, Norway, October 2010.

Muller P A, Fleurey F, and Jézéquel J M, *Weaving Executability into Object-Oriented Meta-languages*. In MoDELS'05: 8th Int. Conf. on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, Oct 2005.

Nejati S, Sabetzadeh M, Chechik M, Easterbrook S M, Zave P: *Matching and Merging of State charts Specifications*. ICSE 2007: 54-64.

Reddy Y R, Ghosh S, France R B, Straw G, Bieman J M, McEachen N, Song E, and Georg G., *Directives for Composing Aspect-Oriented Design Class Models*. In Awais Rashid and Mehmet Aksit, editors, Transaction on Aspect-Oriented Software Development, volume vol 3880 of Lecture Notes in Computer Science, pages 75–105. Springer, 2006.

Smith K, Mork P, Seligman L, et al. *The Role of Schema Matching in Large Enterprises*, CIDR Perspectives 2009.

Steel J and Jézéquel J-M. On model typing. *Journal of Software and Systems Modeling (SoSyM),* 6(4):401–414, December 2007.

Whittle J, Jayaraman P K, Elkhodary A M, Moreira A, and Araújo J. *Mata: Aunified approach for composing uml aspect models based on graph transformation*. T. Aspect-Oriented Software Development VI, 6:191–237, 2009.