Applying a Model-driven Approach to Model Transformation Development

E. Victor Sánchez Rebull[1], Orlando Avila-García[1], José Luis Roda García[2] and Antonio Estévez García[1]

[1] Open Canarias, S.L., C/. Elías Ramos González, 4 - Oficina 304
38001 Santa Cruz de Tenerife, Spain

[2] Universidad de La Laguna
La Laguna, Spain

Abstract. One of the cornerstones of MDA is the specification and execution of model transformations. This paper proposes a practical application of MDA to the development of model transformations. This approach involves transforming instances of different transformation languages with differing levels of abstraction in a PIM-PSM style. By means of two case studies, we discuss technical details and assess what possible gains it can offer in terms of productivity and maintainability.

1 Introduction

MDA (Model-Driven Architecture) [13] proposes a scheme of successive transformations between formal models describing software. These models represent the software at different levels of abstraction and degrees of platform independence, as in the classic Platform Independent vs Platform Specific Model (PIM-PSM) separation. A set of standards such as MOF (Meta-Object Facility) [14] for metamodeling and QVT (Query/Views/Transformations) [15] for model transformation specification, exists in MDA to sustain the application development cycle centered on these models as first-class artifacts. Thus, creation and execution of model transformations play a central role in MDA. This implies the need to designing model transformation languages (such as QVT) and facilities to create, manipulate and execute their instances.

This paper presents a practical MDA approach to model transformation development, which is the application of the MDA paradigm to the development of model transformations (Sect. 2). We discuss two case studies (Sect. 4) to reflect its advantages, after introducing a platform specific transformation language in Sect. 3. The first one shows one solution approach to execute instances of QVT, as a general purpose model transformation language; the second one shows the execution of instances of MTTL (Model Template Transformation Language) [2], a domain specific transformation language. Related work and other approaches are presented in Sect. 5 and conclusions in Sect. 6.

Victor Sánchez Rebull E., Avila-García O., Luis Roda García J. and Estévez García A. (2007).
Applying a Model-driven Approach to Model Transformation Development.
In *Proceedings of the 3rd International Workshop on Model-Driven Enterprise Information Systems*, pages 53-62
DOI: 10.5220/0002430200530062

2 MDA Approach to Model Transformation Development

By applying a MDA approach to model transformation development we mean splitting up the process of specifying executable transformations in different levels of abstraction (as explained in [4]). This implies considering model instances of model transformation languages as PITs (Platform Independent Transformations) which are then transformed onto lower level transformation languages; which in our case is ATC (Atomic Transformation Code) [9]. Transformation models expressed in ATC are tied to a specific runtime platform (which executes them) named VTE (Virtual Transformation Engine), hence they can be considered PSTs (Platform Specific Transformations). This approach allows us to bring support to the execution of other model transformation languages' instances in VTE indirectly through ATC.

3 ATC as Platform Specific Transformation Language

ATC (see above and [9]) is a general purpose low-level ATC transformation language. Its instances take the shape of models conforming to the ATC metamodel, the most recent version of which can be found at [7].

ATC models are executed by the VTE model transformation engine, a virtual machine created with this sole purpose. VTE is implemented as a thin software layer built on top of EMF [5], which is considered to be a MOF implementation. VTE abstracts from its API those EMF fundamental components that thoroughly combined represent each of a set of low-level model transformation primitives called atom types in the ATC jargon. These primitives form up the ATC instruction set. Although ATC models are easily manipulated by hand with the default EMF tree editor, they are usually obtained as the result of an (automated) transformation development process.

It is expected that ATC contains all the necessary model transformation mechanics so it is feasible to map (translate) complete model transformation languages' semantics to equivalent ATC constructs. This includes general purpose languages such as QVT, and any languages that may cover a wide range of specific or particular domains of application, like the MTTL explained in Sect. 4.2. This translation will be discussed in Sect. 4.

3.1 How Much PST is ATC

We claim that ATC models are platform-specific. Indeed ATC is tied to VTE, as its target runtime platform. This PST condition will remain unconditionally true for the purpose of this work (and according to [4]) even with the possibility of unfolding the virtual engine into native java code specific to each ATC instance (see [9]), or if other engine implementations are constructed that support the ATC language, over the same EMF or (porting VTE to) other metamodeling facilities such as NetBeans MDR [12], for which the ATC language should not suffer almost any modifications to its design.

The ATC's PST nature helps us emphasize its low-level orientation and put it in contrast alongside other model transformation languages, which in their majority can be considered of a higher abstraction level, and which in general preserve a complete

neutrality from any known technology or platform. As alternatives to ATC as low-level model transformation languages we will mention ATL-VM [10], which has implementations based on at least two different metamodeling infrastructures (the both aforementioned EMF and MDR).

4 Platform Independent Transformation Languages

For transformation languages whose instances can have a model representation, nothing prevents us from formalizing a (collection of related) transformation(s) that translate them to semantically equivalent ATC models, much in the sense of the MDA's PIM to PSM transformation paradigm. Thus, regardless of which Platform-Specific model transformation language is finally chosen for our execution, if any, PITs fulfill their goal of being static artifacts reusable through different underlying transformation engines or tools.

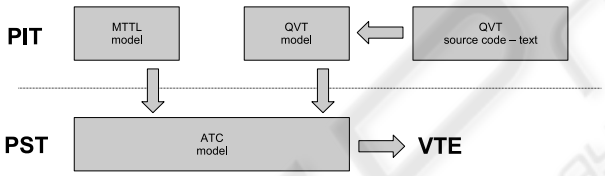

Fig. 1. Instances of MTTL, a domain specific transformation language, are treated as PITs. Textual instances of QVT, a standard and general purpose transformation language, are converted to models (conforming to the QVT metamodel or abstract syntax) which in turn may be considered PITs.

In this section we show the benefits coming from taking a MDA approach to model transformation development by means of two case studies (see Figure 1). The first case shows a solution to indirectly execute instances of QVT, a general purpose and standard model transformation language (Sect. 4.1), in the VTE engine; likewise, the second one shows the execution of instances of MTTL (Model Template Transformation Language), a domain specific transformation language (Sect. 4.2).

4.1 The General-purpose QVT

As already mentioned, notation to express model transformation semantics in MDA is specified in the QVT standard. This specification details an architecture composed by three languages. Relations is a declarative language where mappings can be expressed between related domains. Also declarative, Core is a lower-level language that offers the same semantic capabilities as Relations.

Finally, Operational Mappings (OM) is the only imperative QVT language. It serves two purposes. Firstly it assists Relations to complement certain rules by providing procedural implementations that guide QVT engines on how these rules must be resolved. This is particularly useful whenever it is difficult, if not impossible, to properly express

those declarative rules in Relations. Secondly it can be used as a standalone imperative model transformation language

This case study addresses ways to support QVT in VTE in terms of OM, although the discussion is applicable to any other textual model transformation language, as long as it has a well defined metamodel for which conforming executable models can be formalized.

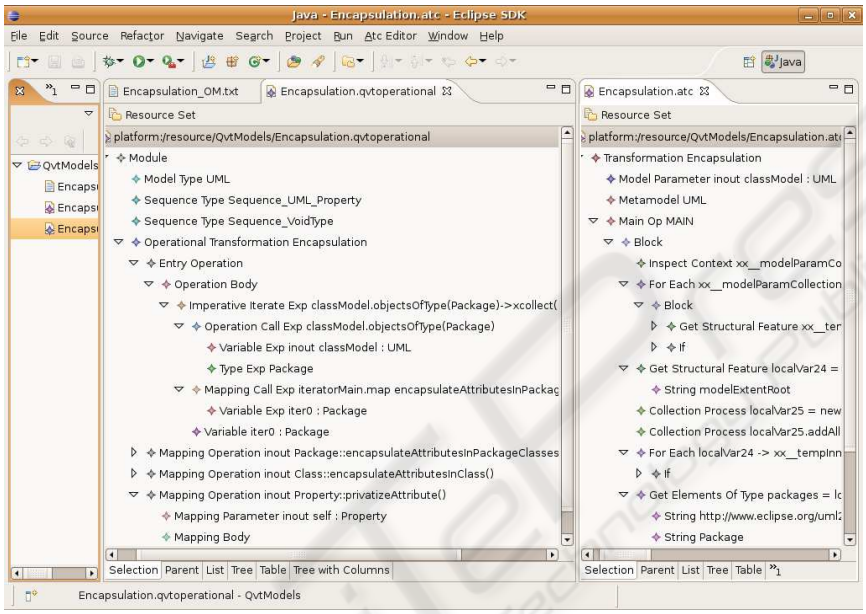

Fig. 2. A model conforming to the OM's metamodel as PIT to the left and its transformed ATC semantically equivalent PST model to the right.

Model transformation languages can be supported in VTE by, first, parsing the textual instances into an AST (*abstract syntax tree*) and, second, traversing the tree to generate the equivalent atoms that composed one or more ATC transformations, basically using an automatically generated parser [9]. An alternative translates the AST to a model conforming to the metamodel of the transformation language (an example of such model is shown in Fig. 2). This model (which represents a transformation!) can then be checked and optimized before it is finally transformed onto ATC. It can also be stored within model repositories and even shared among tools, as if it were an ordinary model.

In this scenario we are rightfully applying the MDA principles in terms of PIT to PST transformations. Moreover, if the AST created by the textual syntax parser comes also to be a model conforming to certain metamodel (for instance, GASTM [16]), then we have another model in the transformation chain to obtain ATC instances. This is a clear example of a practical PIM to PSM MDA transformation development process in

the context of MDA. Here the model paradigm is applied in two stages. First, to the process of obtaining the model of the source language from the AST model, once it is automatically produced from the source text. Second, to the translation from this transformed model to the ATC model that is the output target artifact for the whole process, and which represents an executable transformation suitable for the VTE environment. Both transformation steps supersede the traditional semantic parsing execution stage that analyses the inferred AST to generate output.

By specifically taking advantage of model transformation tools and languages and exploiting their model-orientation characteristics, we can get abstracted from a significant amount of functionality and increase the overall productivity. Also, efforts invested towards this goal are expected to be paid off by succeeding in somehow reusing these transformatons to support other languages.

Operational Mappings vs. ATC. ATC is (at least almost) as low level as it can be, so it may be considered a reference point in an absolute scale that measures a transformation language's level of abstraction. Making an estimation about how distant from ATC is OM in this scale, or starting a study to obtain metrics about its level compared to other high-level languages, could be interesting to calibrate OM's absolute abstraction level. While such a study is beyond the scope of this work, it is easy to infer that subtle mechanisms involved in an OM's transformation execution such as trace instance management, and instantiation sections in mapping operations, put the language on a rather high level starting point. As per a sentence by sentence comparison, there is much complexity involved to give an approximate rate of sentence versus number of combined ATC atoms representing the same semantics, but they often comprise two or three of them (worst cases can raise this figure above twenty). However, to be fair we should not be counting sentences in OM against ATC atoms, since each of them usually comprises several model elements. To present some figures regarding model element count, a typical Encapsulation transformation example gives about 115 elements in OM vs. about 220 in ATC, which increases over 260 if we include the OM's implicit execution trace information tracking.

4.2 The Domain-specific MTTL

As explained before, QVT [15] is a standard transformation language that is expected to be used across companies and projects. Its general-purpose nature allows us to specify different types of transformations. Although the benefits coming from using such a language are clear, many studies exist showing the advantages of rising the level of abstraction by introducing Domain-Specific Transformation Languages (DSTL), as when using any other domain specific language (DSL) [2]. In this case study we show MTTL (Model Template Transformation Language), a domain-specific language to create software product lines of models [2], and a MDA approach to execute its instances.

Software product lines of models are an effective solution to maintain model families in Model Driven Engineering. An approach to implement this kind of product lines is based on the use of model templates containing all model variants of the family in a superimposed form [6]. In order to obtain a concrete family member, a model transformation automatically specializes the model template by purging relations between

model elements. The specialization is carried out following a set of selected features that are specified aside, in a feature model which characterizes the model family.

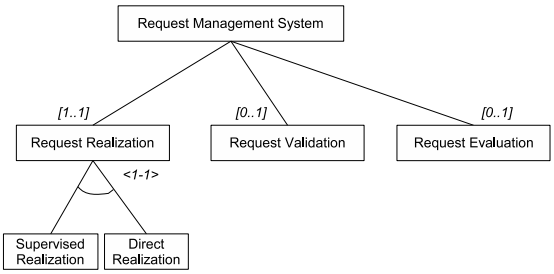

Fig. 3. Feature Model in Cardinality-Based Feature Model notation [8] for a family of Request Management Systems (RMS). *Request Validation* and *Request Evaluation* are optional features in this family; feature *Request Realization* is mandatory, while *Supervised Realization* and *Direct Realization* are alternative features. By selecting optional and alternative features from the feature model we are able to identify up to eight different variants of RMS.

Model Template Transformation Language (MTTL) is a DSTL that has been developed to ease the engineering of product lines of models using a template approach. MTTL is an imperative language with four basic operations to manipulate object relations within model templates: REMOVE, SELECT, CLEAR and INSERT.

In this case study, we are to create a software product line of use case models in the domain of request management systems (RMS). In our experience, those systems share many commonalities while vary along well defined characteristics. Therefore, the development of such systems may greatly benefit from the use of a software product line approach. In this example we are to create a software product line that automates the analysis phase in the development of a family of RMS systems. This product line will maintain a family of UML Use Case models that represent the analysis solution to those systems. Fig. 3 shows a feature model describing the common and variable features of the family.

Following [6], we use a template approach to implement this software product line. First, we create a model template containing all variants of use case models in a superimposed form (see Fig. 4). Each use case model, solution to the analysis of a single RMS, is obtained by deleting object relations from the model template. Secondly, we define the transformation that shall specialize this model template when optional and alternative features from the feature model are selected. It is here where we take advantage of using a DSTL in combination with a MDA approach to transformation development: we specify such a transformation in MTTL (see Fig. 5) and transform it onto ATC in order to execute it.

Figure 5 shows the MTTL model we have created to solve the problem and its corresponding ATC model. While the former is made of six objects, the latter contains more than three hundred. Moreover, the use of MTTL, a DSTL, is straight and easy, abstracting us from implementation details such as the metamodeling framework. This is a clear

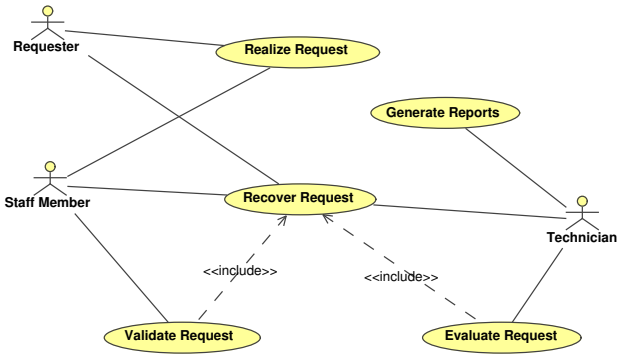

Fig. 4. Template for a family of use case models. It is a use case model itself, containing all the family members in a superimposed form, so it contains all objects and relations belonging to any of the model variants. By purging a subset of these objects and relations it is transformed (specialized) to generate those family members.

example of time savings when using domain specific languages to specify transformations, which in this case builds on top of benefiting from using transformation models, as discussed in Sect. 4.1. Another important point is that the transformation between MTTL and ATC was developed once, when MTTL itself was created, and took much less effort than building up a compiler or a transformation engine exclusively for it (see [2]).

5 Related Work and Other Approaches

There have been some other examples of applying a MDA approach to model transformation development. For example, [3] proposes an extensible weaving metamodel and its transformation onto general purpose transformation languages. This metamodel may be extended to become a domain-specific language tailored to specify mappings or relations between model objects in different domains [3]. For example, [11] proposes an extension to specify model transformations at a higher level of abstraction as mappings between metamodel elements. In that particular study, the instances of the extension where transformed onto ATL [10]. That mapping language differs from our MTTL in two respects: first, the former links objects at the metamodeling level (M2) while the latter does it at the modeling level (M1); and secondly, MTTL is suitable for the domain of transformations of model templates following feature models, a domain much more specific that the former's.

There exist other approaches to solve the problem of model transformation production in QVT or in DSTLs such as MTTL. For example, textual instances of QVT might be compiled into Java to be executed. Likewise, QVT or DSTL models might be transformed onto Java code by model-to-text transformations. The problem with these

[3] See [1] for an extension implementing MTTL.

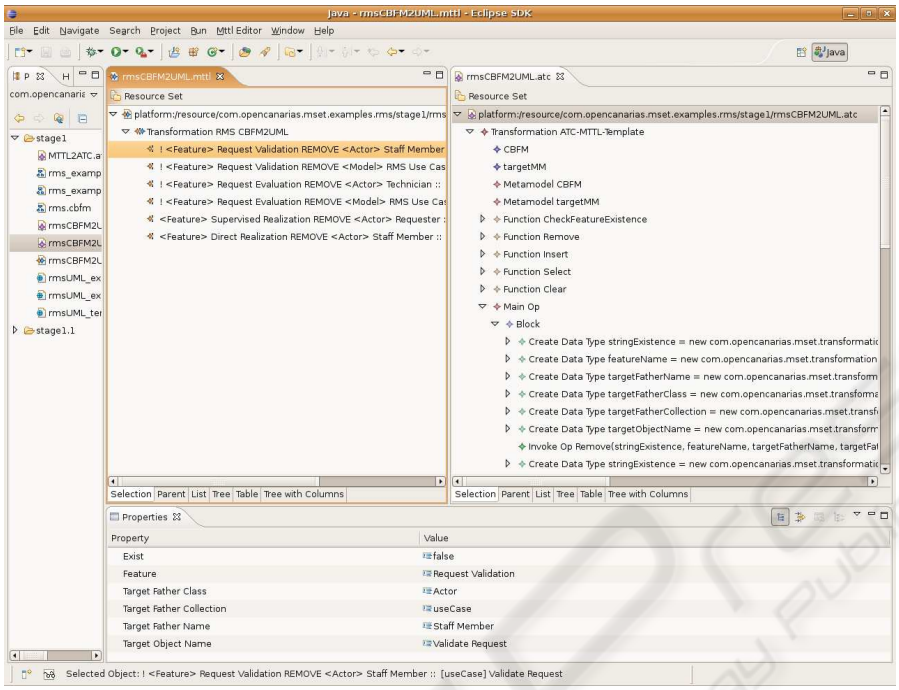

Fig. 5. A MTTL mapping model on the left. It describes the specialization of the model template of Fig. 4 after the selection of features from the feature model of Fig. 3. It contains six elements, all specifying REMOVE operations, that is, the purge of model template objects. On the right, corresponding ATC model. The transformation of MTTL instances onto ATC is carried out by filling up an ATC template containing a function per MTTL operation. Each mapping object from the MTTL model is then translated into a call operation (*invoke* object) to the corresponding function with the proper parameters.

approaches is the inability of reuse: none of the efforts applied in the development of a compiler or model-to-text transformation may help in the development of the others. An alternative approach implies to build specific engines for those transformation languages. One of the shortcomings mentioned for the previous approach is completely valid here: the inability of reuse in developing these alternative transformation engines may make their development a too costly enterprise.

6 Conclusions

In this paper we have proposed a MDA approach to model transformation development. By exploring two case studies that reflect the benefits of this approach, we have shown ways to transform PITs to PSTs. The first of them showed a solution to allow instances of QVT, a standard and general purpose transformation language, to be indirectly executed in the VTE runtime platform. The second example showed the application of the

same approach to execute instances of MTTL, a domain specific transformation language. Similar transformations applied upon ATC models can transitively extend the compatibility (and support) of the source language to other tools. Conversely, middleware languages and tools can also be used to bring compatibility of distant languages in VTE.

We think that we have shown evidence enough so as to argue that all the benefits expected from MDA concerning model-driven evolution can directly be expanded to the use of this very same approach to model transformation development. Among them we highlight the reuse of transformation facilities, which helps us save costs and increase productivity when creating MDA solutions in our software development projects.

Acknowledgements

Paper supported by the *Ministerio de Educación y Ciencia* (PTQ2004-1495, PTR1995-0928-OP), the *Fondo Social Europeo* and *DGUI, Consejería de Educación, Cultura y Deportes, Gobierno de Canarias* (PI042005/007). Thanks also go to the *Red de Desarrollo de Software Dirigido por Modelos (DSDM)*, ref: TIN2005-25866-E.

References

1. Orlando Avila-García and Marcos Didonet Del Fabro. AMW use case: Mapping features to models. Technical Report MST-9, Open Canarias, S.L., Apr 2007. Available at http://www.eclipse.org/gmt/amw/usecases/softwareproductline/.
2. Orlando Avila-García, Antonio Estévez García, Victor Sánchez Rebull, and Jose Luis Roda García. Using software product lines to manage model families in model-driven engineering. In *SAC 2007: Proceedings of Symposium on Applied Computing, track on Model Transformation*. ACM Press, Mar 2007.
3. Jean Bézivin, Salim Bouzitouna, Marcos Didonet Del Fabro, Marie-Pierre Gervais, Fréderic Jouault, Dimitrios Kolovos, Ivan Kurtev, and Richar F. Paige. A canonical scheme for model composition. In *ECMDA 2006: Proceedings of the European Conference on Model-Driven Architecture*, volume 4066 of *LNCS*, pages 346–360. Springer-Verlag, Jul 2006.
4. Jean Bézivin, Nicolas Farcet, Jean-Marc Jézéquel, Benoit Langlois, and Damien Pollet. Reflective Model Driven Engineering, 2003. Available at http:/www.lina.sciences.univ-nantes.fr/Publications/2003/BFJLP03.
5. Frank Budinsky, David Steinberg, Ed Merks, Ray Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework (EMF)*. Addison Wesley, Aug 2003. ISBN 0-13-142542-0.
6. Krzysztof Czarnecki and Michal Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *Proceedings of GPCE 2005*, volume 3676 of *LNCS*, pages 422–437. Springer-Verlag, 2005.
7. Open Canarias. ATC metamodel, Feb 2007. URL `http://www.modelset.es/files/ATC.ecore`.
8. Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice, special issue of best papers from SPLC04*, 10(1):7–29, 2005.
9. Antonio Estévez, Javier Padrón, Victor Sánchez, and José Luis Roda. ATC: A low-level model transformation language. In *MDEIS 2006: Proceedings of the 2nd International Workshop on Model Driven Enterprise Information Systems*, May 2006.

10. Fréderic Jouault and Ivan Kurtev. On the architectural alignment of ATL and QVT. In *SAC 2006: Proceedings of the Symposium on Applied Computing*. ACM Press, Apr 2006.
11. Denivaldo Lopes, Slimane Hammoudi, Jean Bézivin, and Frédéric Jouault. Mapping specification in MDA: From theory to practice. In *INTEROP-ESA 2005: Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications*, pages 253–264. Springer-Verlag, Feb 2005. Available at http://www.lina.sciences.univ-nantes.fr/Publications/2005/LHBJ05.
12. Sun Microsystems. NetBeans Metadata Repository. Homepage at http://mdr.netbeans.org/.
13. OMG. MDA guide version 1.0.1. Technical Report omg/2003-06-01, Jun 2003. Available at http://www.omg.org/docs/omg/03-06-01.pdf.
14. OMG. Meta Object Facility (MOF) 2.0 core specification. Technical Report ad/2003-10-04, Apr 2003. Available at http://www.omg.org/docs/ad/03-10-04.pdf.
15. OMG. MOF 2.0 Query/Views/Transformations. Technical Report ptc/05-11-01, Nov 2005. Available at http://www.omg.org/docs/ptc/05-11-01.pdf.
16. OMG. Architecture-Driven Modernization (ADM): Abstract Syntax Tree Metamodel (ASTM). Technical Report admtf/2006-11-01, Nov 2006. Available at http://www.omg.org/docs/admtf/06-11-01.pdf.