A MDA Approach to Model & Implement Transformations

Jean-Marc Jézéquel

Irisa (INRIA & Université de Rennes), France

Abstract Only in software and in linguistics a model has the same nature as the thing it models. In software at least, this opens the possibility to automatically derive software from its model. This property is well known from any compiler writer (and others), but it was recently be made quite popular with an OMG initiative called the Model Driven Architecture (MDA). The model transformations allowing the engineers to more or less automatically go from platform-independent models (PIM) to platform-specific models (PSM) are increasingly seen as vital assets that must be managed with sound software engineering principles. We believe that transformations should be first-class models in the MDA world; we propose to adopt the object-oriented approach and to leverage the expressive power of UML as a metamodel defining the transformation language.

1 Introduction

Once upon a time, software development looked simple. In the early 80's, there was a dream that it was possible to unify everything into just one concept, the notion of object. *"Everything is an object"* was a powerful vision that led for instance to Smalltalk, where close to everything, including an integer value such as 5, an instruction, a bank account, a method, a message, a pixel, or a class was indeed an object. Soon enough however, it was clear that while the notion of object could really be a building block, software development required complementary notions. For example if objects are modular entities made to be as simple as possible, then we find complexity lurking in the way objects collaborate to implement a functionality. Trying to document these collaborations in order to make the most interesting ones reusable ultimately led to the very important notion of Design Patterns. Another dimension of complexity lied in the poor marriage between objects and distribution: allowing distributed object to communicate smoothly revealed itself surprisingly difficult and led to the development of middle-ware such as CORBA, or DCOM and now .NET.

So we started with a simple idea ("Everything is an object") and for good reasons we ended up with a fairly complex situation in software development. It is then natural that people resorted to modelling to try to master this complexity. According to Jeff Rothenberg,

> *Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use*

Dagstuhl Seminar Proceedings 04101
http://drops.dagstuhl.de/opus/volltexte/2005/20

something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.

Usually in science, a model has a different nature that the thing it models ("do not take the map for the reality" as Sun Tse put it many centuries ago). Only in software and in linguistics a model has the same nature as the thing it models. In software at least, this opens the possibility to automatically derive software from its model. This property is well known from any compiler writer (and others), but it was recently be made quite popular with an OMG initiative called the Model Driven Architecture (MDA) [6].

The OMG has built a meta-data management framework to support the MDA. It is mainly based on a unique M3 "meta-meta-model" called the Meta-Object Facility (MOF) [5] and a library of M2 meta-models, such as the Unified Modeling Language (UML) (or SPEM for software process engineering), in which the user can base his M1 model.

2 Complex Model Transformations

Large companies having to develop, maintain and evolve large scale software systems over long periods of time are considering strongly the OMG initiative on Model Driven Architecture (MDA) [6]. In the air traffic management domain for instance, the domain specific models are less likely to change rapidly than platform-specific ones. So the MDA core idea that it should be possible to capitalize on platform-independent models (PIM), and more or less automatically derive platform-specific models (PSM) –and ultimately code– from PIM through model transformations is extremely appealing. But in some business areas involving fault-tolerant, distributed real-time computations, there is a growing concern that the added value of a company not only lies in its know-how of the business domain (the PIM) but also in the design know-how needed to make these systems work in the field (the transformation to go from PIM to PSM). Reasons making it complex to go from a simple and stable business model to a complex implementation include:

- Various modeling languages used beyond UML
- As many points of views as stakeholders
- Deliver software for (many) variants of a platform
- Heterogeneity is the rule
- Reuse technical solutions across large product lines (e.g. fault tolerance, security, etc.)
- Customize generic transformations
- Compose reusable transformations
- Evolve and maintain transformations for 15+ years.

Beyond transformations defined as mapping of model level concepts to target level ones, as it is often the case for code generation, XMI generation, documentation generation, database schema generation etc., another dimension of complexity appears when we want to make semantic driven transformations. Exemple of these more profound transformations are:

- Class Diagrams+Statecharts+Snapshots to Tests [3]
- Design Pattern Applications [7]
- Class Diagrams to Integration Test Plan [9]
- Refactorings of UML models [8]
- Weaving Design Level Aspects [2]
- Uses Cases+Contracts to Tests for Product Lines [4]
- Specialization of models of Product Lines [11]
- HMSCs to StateCharts [10]

Until now, model transformations have in most cases been developed within modeling tools using tool-specific proprietary languages. This tool adherence jeopardizes the reusability of domain models which is one of the key advantages drawn from MDA. For the same reason that domain know-how should not be tied to a particular platform, it is thus critical that model transformations are not prisoners of a given CASE tool.

A second approach that has been tried corresponds mainly to the tree-transformation systems such as XSLT. An XSLT script declaratively specifies how to explore the input tree and how to generate fragments of the output tree. Languages like XSLT are heavily used in some domains, but the experience of using XSLT for implementing complex transformations in an MDA context proved itself quite negative with respect to issues such as modularity, efficiency, reusability and above all maintenability.

3 Modeling Transformations

3.1 Requirements for a Transformation Meta-Model

Although models capture the design features of a product, model transformations capture the model manipulation expertise, e.g. specific mechanizable refinement steps during the product design, the chain of successive refinements needed to obtain a concrete product, or how to implement PIM from a given metamodel using a specific platform.

For large companies, such expertise represents a long-term investment, and the initial cost of developing model transformations should be balanced over time, when transformations can be reused at a negligible cost compared to an ad hoc solution.

However the increasing complexity due to adaptation and evolution of transformations should not jeopardize reuse and thus return on investment. The problem is therefore to identify techniques and methods enabling transformation development and maintenance, by addressing at the M2 level standard M1

level notions such as reusability, composability, genericity, customizability and maintenability.

Because transformations become complex software products, we should develop them using established software engineering techniques. This idea led to the notion of reflective Model Driven Engineering presented in [1].

3.2 Using UML to Design Transformations

We believe that transformations should be first-class models in the MDA world. We then propose to adopt the object-oriented approach and to leverage the expressive power of UML as a metamodel defining the transformation language. Both as a modeling language and as a management tool, UML provides concepts useful to analysis, design, and development of transformations:

Model management diagrams address the macro-organization of the transformation components in UML packages.

Class diagrams reveal the structure and the patterns in the transformation design. Rules are expressed as operations, organized in classes and packages. Subclassing and dynamic binding can be used to handle variability, for example by leveraging the classical design patterns.

Activity diagrams express the transformation process by capturing the dependencies between transformation subtasks and can be used to combine multiple transformations.

Deployment diagrams specify platform-specific aspects, e.g. which CASE tool should be used to handle models of a given metamodel.

UML alone cannot model transformations directly, so a profile for transformations should be defined. Besides the use of UML, developers will eventually use a specific runtime platform to actually transform models; however the choice of a runtime platform should not impact on the transformation design. This is in line with the separation of concerns between PIM and PSM in the MDA, and leads to the concepts of platform-independent transformation (PIT) and platform-specific transformation (PST).

Platform-independent transformations are models of the transformation program, relying on a generic library of simpler transformations and transformation primitives. They will eventually be refined to the point where they can be used as the source to generate platform-specific transformations. Then if UML is the language of PIT, PST are models of tool-specific formalisms or API; for example, while XSLT is a textual language, it is possible to consider a MOF-compliant metamodel of XSLT; the PST is then an XSLT model which is actually serialized behind the scene to its XML representation.

4 Conclusion

Model transformations are increasingly seen as vital assets that must be managed with sound software engineering principles: they must be analyzed, designed, implemented, tested, maintained and be subject to configuration management. We

believe that transformations should be first-class models in the MDA world. We propose to adopt the object-oriented approach and to leverage the expressive power of UML as a metamodel defining the transformation language. This idea has been implemented in our Model Transformation Language (MTL, available for downloading from modelware.inria.fr), which is actually an executable subset of the UML. It is open source software developped in the framework of the CARROLL research program[1], launched by Thales and two French public research laboratories: CEA (Commissariat à l'Énergie Atomique) and INRIA (Institut National de Recherche en Informatique et en Automatique).

References

1. Jean Bézivin, Nicolas Farcet, Jean-Marc Jézéquel, Benoît Langlois, and Damien Pollet. Reflective model driven engineering. In G. Booch P. Stevens, J. Whittle, editor, *Proceedings of UML 2003*, volume 2863 of *LNCS*, pages 175–189, San Francisco, October 2003. Springer.
2. W.M. Ho, J.-M. Jézéquel, F. Pennaneac'h, and N. Plouzeau. A toolkit for weaving aspect oriented UML designs. In *Proceedings of 1st ACM International Conference on Aspect Oriented Software Development, AOSD 2002*, Enschede, The Netherlands, April 2002.
3. Jean-Marc Jézéquel, Alain Le Guennec, and François Pennaneac'h. Validating distributed software modelled with UML. In *Proc. Int. Workshop UML98, Mulhouse, France*, June 1998.
4. Clémentine Nebut, Simon Pickin, Yves Le traon, and Jean-Marc Jézéquel. Automated requirements-based generation of test cases for product families. In *Proc. of the 18th IEEE International Conference on Automated Software Engineering (ASE'03)*, 2003.
5. OMG/MOF. Meta Object Facility (MOF) specification. OMG Document ad/97-08-14, September 1997.
6. R. Soley and the OMG Staff. Model-Driven Architecture. OMG Document, November 2000.
7. G. Sunyé, A. Le Guennec, and J.-M. Jézéquel. Design pattern application in UML. In E. Bertino, editor, *ECOOP'2000 proceedings*, number 1850 in LNCS, pages 44–62. Springer Verlag, June 2000.
8. Gerson Sunyé, Damien Pollet, Yves Le Traon, and Jean-Marc Jézéquel. Refactoring UML models. In *Proceedings of UML 2001*, volume 2185 of *LNCS*, pages 134–148. Springer Verlag, 2001.
9. Yves Le Traon, Thierry Jéron, Jean-Marc Jézéquel, and Pierre Morel. Efficient OO integration and regression testing. *IEEE Trans. on Reliability*, 49(1):12–25, March 2000.
10. T. Ziadi, L. Hélouët, and J.-M. Jézéquel. Revisiting statechart synthesis with an algebraic approach. In *26th International Conference on Software Engineering (ICSE 04)*, number to be published, 2004.
11. Tewfik Ziadi, Jean-Marc Jézéquel, and Frédéric Fondement. Product line derivation with uml. In *Proceedings Software Variability Management Workshop, University of Groningen Departement of Mathematics and Computing Science*, February 2003.

[1] See www.carroll-research.org for more details.