The TopModL Initiative

Pierre-Alain Muller
pa.muller@uha.fr
INRIA/Irisa – Université de Rennes – France

Cédric Dumoulin
cedric.dumoulin@lifl.fr
LIFL – Université de Lille – France

Frédéric Fondement
frederic.fondement@epfl.ch
EPFL/IC/LGL – Lausanne – Switzerland

Michel Hassenforder
m.hassenforder@uha.fr
MIPS/LSI – Université de Haute-Alsace – France

Abstract

We believe that there is a very strong need for an environment to support research and experiments on model-driven engineering. Therefore we have started the TopModL project, an open-source initiative, with the goal of building a development community to provide:

- an executable environment for quick and easy experimentation,
- a set of source files and a compilation tool chain,
- a web portal to share artefacts developed by the community.

The aim of TopModL is to help the model-engineering research community by providing the quickest path between a research idea and a running prototype. In addition, we also want to identify all the possible contributions, understand how to make it easy to integrate existing components, while maintaining architectural integrity. At the time of writing we have almost completed the bootstrap phase (known as Blackhole), which means that we can model TopModL and generate TopModL with TopModL.

Beyond this first phase, it is now of paramount importance to gather the best possible description of the requirements of the community involved in model-driven engineering to further develop TopModL, and also to make sure that we are able to reuse or federate existing efforts or goodwill.

This paper is more intended to set up a basis for a constructive discussion than to offer definitive answers and closed solutions.

Introduction – About model-driven engineering

At the end of the year 2000, the OMG proposed a radical move from object composition to model transformation[1], and started to promote MDA[2] (Model-Driven Architecture) a model-driven engineering framework to manipulate both PIMs (Platform Independent Models) and PSMs (Platform Specific Models). The OMG also defined a four level meta-modeling architecture, and UML was elected to play a key role in this architecture, being both a general purpose modeling language, and (for its core part) a language to define metamodels. As MDA will become mainstream, more and more specific metamodels will have to be defined, to address domain specific modeling requirements. Examples of such metamodels are CWM (Common Warehouse Metamodel) and SPEM (Software Process Engineering Metamodel). It is likely that MDA will be applied to a wide range of different domains.

While preeminent in the current days, MDA is only a specific case, and we suggest considering model-driven engineering as a wider research field, which includes the study of the following issues:

- What are the essential entities for model-driven engineering?
- How to classify these entities?
- How to translate models into executable code?
- What are the essential operations of model-driven engineering?
- How to classify these operations?
- How to separate and merge the business and platform aspects?
- How to build transformation systems?
- How to maintain a model-driven application?
- How to migrate a legacy application to a model-driven application?
- How to integrate conventional application with model-driven applications?
- Which abstractions and notations should be used to support the previous points?
- What kind of supporting environment should be defined?

Obviously the scope of model-driven engineering is wide and a lot of work is still ahead of us. We believe that a common research platform which would provide the fundamentals services required by model-driven engineering would significantly contribute to the advance of research in this field.

Basic principles of model-driven engineering

The point is to identify the fundamental characteristics upon which to build model-driven engineering. In our specific case, this means identify the requirements for a supporting environment dedicated to model-driven experiments.

The fundamental principles identified so far by the TopModL initiative are:

- The fact that everything is a model. For TopModL, models are first-class entities; everything is expressed explicitly in terms of models, including business models, platform models, executable models, debugging models, trace models, transformation models, process models…

- The notions of languages, models and roles. A model is expressed in a language; this language is a model which plays the role of meta-model for the models expressed in that language.

- The fact that TopModL itself is a model. We want TopModL to be model-driven; we want everything in TopModL to be explicit and customizable.

- The fact that everything is explicit, including the meta-modeling framework. For instance TopModL does not require the M3 to be MOF, it does not even require a 4 layer meta-modeling architecture.

- The independence versus the model repository. We want to have a uniform access to several repositories including EMF[3], MDR[4], or XDE.

These principles shape the requirements for the services to be provided by TopModL.

Toward a research platform for Model-Driven Engineering

The goal of TopModL is to provide an infrastructure for model-driven engineering, including a reference implementation of MDA as promoted by the OMG. TopModL wants to act as a facilitator for researchers in model-driven engineering in connection with other fields including generative programming, graph transformations, domain specific languages (DSLs), or aspects.

The basic services offered by TopModL are:

- Model (meta-model) persistence
- Model (meta-model) serialization in XMI (XML Meta-data Interface)
- JMI (Java Meta-Data) interfaces generation for model (meta-model) manipulation
- Visual edition of models (meta-models)
- Model-Driven parameterization of TopModL
- Model-Driven textual editor generation
- Model-Driven visual editor generation
- OCL evaluation during meta-model and model edition
- Code (Java, SQL) generation
- Model transformation

The TopModL artefacts include:

- A set of source files (Java, XML, text…) and model-driven compilation tool chain to bootstrap TopModL.

- An executable environment. The first release (known as Blackhole) allows the visual edition of meta-model which conforms to the UML Infrastructure (TopModL by default uses the UML Infrastructure as M3).

- A model-driven web portal to share artefacts developed by the community (libraries of meta-models, profiles, models and transformations…).

The first release of TopModL (Blackhole) contains all the artefacts required to bootstrap TopModL. Bootstrapping means being able to use TopModL to model and generate TopModL itself.

Blackhole - Technical architecture of the bootstrap

The technical infrastructure of TopModL has been a recurring concern since the inception of the TopModL initiative (November 2003). The basic question is: *What should be the existing technology (if any) onto which to layer the new developments to be done in the initiative?*

The initial partners acknowledged that the answer to that question would not be obvious, and considered that trying to determine upfront the best technology would be counterproductive, and would significantly delay the timeliness delivery of the TopModL artefacts.

The decision was then taken to use the technology that the partners felt the most comfortable with at the time of the launch of the project, and not to wait for the next wonderful technology yet to come. The direct advantage of this approach was to accumulate practical experience and then to be able to make decisions based on explicit knowledge - rather than informal feelings - if a new technical element had to be incorporated.

The technologies and standards that were chosen for the bootstrap phase include:

- Java for the programming language
- JMI for the metadata API
- MDR for the model repository
- GEF/SWT for the graphical interface
- The Eclipse framework and plugging mechanism
- The UML Infrastructure as M3

It is the intent of the TopModL partners to take any appropriate action to simplify the potential transition from one technology to another one. Technology independence is achieved as much as possible by:

- Sticking to established standards when these standards are available (JMI, XMI, MOF, UML…)
- Defining neutral pivot on top of alternative technical solutions, for instance to be independent of the repositories.

The following picture describes the bootstrap process of TopModL.

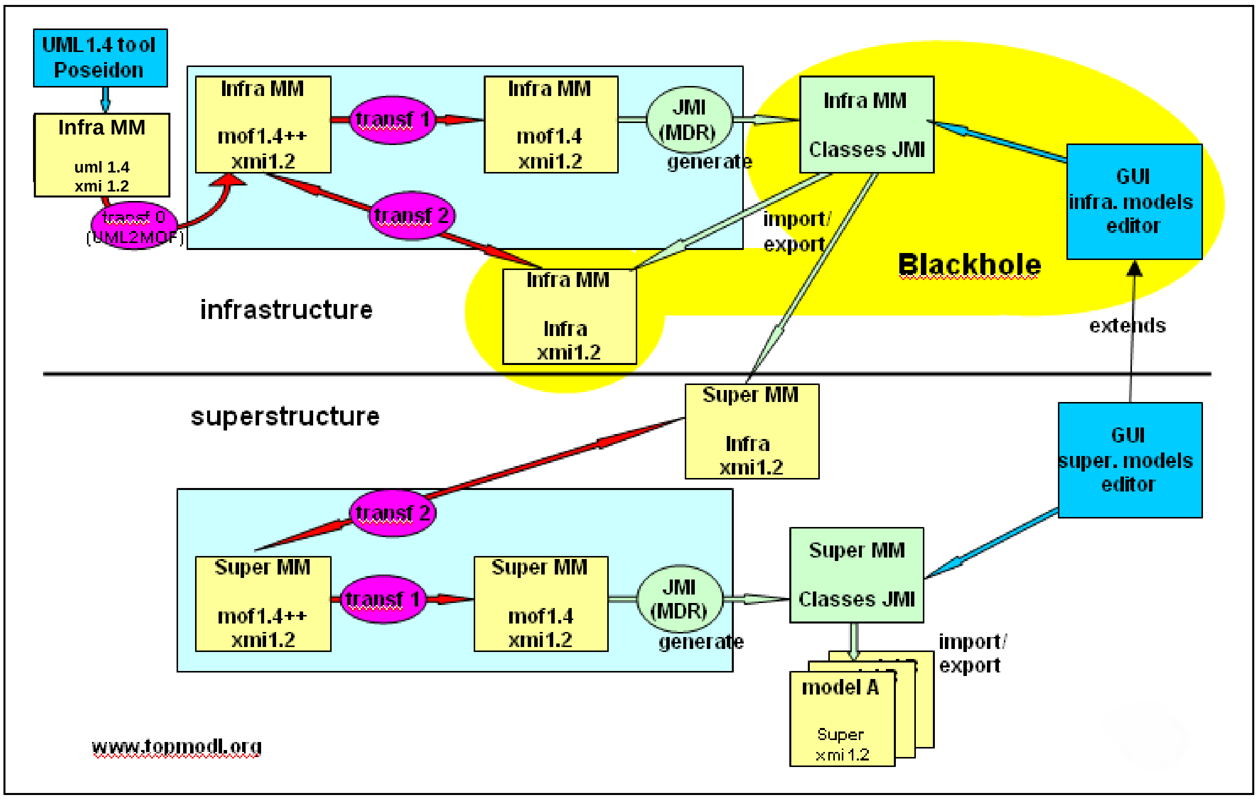

Figure 1 : Bootstrapping TopModL

One of the concerns of TopModL is to make explicit the meta-modeling framework, and not to constrain the usage of a meta-modeling language at the M3 modeling level. There are mainly two reasons for that. The first reason is that TopModL, as a research-oriented tool, should be able to evaluate new possible meta-modeling languages and new meta-modeling constructs. The second reason is that there are already many different meta-modeling languages available today, and that there will be even more in the future (MOF 1.3, MOF 1.4, MOF 2.0, EMF ECore, …), and so TopModL should be able to deal with all of them. As an example, the first M3 language that we want Blackhole to support is the UML 2.0 Infrastructure. In addition, there is also an architectural problem, as model repositories are relying on a given meta-modeling language for both meta-model definition and persistence (actually to shape the schema of the repository). For instance, MDR requires metamodels to be MOF 1.4 described. Again, to promote research one of the most important issue is to abstract away this MOF platform. This was possible by introducing model transformations.

The first step is to model the meta-modeling language using UML class diagram in a UML CASE tool such as the Community Edition of Poseidon[5]. Then, this model is exported in an XMI file as a UML 1.4 model, which can be taken as input by the MDR *UML2MOF* model transformation, which promotes UML models to MOF models. A problem is that the Infrastructure is making an extensive use of constructs that do not have their equivalent in MOF 1.4 (like package merge or refinement/subsetting of association-ends) such that the MDR UML2MOF had to be enhanced to interpret such constructs. Actually, this was done directly on the XMI resulting from the transformation, on which we have reported occurrences of these constructs, thus extending MOF 1.4 a little bit to what we called MOF 1.4++. As these constructs cannot be understood by MOF 1.4 implementation such as MDR, they are interpreted by the *transf1* model transformation, for the meta-modeling language (Infrastructure in this case) to be represented as MOF 1.4. This makes possible to load the

Infrastructure as a metamodel in the MDR tool, which means to be able to generate the JMI interfaces specific to the Infrastructure, to store Infrastructure models, and to be able to serialize/de-serialize Infrastructure models to XMI 1.2.

We have also developed a visual editor which connects to this repository, and we can this way edit metamodels (which conform to the UML Infrastructure). This editor is currently hand-coded, but it is our intent to generate it from models, in a similar way of what is done in Netsilon[6].

At this point, TopModL is able to edit and maintain Infrastructure models, thus it is possible to model the Infrastructure in TopModL. The result is the Infrastructure modeled as an Infrastructure model. The *transf2* model transformation was developed to transform Infrastructure models to MOF 1.4++ models, so that the Infrastructure modeled in TopModL can be automatically translated to a MOF 1.4++ model. By reusing this MOF 1.4++ model of the Infrastrucure, it is possible to perform again the process, this time using TopModL instead of Poseidon: TopModL is at this phase bootstrapped, i.e. TopModL is modeled and designed using TopModL. Obviously, the bootstrapping process may be repeated for any modeling language supposed to play the role of a meta-modeling language.

The second part of the picture shows how the approach can then be used to generate a tool for another modeling language such as the UML 2.0 Superstructure. Basically, the process is the same: modeling the new modeling language in TopModL as an Infrastructure model, and using the *transf2* to regenerate a TopModL intended to edit and maintain models in the new modeling language. Again, the issue of modeling the concrete syntax of the new modeling language, and the behavior of the editor dedicated to it is not yet solved and needs further research. The following paragraph further motivates such CASE tool generation.

Toward a model-driven CASE tool for the UML

TopModL is a meta-environment which can be used to realize model-driven CASE tools. The TopModL development community has considered that realizing a model-driven UML CASE tool would be an excellent use case for TopModL, and would also allow bypassing the current limitations of the available tools, most notably:

- The fact that no commercial tool fully implements the OMG standards, mainly because the first generation of UML CASE tools do not rely on an explicit description of the meta-models, but on hand-coded implementations, often themselves derived from earlier modeling tools (designed before the advent of the UML). Our experience also shows that CASE tools vendors do not have the resources to realize quick update of their tools, or to satisfy specific demands (to validate a research advance or to support a specific extension, or even simply to implement a standard).

- The lack of model-driven open-source tools, mainly because all the open-source efforts are based on programming and not modeling. TopModL is an open-model project before being an open-source project.

- The hard-wired nature of existing tools which embed a lot of decisions in their code and are therefore very difficult to customize. When buried in the code, decisions are implicit, as opposed to explicit when they are expressed in models and meta-models

In this context, TopModL will develop a new kind of UML CASE tool, entirely model-driven, with all the decisions made explicit by means of models. It is more accurate to talk about a model compilation tool chain, as the resulting CASE tool will always be conforming to a user-defined set of meta-models which will define its context. These capacities of customization will concern:

- The internal repository, generated in conformance to the meta-models referenced by the context.

- The model serialization (import-export via XMI)

- The visual editors, derived from a generic graph editor by explicit customization (this feature will be of special interest to realize domain specific languages).

- The textual editors (with syntactic and semantic completion) generated from meta-models (the abstract syntax) and mapped on a concrete syntax (also expressed via models).

- The overall behavior of the tool which will be described in a process model conforming to SPEM (Software Process Engineering Model).

Moreover, all the generated (UML or other modeling language) CASE tools will present the ability to interoperate, thus allowing for instance model transformations. One possible way for generating code is to generate two TopModL CASE tools for a modeling language (e.g. UML) and for the desired target language (e.g. Java) by using its abstract syntax as a metamodel. A model transformation from the modeling language to the target language would actually transform a user-specific model to a target-language model. By taking advantage of the textual concrete syntax of the target language, the latter model may be expressed as a set of files using the target language syntax. This is another reason for defining precisely concrete syntaxes of modeling language explicitly.

Related works

There are many related works, which share a common vision with the TopModL initiative. We summarize some of these approaches below:

- Meta CASE tools including Metaedit+[7], Dome[8] or GME[9], provide customizable CASE tools, however they are fairly closed in the sense that they are neither open-source, nor themselves model-driven.

- Dedicated model-driven tools, which generate specific applications, like Netsilon[6] for Web information systems, Accord/UML[10] for embedded distributed real-time systems.

- OCL based tools, including KMF[11] which generates modeling tools from the definition of modeling languages expressed as meta-models, or Octopus[12] an Eclipse plug-in OCLE[13] or the Dresden OCL toolkit[14], which are able to check the syntax of OCL expressions, as well as the types and correct use of model elements like association roles and attributes.

- Meta-modeling frameworks like Eclipse EMF[3], Netbeans MDR[4] or Coral[15] which offer model persistence, model serialization and programmatic access to models via an API, or integration technologies like ModelBus[16]. These frameworks provide part of the functionalities required by TopModL.

- Open-source modeling tools, including ArgoUML[17] of Fujuba[18] which offer significant feature at the M1 level, but lack customization at the M2 level.

One of the goals of TopModL is to understand how to reuse or leverage these related works, and to find how to integrate them as much as possible in a research platform for model-driven engineering.

Conclusion

The TopModL open-source initiative has been launched with the goal of providing tool support to the model-driven engineering research community.

The TopModL initiative groups a development community, a web portal to share the artefacts developed by the community, a set of source files and an executable program for meta-modeling.

TopModL is itself a model-driven application, and a first phase known as Blackhole delivers the bootstrap of TopModL, which means that TopModL is modeled and generated with TopModL.

TopModL makes all decisions explicit, including the meta-modeling framework, by using models. However, some modeling language will still have to be defined, for instance to express concrete syntax or model transformation. Having such a variable meta-modeling framework may raise questions, for instance about the universality of the 4 layers meta-modeling architecture. TopModL will provide the opportunity to experience a step further the meta-modeling activity, and will point out problems that may have not been already reported, like the necessity of modeling concrete syntaxes, or edition facilities.

References

[1] J. Bézivin, *"From Object Composition to Model Transformation with the MDA"*, in proceedings of TOOLS'2001. IEEE Press Tools#39, pp. 350-354 . (August 2001).
[2] Object Management Group, Inc., *"MDA Guide 1.0.1"*, omg/2003-06-01, June 2003.
[3] Eclipse EMF, web site http://www.eclipse.org/emf/
[4] Netbeans MDR, web site http://mdr.netbeans.org/
[5] Poseidon web site http://www.gentleware.de
[6] P.-A. Muller, P. Studer, and J. Bezivin, *"Platform Independent Web Application Modeling"*, in P. Stevens et al. (Eds): UML 2003, LNCS 2863, pp. 220-233, 2003.
[7] R. Pohjonen, "Boosting Embedded Systems Development with Domain-Specific Modeling", in RTC Magazine, April, pp. 57-61, 2003
[8] Honeywell, 1992, *"DOME Guide"*, available from "www.htc.honeywell.com/dome/"
[9] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, *"The Generic Modeling Environment"*, Proceedings of the IEEE International Workshop on Intelligent Signal Processing, WISP'2001, Budapest, Hungary, may 24-25, 2001
[10] S. Gérard, N. S. Voros, C. Koulamas, and F. Terrier, *"Efficient System Modeling of Complex Real-Time Industrial Networks Using the ACCORD UML Methodology"*, DIPES'2000,Paderborn, Germany, 2000.
[11] Kent Metamodeling Framework, web site http://www.cs.kent.ac.uk/projects/kmf/index.html
[12] Octopus web site http://www.klasse.nl/ocl/octopus-intro.html
[13] OCLE web site http://lci.cs.ubbcluj.ro/ocle/
[14] Dresden OCL toolkit web site http://dresden-ocl.sourceforge.net/
[15] Coral, web site http://mde.abo.fi/tools/Coral/
[16] X. Blanc, M.-P. Gervais, P. Sriplakich, "Model Bus : Towards the interoperability of modelling tools", MDAFA'04, Linköping, June 10-11, 2004
[17] ArgoUML web site http://argouml.tigris.org/
[18] Fujaba web site http://wwwcs.upb.de/cs/fujaba/index.html