

WHY TO DO WITHOUT MODEL DRIVEN ARCHITECTURE IN EMBEDDED SYSTEM CODESIGN?

¹Jean-Luc Dekeyser, ¹Philippe Marquet ¹Samy Meftali,
¹Cédric Dumoulin, ¹Pierre Boulet and ²Smail Niar

¹Jean-Luc.Dekeyser@lifl.fr

¹INRIA Futurs DaRT, Université des Sciences et Technologies de Lille,
Cité Scientifique, 59655 Villeneuve d'Ascq Cedex, France

²INRIA Futurs DaRT, Université de Valenciennes et du Hainaut-Cambrésis
Le Mont Houy, 59313 Valenciennes Cedex 9, France

ABSTRACT

The Model-Driven architecture is an initiative by the Object Management Group (OMG) to define an approach to software development based on modeling and automated mapping of models to implementations. The basic MDA pattern involves the definition of a platform-independent model (PIM) and its automated mapping to one or more platform-specific models (PSMs). By defining different PIM and PSM dedicated to embedded systems, we will show the benefits of using the MDA approach in System on Chip codesign. From UML 2.0 [1] profiles to System C or VHDL codes, the same model transformation engine is used with different rules expressed in XML.

1. INTRODUCTION

Because of the vast scope of the encountered problems, of the quick evolution of the architectures, a variety of dialects were born, without relieving the programmer of the usual constraints of code development. Portability of an application from one language to another (a new one for example) increases the workload of the programmer. This drawback is also true for the development of embedded applications. It is even worse, because the number of abstraction levels has to be added to the diversity of the languages. It is essential to associate a target hardware architecture model to the application specification model, and to introduce as well a relationship between them. In practice, these two models are always different; they are often expressed in two different languages.

From this experience, one can derive some principles for the design of the next generation of environments for embedded application development. They are:

- To refrain from designing programming languages to express the two different models, application and hardware architecture.
- To profit from all the new systems dedicated to simulation or synthesis without having to reformatize

these two models.

- To use a single modeling environment possibly supporting a visual specification.
- To benefit from standard formats for exchange and storage.
- To be able to express transformation rules from model to model. Possibly the transformation tools could be generated automatically from this expression.

2. MDA IN SOC CODESIGN

We believe that the Model Driven Architecture [2, 3] can enable us to propose a new method of system design respecting these principles. Indeed, it is based on the common UML modeling language to model all kinds of artifacts. The clear separation between the models and the platforms makes it easy to switch to a new technology while re-using the old designs. This may even be done automatically provided the right tools. The MDA is the OMG proposed approach for system development. It primarily focuses on software development, but can be applied to any system developments. The MDA is based on models describing the systems to be built. A system description is made of numerous models, each model representing a different level of abstraction. The modeled system can be deployed on one or more platforms via model to model transformations.

A key point of the MDA is the transformation between models. The transformations allow to go from one model at a given abstraction level to another model at another level, and to keep the different models synchronized. Related models are described by their metamodels, on which we can define some mapping rules describing how concepts from one metamodel are to be mapped on the concepts of the other metamodel. From these mapping rules

we deduce the transformations between any models conforming to the metamodels.

SoC (System-on-Chip) can be considered as a particular case of embedded systems. SoC design covers a lot of different viewpoints including as much the application modeling by the aggregation of functional components, as the assembly of existing physical components, as the verification and the simulation of the modeled system, as the synthesis of a complete end-product integrated into a single chip. As a rule a SoC includes programmable processors, memory units (data/instructions), interconnection mechanisms and hardware functional units (Digital Signal Processors, application specific circuits). These components can be generated for a particular application; they can also be obtained from IP (Intellectual Property) providers. The ability to re-use software or hardware components is without any doubt a major asset for a codesign system.

The multiplicity of the abstraction levels is appropriate to the modeling approach. The information is used with a different viewpoint for each abstraction level. This information is defined only once in a single model. The links or transformation rules between the abstraction levels permit the re-use of the concepts for a different purpose.

Our proposal is partially based upon the concepts of the “Y-chart” [4]. The MDA contributes to express the model transformations which correspond to successive refinements between the abstraction levels.

Metamodeling brings a set of tools which will enable us to specify our application and hardware architecture models using UML tools, to reuse functional and physical IPs, to ensure refinements between abstraction levels via mapping rules, to ensure interoperability between the different abstraction levels used in a same codesign, and to ensure, through the use of standards, the opening to other tools, like verification tools.

3. METAMODELS FOR THE “Y” DESIGN

The application and hardware architecture are described by different metamodels. Some concepts from these two metamodels are similar in order to unify and so simplify their understanding and use. Models for application and hardware architecture may be done separately (maybe by two different people). At this point, it becomes possible to map the application model on the hardware architecture model. For this purpose we introduce a third metamodel, named association metamodel, to express associations between the functional components and the hardware components. This metamodel imports the two previously presented metamodels.

All the previously defined models, application, architecture and association, are platform independent. No component is associated with an execution, simulation or syn-

thesis technology. Such an association targets a given technology (Java, SystemC RTL, SystemC TLM, VHDL, etc). Once all the components are associated with some technology, the deployment is realized. This is done by the refinement of the PIM association model to the PIM TLM model first (Transaction Level Model), and to the PIM RTL model second (Register Transfer Level).

The diversity of the technologies requires interoperability between abstraction levels, simulation and synthesis languages. For this purpose we define an interoperability metamodel allowing to model interfaces between technologies.

Mapping rules between the deployment metamodel, and interoperability and technology metamodels can be defined to automatically specialize the deployment model to the chosen technologies. From each of the resulting models we could automatically produce the execution/simulation code and the interoperability infrastructure.

The simulation results can lead to a refinement of the application, of the hardware architecture, of the association or of the deployment models. The stages of design could be:

- Separation of application and hardware architecture modeling.
- Association with semi-automatic mapping and scheduling.
- Deployment (choice of simulation or execution level and platform for each component).
- Automatic generation of the various platform specific simulation or execution models.
- Automatic simulation or execution code generation.
- Refinement at the PIM level given the simulation results.

4. THE TRANSFORMATION ENGINE: MODTRANSF

Model to model transformations are at the heart of the MDA approach. Anyone wishing to use MDA in its projects is soon or later facing the question: how to perform the model transformations? There are not so much publicly and freely available tools, and the OMG QVT standardization process is not completed today. To fulfill our needs in model transformations, we have developed ModTransf, a simple but powerful transformation engine. ModTransf developments follow the recommendations done after the review of the first QVT proposals [2] and on the latest proposals [3]. Based on these

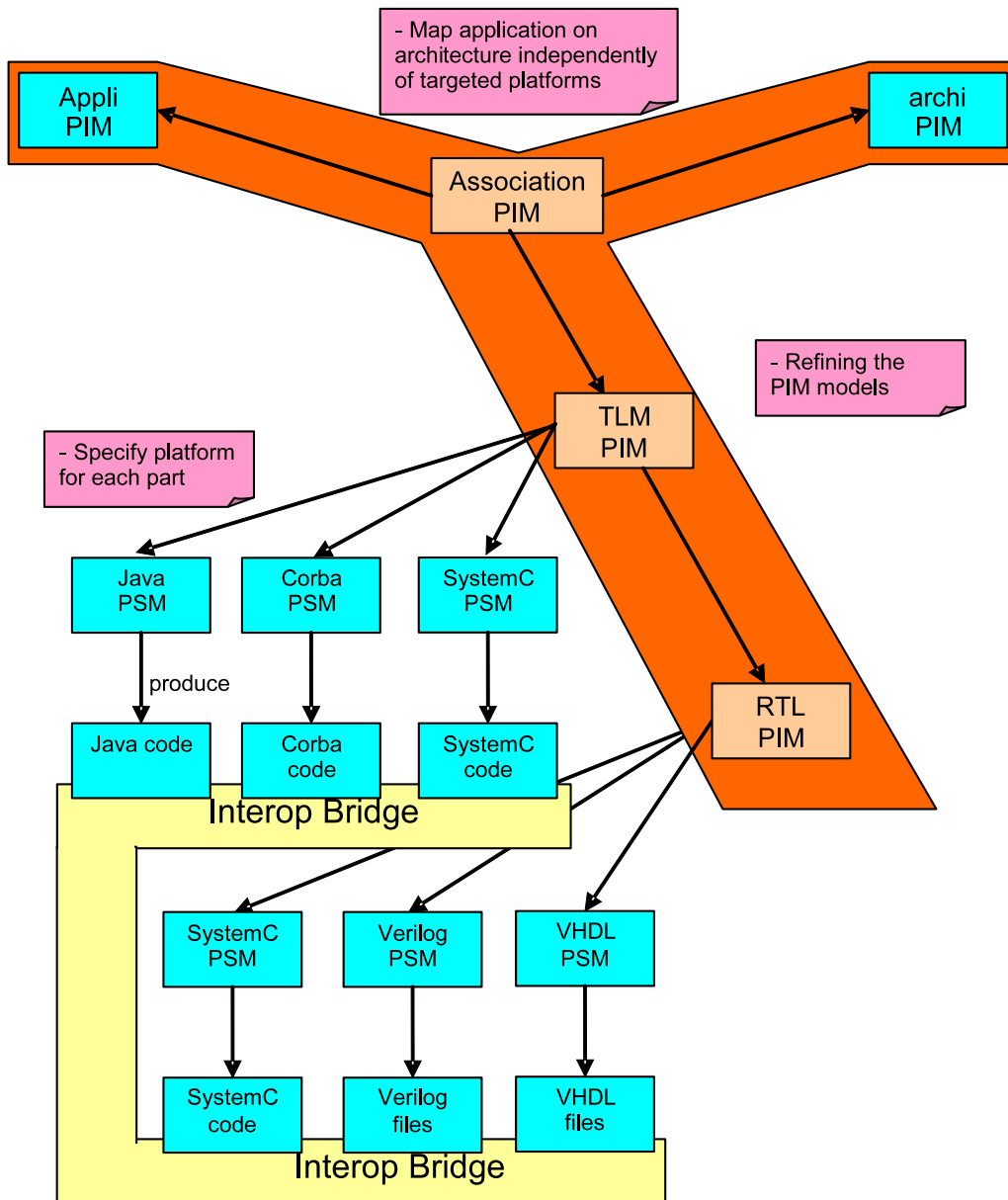


Figure 1: Metamodels for the “Y” design

recommendations and on our needs, we have identified the following requirements for the transformation engine:

- Multi models as inputs and outputs.
- Different kind of models: MOF and JMI based, XML with schema based, graph of objects.
- Simple to use.
- Easy modification of rules to follow metamodel changes.
- Hybrids: Imperative and declarative rules.
- Inheritance for the rules.
- Reversible rules when possible.
- Customizable, to do experimentations.
- Code generation.
- Free and Open-Sources.

The proposed solution fulfills all these needs: ModTransf is a rule based engine taking one or more models as inputs and producing one or more models as outputs. The rules can be expressed using an XML syntax and can be declarative as well as imperative. A transformation is done by submitting a concept to the engine. The engine then searches the more appropriate transformation rule for this concept and applies it to produce the corresponding result concept. The rule describes how properties of the input concept should be mapped, after a transformation, to the properties of the output concept.

The code generation follows the same principle, but the output concept creation is replaced by code generation performed with a template mechanism. A rule specifies one or more template to use, and each template contains holes replaced by the values of the input concepts.

The ModTransf engine is an Open Source project available on our web site [5].

5. CONCLUSION

Model driven in codesign brings the same benefits than in software design. Reuse of hardware and software components becomes easier, they are defined in UML. Investment in model development is profitable, you do not need to rewrite old components in new languages, just transform them. IP integration is possible thanks to interoperability metamodel definition. New synthesis or simulation platforms are easily integrated without to rewrite all the models. Verification and refactoring can be applied at PIM level independently of the platform specificities.

But we still have to develop the metamodel for each part of the design, and for each abstraction level. The transformation rules have to be specified and implemented in the transformation tool.

Software engineering brings to the embedded system designers the opportunity to develop systems reliable, reusable and verifiable. We'd be a fool not to take it!

6. REFERENCES

- [1] Object Management Group, Inc., Ed., (*UML 2.0): Superstructure Draft Adopted Specification*, <http://www.omg.org/cgi-bin/doc?ptc/03-07-06/>, July 2003.
- [2] OMG Architecture Board, "Model driven architecture (MDA)," Tech. Rep. ormsc/2001-07-01, OMG, 2001.
- [3] Joaquin Miller and Jishnu Mukerji, Eds., *MDA Guide (Draft Version 0.2)*, {<http://www.omg.org/docs/ab/03-01-03.pdf>}, 2003.
- [4] D. D. Gajski and R. Kuhn, "Guest editor introduction: New VLSI-tools," *IEEE Computer*, vol. 16, no. 12, pp. 11-14, Dec. 1983.
- [5] Cédric Dumoulin, "ModTransf: A model to model transformation engine," Nov. 2004, <http://www.lifl.fr/west/modtransf>.
- [6] T.Gardner, C.Griffin, A. Koehler, and R.Hauser, "A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard," OMG document 03-08-02, July 2003, OMG document. Review of QVT proposals.